THE DESIGN, ARCHITECTURE, IMPLEMENTATION AND ARTIFICIAL INTELLIGENCE SUPPORT OF A SMART BIO AND CULTURAL GUIDING SYSTEM



by

GEORGE TSAMIS

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

FACULTY OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

THIS DISSERTATION IS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FEBRUARY 2026

Dedicated to my Family (Vlasis, Anna and Chrysa Tsamis), my Wife (Maria Rousaki), my Grandma (Hara Karampolli), my Aunt (Maria Karampolli), my Mentors (John and Mairi Dryllerakis) and to my Professor (Dr. Nikolaos Papadakis)

If I have seen further it is by standing on the shoulders of Giants.

Sir Isaac Newton

The famous declaration of humility and acknowledgment that his ground breaking scientific work was built upon the foundation of previous scientists.

DECLARATION

This dissertation is the result of my own work and includes nothing, which is the outcome of work done in collaboration except where specifically indicated in the text. It has not been previously submitted, in part or whole, to any university of institution for any degree, diploma, or other qualification.

Signed:

Date: 29/09/2025

GEORGE TSAMIS

ABSTRACT

This study presents the design, architecture, implementation, and evaluation of a smart bio- and cultural-guiding system that integrates artificial intelligence–driven personalization with augmented reality (AR) guidance on commodity Android devices. Unlike conventional mobile guiding applications that rely on static proximity-based listings or rule-based user profiles, the proposed platform introduces a neural-network-based recommender and an AR-assisted navigation interface, aiming to measurably improve recommendation relevance, spatial understanding, and overall user experience without requiring human guide intervention.

The novelty of the proposed approach is demonstrated through a rigorously separated dual-method evaluation framework combining synthetic-data-based system analysis with controlled observational usability studies. Quantitative results show that the AI recommender significantly outperforms proximity-based and rule-based baselines, achieving improvements of up to +19 percentage points in Precision@5 (0.41 vs. 0.22) and +21 points in NDCG@5 (0.45 vs. 0.24), confirming superior ranking quality for personalized points of interest. User classification accuracy is also enhanced through a multi-layer neural network, reaching 85% accuracy compared to 78% for a K-Nearest-Neighbors baseline.

Furthermore, controlled usability experiments demonstrate that AR-based guidance yields substantial performance gains in real-world navigation tasks. In spatial POI location tasks, the proposed system increases task success rates from 78% to 83% while reducing median completion time by 38% (11 s to 6.8 s), indicating a marked reduction in cognitive load when translating digital guidance into physical navigation. Overall usability is rated as excellent, with a System Usability Scale (SUS) score of 82.5, compared to 68.0 for a non-AR baseline application.

Despite incorporating computationally intensive AR and AI components, system profiling confirms practical deploy ability on mid-range Android devices, with average CPU utilization remaining below 25% in standard interaction scenarios and peak usage limited to short-lived AR and vision-related operations. Network latency for core services consistently remains within sub-second thresholds, supporting responsive user interaction even in bandwidth-constrained environments.

In practical terms, the proposed system provides a scalable technological foundation for sustainable bio- and cultural-tourism applications, enabling data-driven tourism management, enhanced visitor education, and improved engagement with natural and cultural heritage sites. By empirically demonstrating that AI personalization and AR guidance deliver measurable benefits in recommendation quality, navigation efficiency, and usability, this work contributes validated evidence that intelligent, augmented mobile guiding systems can effectively replace traditional human-guided experiences while preserving ecological and cultural sensitivity.

Keywords: Design, Architecture, Implementation, Artificial Intelligence, Bio, Culture, Guiding System

ΠΕΡΙΛΗΨΗ

Η παρούσα μελέτη παρουσιάζει τον σχεδιασμό, την αρχιτεκτονική, την υλοποίηση και την αξιολόγηση ενός έξυπνου συστήματος βιολογικής και πολιτιστικής ξενάγησης, το οποίο ενσωματώνει μηχανισμούς εξατομίκευσης βασισμένους σε τεχνητή νοημοσύνη και καθοδήγηση μέσω επαυξημένης πραγματικότητας (Augmented Reality – AR) σε φορητές συσκευές Android. Σε αντίθεση με τις συμβατικές εφαρμογές ξενάγησης που βασίζονται αποκλειστικά στη γεωγραφική εγγύτητα ή σε στατικά, κανονοκεντρικά προφίλ χρηστών, το προτεινόμενο σύστημα αξιοποιεί νευρωνικά δίκτυα για δυναμική προσαρμογή των προτεινόμενων σημείων ενδιαφέροντος και AR διεπαφές για βελτιωμένη χωρική κατανόηση και πλοήγηση, χωρίς την ανάγκη ανθρώπινου ξεναγού.

Η καινοτομία της προσέγγισης τεκμηριώνεται μέσω ενός αυστηρά διαχωρισμένου πλαισίου αξιολόγησης δύο επιπέδων, το οποίο συνδυάζει συνθετικά δεδομένα για τεχνική και αλγοριθμική αποτίμηση με ελεγχόμενη παρατηρησιακή μελέτη ευχρηστίας. Τα ποσοτικά αποτελέσματα δείχνουν ότι ο προτεινόμενος μηχανισμός εξατομίκευσης υπερτερεί σημαντικά έναντι προσεγγίσεων βάσει εγγύτητας και κανόνων, επιτυγχάνοντας βελτίωση έως και 19 ποσοστιαίες μονάδες στο Precision@5 (0,41 έναντι 0,22) και 21 μονάδες στο NDCG@5 (0,45 έναντι 0,24), γεγονός που αποδεικνύει ανώτερη ποιότητα κατάταξης των σημείων ενδιαφέροντος. Παράλληλα, η ταξινόμηση χρηστών μέσω πολυεπίπεδου νευρωνικού δικτύου επιτυγχάνει ακρίβεια 85%, έναντι 78% μιας προσέγγισης K-Nearest Neighbors.

Η μελέτη ευχρηστίας καταδεικνύει ότι η καθοδήγηση με επαυξημένη πραγματικότητα επιφέρει ουσιαστικά οφέλη σε εργασίες χωρικής πλοήγησης. Συγκεκριμένα, σε εργασίες εντοπισμού σημείων ενδιαφέροντος, το προτεινόμενο σύστημα αυξάνει το ποσοστό επιτυχίας από 78% σε 83%, ενώ μειώνει τον διάμεσο χρόνο ολοκλήρωσης κατά 38% (από 11 s σε 6.8 s), υποδεικνύοντας σημαντική μείωση του γνωστικού φορτίου των χρηστών. Η συνολική ευχρηστία αξιολογείται ως εξαιρετική, με δείκτη SUS 82,5, σε σύγκριση με 68,0 για εφαρμογή χωρίς AR.

Παρά την ενσωμάτωση υπολογιστικά απαιτητικών λειτουργιών AR και τεχνητής νοημοσύνης, η ανάλυση απόδοσης επιβεβαιώνει την πρακτική δυνατότητα ανάπτυξης του συστήματος σε συσκευές μεσαίας κατηγορίας, με μέση χρήση CPU κάτω από 25% σε τυπικά σενάρια αλληλεπίδρασης και παροδικές μόνο αιχμές κατά τη χρήση AR. Οι καθυστερήσεις δικτύου παραμένουν σταθερά κάτω του ενός δευτερολέπτου,

διασφαλίζοντας άμεση απόκριση ακόμη και σε περιοχές με περιορισμένη συνδεσιμότητα.

Συνολικά, το προτεινόμενο σύστημα παρέχει μια επεκτάσιμη και τεκμηριωμένη τεχνολογική βάση για εφαρμογές βιολογικού και πολιτιστικού τουρισμού, υποστηρίζοντας τη βιώσιμη διαχείριση προορισμών, την εκπαιδευτική εμπειρία των επισκεπτών και την ουσιαστική αλληλεπίδραση με φυσική και πολιτιστική κληρονομιά. Τα εμπειρικά αποτελέσματα επιβεβαιώνουν ότι ο συνδυασμός εξατομίκευσης μέσω τεχνητής νοημοσύνης και επαυξημένης πραγματικότητας προσφέρει μετρήσιμα και ουσιαστικά πλεονεκτήματα έναντι παραδοσιακών προσεγγίσεων κινητής ξενάγησης.

Λέξεις Κλειδιά: Σχεδιασμός, Αρχιτεκτονική, Υλοποίηση, Τεχνητή Νοημοσύνη, Βιολογία, Πολιτισμός, Σύστημα Καθοδήγησης

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Professor of the Department of Electrical and Computer Engineering at the Hellenic Mediterranean University, Dr. Nikolaos Papadakis who believed in me, honored me with his trust and remained a co-pilot and a guide in this tremendous journey.

I would like to thank my family, Vlasis, Anna and Chrysa Tsamis, my grandmother Hara Karampolli and my aunt Maria Karampolli for the support and love they have shown me over the years.

I would like to thank my wife, Maria Rousaki, for the love, understanding, support, dedication, patience, encouragement and the interest she showed during my studies.

Last but not least, i would like to thank my mentors, John and Mairi Dryllerakis, for their guidance, interest and assistance throughout my academic career.

CONTENTS

1 INTRODUCTION 1

1.1 THE TOUR GUIDE OPERATORS FIELD AND ITS IMPACT ON THE ECONOMY 1

1.1.1 Overview of the Tour Guide Operators Industry 1

1.1.2 The Structure of the Tour Guide Operator Industry 2

1.1.3 Roles and Responsibilities of Tour Guide Operators 3

1.1.4 Direct Economic Impacts 5

1.1.5 Indirect and Induced Economic Impacts 6

1.1.6 Cultural and Social Economic Benefits 7

1.1.7 Environmental and Regional or National Economic Impacts 8

1.1.8 Challenges Facing Tour Guide Operators 10

1.1.9 The Future of the Tour Guide Operator Field 12

1.2 AUGMENTED REALITY GUIDING SYSTEMS 13

1.2.1 Introducing Augmented Reality Guiding Systems 13

1.2.2 History and Evolution of Augmented Reality Guiding Systems 15

1.2.3 Core Technologies of AR Guiding Systems 16

1.2.4 Types of AR Guiding Systems 18

1.2.5 Fields of Applications for AR Guiding Systems 22

1.2.6 Design Considerations for AR Guiding Systems 27

1.2.7 Benefits and Technical Challenges 33

1.2.8 Case Studies and Real-World Examples of AR Guiding Systems 37

1.2.9 Trends and Future Directions in AR Guiding Systems 39

1.2.10 AR Guiding Systems Conclusions and Final Thoughts 40

1.3 THE ANDROID OPERATING SYSTEM 41

1.3.1 Introducing Android OS 41

1.3.2 The Architecture of Android 43

1.3.3 Android's Key Features and Capabilities 44

1.3.4 Main Advantages and Challenges 45

1.3.5 Future Trends and Innovations in Android 47

1.4 NOVELTY AND CONTRIBUTIONS 48

2 SYSTEM'S ARCHITECTURE AND DESIGN 50

2.1 INTRODUCTION 50

2.2 KEY PRINCIPLES OF SYSTEM ARCHITECTURE 50

2.3 WEBSITE SYSTEM ARCHITECTURE 51

2.4 SMARTPHONE APP SYSTEM ARCHITECTURE 51
2.5 DESIGN CONSIDERATIONS 52
2.6 PERSONAL TOUR GUIDE APPLICATION'S AND WEB PLATFORM'S ARCHITECTURE 53
2.6.1 External Services *54*
2.6.2 Android Libraries *54*
2.6.3 Android Application *55*
2.7 PERSONAL TOUR GUIDE APPLICATION'S DESIGN 55
2.8 FIREBASE UML DIAGRAM 59
2.9 WEBSITES AND APPLICATIONS SECURITY 60
2.9.1 Common Security Threats *61*
2.9.2 Security Best Practices *61*
2.9.3 Importance of Security *61*
2.10 UML CLASS DIAGRAM 63
2.10.1 Components of a Class Diagram *63*
2.10.2 Relationships in Class Diagrams *63*
2.10.3 Importance of Class Diagrams *64*
2.11 UML SEQUENCE DIAGRAM 65
2.11.1 Components of a Sequence Diagram *66*
2.11.2 Purpose of Sequence Diagrams *66*
2.11.3 Main Window and Tour Guide Scenarios using Sequence Diagrams *67*
3 ARTIFICIAL INTELLIGENCE NEURAL NETWORKS ALGORITHMS IN THE FIELD OF USER DETECTION AND USERS' PREFERENCES PREDICTION 69
3.1 INTRODUCTION 70
3.2 RELATED WORK 72
3.2.1 The Data Mining Procedure *72*
3.2.2 Users' Identification Techniques *74*
3.2.3 Current state of Technology on User Identification in Websites or Smartphone Applications *75*
3.3 BENEFITS FROM TARGETED AND SELECTIVE PROMOTION TO SPECIFIC USERS 79
3.3.1 Demonstrating Main Benefits *79*
3.3.2 Selective or Targeted Promotion Areas and Categorizations *80*
3.4 ARTIFICIAL INTELLIGENCE NEURAL NETWORKS AND ITS PRINCIPLES 82
3.4.1 Introducing Artificial Intelligence Mechanisms and Principles *82*
3.4.2 Basic Foundations of Neural Networks *84*

3.4.3 Learning Mechanisms in Neural Networks *85*
3.4.4 Key Principles of Neural Networks *86*
3.4.5 Types of Neural Networks *88*
3.4.6 Neural Networks' Applications *91*
3.4.7 Neural Networks and Future Directions *92*
3.5 USER CLASSIFICATION NEURAL NETWORKING ALGORITHMS 93
3.5.1 Single Layer Neural Network *93*
3.5.2 Multi Layer Neural Network *96*
3.6 IDENTIFYING USER THROUGH NEURAL NETWORKING APPLICATIONS 99
3.6.1 The K Nearest Neighbors (KNN) in Neural Networks *99*
3.6.2 Generic Implementation of the K Nearest Neighbors Neural Network *101*
3.6.3 Users' Identification using K Nearest Neighbors Neural Network *103*
3.7 ACCOUNT PREFERENCES PREDICTIONS USING NEURAL NETWORKING AI IMPLEMENTATION 107
3.7.1 Introducing Account Preferences Prediction *107*
3.7.2 Account Preferences Prediction Mechanisms *107*
3.7.3 Recommender Systems and Similarity Scores *109*
3.7.4 Discovering similar users using collaborative filtering *111*
3.7.5 Predicting Users' Preferences using Similarity Scores Application *112*
3.7.6 Benefits and Ethical Considerations of Predicting Users' Preferences *113*
3.7.7 Conclusions *113*
4 PLATFORM IMPLEMENTATION 116
4.1 RESTFUL APIS AND MYSQL 116
4.1.1 Introduction to RESTful APIs in Android Development *116*
4.1.2 How Android Apps Communicate with RESTful APIs *117*
4.1.3 Authentication and Security in RESTful API Integration *119*
4.1.4 Real-World Use Cases of RESTful APIs in Android Apps *120*
4.1.5 Best Practices for Using RESTful APIs in Android *121*
4.1.6 MySQL and Its Relevance to Android *121*
4.1.7 Integrating MySQL with Android via Web Services *122*
4.1.8 Android and MySQL Use Cases, Benefits, and Best Practices *123*
4.1.9 MySQL & RESTful API, Android Login and Registration with PHP, MySQL, and SQLite *124*
4.1.10 System Architecture *124*
4.2 SQLITE IN ANDROID 139

4.2.1 Introduction to SQLite in Android *139*
4.2.2 Implementing SQLite in Android Apps *139*
4.2.3 Advantages, Challenges, and Best Practices *140*
4.3 GEOLOCATION SYSTEM 141
4.3.1 Introducing GPS and its Evolution *141*
4.3.2 Main Functionalities of GPS and Basic Components *142*
4.3.3 Applications of GPS *143*
4.3.4 Limitations and Future Developments *143*
4.3.5 GPS and Android Devices *144*
4.4 INTERACTIVE MAPS 147
4.4.1 Introduction to OpenStreetMap *147*
4.4.2 Integrating OpenStreetMap with Android *147*
4.4.3 Use Cases and Applications on Android *148*
4.4.4 Benefits of Using OSM on Android *148*
4.4.5 Challenges and Future Potential *149*
4.4.6 Implementing osmdroid in Android with Java and osmbonuspack library *150*
4.4.7 Android Project setup for OSM *151*
4.4.8 Markers, InfoWindows, and basic overlays of osmbonuspack *154*
4.5 QR CODE 158
4.5.1 Introduction to QR Code Technology *158*
4.5.2 Implementing QR Code Readers in Android *159*
4.5.3 Use Cases, Benefits, and Best Practices *160*
4.5.4 Implementing a QR Code Reader in Android with ZXing Library *161*
4.5.5 Creating the Scanner Activity *162*
4.6 BIOMETRIC AUTHENTICATION 163
4.6.1 Introduction to Biometric Authentication *163*
4.6.2 Privacy Risks and Data Security *164*
4.6.3 Accuracy, Inclusivity, and System Limitations *164*
4.6.4 How Biometric Systems Work *165*
4.6.5 Public Security and Ethical Concerns *165*
4.6.6 Adoption in the Financial Sector *165*
4.6.7 Digital Identity and Centralized Systems *166*
4.6.8 Emerging Technologies and Innovations *166*
4.6.9 The Future of Biometric Authentication *166*
4.6.10 Android Fingerprint Technology, Mobile Security and User Experience *167*

4.6.11 Implementing Biometrics in Android with Java *170*
4.7 TEXT-TO-SPEECH 173
4.7.1 Introduction to Text-to-Speech in Android *173*
4.7.2 Implementing Text-to-Speech in Android Applications *173*
4.7.3 Use Cases, Challenges, and Best Practices *174*
4.8 BLUETOOTH BEACON 175
4.8.1 Definition and Technology *175*
4.8.2 Applications and Benefits *177*
4.8.3 Bluetooth Beacons Challenges *178*
4.8.4 Bluetooth Beacons and Future Potentials *178*
4.8.5 Implementing Bluetooth Beacons Detectors in Android with Java and AltBeacon Library *179*
4.9 FIREBASE 185
4.9.1 Firebase and Its Role in Android Development *185*
4.9.2 Firebase Realtime Database and Cloud Firestore in Android *185*
4.9.3 Firebase Authentication and User Management *186*
4.9.4 Firebase Cloud Messaging, Storage, and Analytics *187*
4.9.5 Best Practices and Use Cases for Firebase in Android *187*
4.9.6 Implementing Android Application with Firebase Realtime Database *188*
4.9.7 Security and Authentication in Firebase Realtime Database *199*
4.9.8 Firebase Realtime Database Pricing *201*
4.9.9 Advantages of Using Firebase Realtime Database *202*
4.9.10 Limitations of Firebase Realtime Database *204*
4.9.11 Firebase Realtime Database Conclusions *205*
5 RESULTS AND EVALUATION 206
5.1 INTRODUCTION 206
5.2 EVALUATION OBJECTIVES AND RESEARCH QUESTIONS 207
5.3 OVERALL EVALUATION METHODOLOGY 208
5.3.1 Methodological Separation *208*
5.3.2 Baseline Systems *209*
5.4 SYNTHETIC-DATA-BASED SYSTEM PERFORMANCE EVALUATION 210
5.4.1 Synthetic Interaction Streams *210*
5.4.2 CPU and Memory Consumption *211*
5.4.3 Network Traffic and Latency *213*
5.5 SYNTHETIC-DATA-BASED ARTIFICIAL INTELLIGENCE EVALUATION 214

5.5.1 Synthetic Dataset Characteristics *214*
5.5.2 User Classification Results *215*
5.5.3 Recommendation Quality Results *216*
5.6 Controlled Observational Usability Evaluation 217
5.6.1 Study Design *217*
5.6.2 Task Performance Results *217*
5.6.3 System Usability Scale (SUS) *219*
5.7 Comparative Evaluation Summary 220
5.8 Threats to Validity 221
5.9 Discussion 222
5.10 Conclusions of Evaluation 223
6 RELATED WORK 225
6.1 Augmented Reality and Guiding Systems 225
6.2 Artificial Intelligence and Personalization 226
6.3 Mobile Tourism and Cultural Heritage Applications 226
6.4 Positioning the Present Work 227
7 CONCLUSIONS 228
7.1 General Scientific Impact 228
7.2 Novelty and Evaluation Driven Contributions 229
7.3 Dissertation Findings 229
8 REFERENCES 231
9 APPENDICES 244
APPENDIX 1 CLASS DIAGRAM 245
APPENDIX 2 AI ALGORITHMS AND DATA 248

LIST OF TABLES

TABLE 4-1: FIREBASE REALTIME DATABASE PRICING 201

TABLE 5-1: CPU VS MEMORY CONSUMPTION 212

TABLE 5-2: MEASURING NETWORK TRAFFIC AND LATENCY 213

TABLE 5-3: GENERAL DATASET CHARACTERISTICS 215

TABLE 5-4: USER CLASSIFICATION EVALUATION 215

TABLE 5-5: RECOMMENDATION QUALITY EVALUATION 216

TABLE 5-6: TASK PERFORMANCE ANALYSIS 218

TABLE 5-7: MEASURING SYSTEM USABILITY SCALE 219

LIST OF FIGURES

FIGURE 2.1: PERSONAL TOUR GUIDE PLATFORM'S ARCHITECTURE 53

FIGURE 2.2: (A) LOGIN WINDOW AND (B) REGISTRATION WINDOW MOCKUPS 57

FIGURE 2.3: (A) MAIN WINDOW AND (B) GUIDE WINDOW MOCKUPS 58

FIGURE 2.4: FIREBASE JSON UML DIAGRAM 60

FIGURE 2.5: ONLINE SECURITY PACKAGE IMPLEMENTATION 62

FIGURE 2.6: PERSONAL TOUR GUIDE APPLICATION CLASS DIAGRAM 65

FIGURE 2.7: MAIN WINDOW SEQUENCE DIAGRAM 67

FIGURE 2.8: TOUR GUIDE WINDOW SEQUENCE DIAGRAM 68

FIGURE 3.1: VISUAL REPRESENTATION OF A PERCEPTRON AND ITS MATHEMATICAL FUNCTION 84

FIGURE 3.2: SIMPLE VERSUS DEEP NEURAL NETWORKING. 84

FIGURE 3.3: VISUALIZATION OF A SUPERVISED NEURAL NETWORK LEARNING EXAMPLE. 85

FIGURE 3.4: VISUALIZATION OF AN UNSUPERVISED NEURAL NETWORK LEARNING EXAMPLE. 86

FIGURE 3.5: REPRESENTATION OF AN UNDERFITTED VS OVERFITTED AND A GENERALIZED NEURAL NETWORK. 87

FIGURE 3.6: RECURRENT TRAINING ON NEURAL NETWORKS. 90

FIGURE 3.7: USER'S CLASSIFICATION ALGORITHM. 94

FIGURE 3.8: (A) VISUAL REPRESENTATION OF USERS' DATA FROM INPUT FILE; (B) THE TRAINING ERROR PROGRESS. 95

FIGURE 3.9: (A) THE PYTHON ALGORITHM TO CLASSIFY NEW USERS; (B) VISUAL REPRESENTATION OF THE NEW USERS INFO ENTRIES. 95

FIGURE 3.10: (A) CREATING TRAINING DATA AS INPUT FOR A MULTI LAYERED NEURAL NETWORK; (B) VISUAL REPRESENTATION OF THE GENERATED TRAINING DATA. 97

FIGURE 3.11: THE MULTI LAYER NEURAL NETWORK FOR USERS' CLASSIFICATION. 98

FIGURE 3.12: (A) VISUALIZING THE TRAINING ERROR PROGRESS OF THE MULTI LAYER NEURAL NETWORK; (B) DEMONSTRATION OF ACTUAL VERSUS NEURAL NETWORK PREDICTED RESULTS FOR THE SAME INPUT DATA. 99

FIGURE 3.13: PYTHON IMPLEMENTATION OF THE K NEAREST NEIGHBORS NEURAL NETWORK. 102

FIGURE 3.14: (A) VISUALIZATION OF THE KNN INPUT DATA; (B) VISUALIZATION OF THE KNN RESULTS FOR K=5 AND THE GIVEN TEST POINT X. 103

FIGURE 3.15: TRAINING NEURAL NETWORK FOR USERS' IDENTIFICATION THROUGH CLASS MAPPING. 104

FIGURE 3.16: (A) MAPPING OF THE TRAINING POINTS USING FOUR DIFFERENT SHAPES; (B) COLOR GRID TO VISUALIZE THE PRE-CALCULATED MODEL OUTPUT. 105

FIGURE 3.17: USER' IDENTIFICATION THROUGH NEURAL NETWORK CLASS MAPPING 106

FIGURE 3.18: (A) KNN INPUT TRAINING DATA OVER EXPECTED SPATIAL PARTITIONING; (B) EXTRACTION OF THE 12 CANDIDATES OF THE K NEAREST NEIGHBORS AT THE TEST DATA POINT. 107

FIGURE 3.19: MIN-MAX NORMALIZATION FUNCTION. 109

FIGURE 4.1: EXAMPLE OF RETROFIT USAGE IN ORDER TO ACHIEVE ANDROID APP - WEB SERVER RESTFUL API COMMUNICATION 117

FIGURE 4.2: ANDROID DEVICE TO MYSQL SERVER COMMUNICATION 127

FIGURE 4.3: APPLICATION'S LOGIN AND REGISTRATION WINDOW 129

FIGURE 4.4: MYSQL TABLE (USERS) 132

FIGURE 4.5: SQLITE TABLE (LOCAL_USERS) 132

FIGURE 4.6: WEB PLATFORM HIERARCHY 133

FIGURE 4.7: CONTENT OF THE REGISTER.PHP FILE 134

FIGURE 4.8: CONTENT OF THE LOGIN.PHP FILE 135

FIGURE 4.9: CONTENT OF THE APPCONFIG.CLASS FILE 136

FIGURE 4.10: GRADLE DEPENDENCIES FOR HTTP REQUESTS 137

FIGURE 4.11: DEPENDENCIES FOR OSMDROID AND OSMBONUSPACK 152

FIGURE 4.12: ANDROID MANIFEST PERMISSIONS FOR OSMDROID AND OSMBONUSPACK 152

FIGURE 4.13: MAPVIEW DESIGN IMPLEMENTATION 153
FIGURE 4.14: MAPVIEW CONFIGURATION EXAMPLE 154
FIGURE 4.15: ADDING A SIMPLE MARKER WITH A CUSTOM INFOWINDOW 155
FIGURE 4.16: DRAWING A POLYLINE (PATH) 155
FIGURE 4.17: ROUTING EXAMPLE WITH OSRMROADMANAGER 156
FIGURE 4.18: REQUESTING ACCESS TO USER'S LOCATION 156
FIGURE 4.19: NAVIGATING FROM USER'S POSITION TO DESTINATION 157
FIGURE 4.20: ACTIVATION OF QR CODE READER 162
FIGURE 4.21: IMPLEMENTATION OF QR CODE READER 163
FIGURE 4.22: BIOMETRICS DEPENDENCY 170
FIGURE 4.23: DECLARING THE TYPES OF BIOMETRIC AUTHENTICATION 171
FIGURE 4.24: SETUP OF BIOMETRIC PROMPT IN ANDROID 172
FIGURE 4.25: TTS ENGINE SETUP EXAMPLE 174
FIGURE 4.26: ALTBEACON DEPENDENCY IMPLEMENTATION 180
FIGURE 4.27: BLUETOOTH BEACON MANIFEST IMPLEMENTATION 180
FIGURE 4.28: BLUETOOTH BEACON MANAGER CREATION 182
FIGURE 4.29: BLUETOOTH BEACON DETECTION 183
FIGURE 4.30: SETTING UP FIREBASE IN AN ANDROID PROJECT 189
FIGURE 4.31: GRADLE DEPENDENCY FOR GOOGLE SERVICES 190
FIGURE 4.32: IMPLEMENTING FIREBASE DEPENDENCIES 190
FIGURE 4.33: FIREBASE CONNECTION TESTING 191
FIGURE 4.34: FIREBASE JSON EXAMPLE 192
FIGURE 4.35: FIREBASE DATAREFERENCE EXAMPLE 192
FIGURE 4.36: USER CLASS EXAMPLE 193
FIGURE 4.37: FIREBASE SETVALUE EXAMPLE 193
FIGURE 4.38: USERS JSON FILE 194
FIGURE 4.39: CONTENT OF THE TOURDATA CLASS 195

FIGURE 4.40: FIREBASE DATA CHANGE LISTENER 196
FIGURE 4.41: RETRIEVING TOUR DATA FROM FIREBASE REALTIME DATABASE 197
FIGURE 4.42: FIREBASE UPDATE EXAMPLE 198
FIGURE 4.43: FIREBASE OFFLINE CAPABILITY 198
FIGURE 4.44: EXAMPLE RULE TO ALLOW ONLY AUTHENTICATED USERS TO READ/WRITE 199
FIGURE 4.45: FIREBASE READ & WRITE DATA WITHOUT AUTHENTICATION 199
FIGURE 4.46: FIREBASE VALIDATION RULES 200
FIGURE 9.1: CLASS DIAGRAM SECTION (A) 245
FIGURE 9.2: CLASS DIAGRAM SECTION (B) 245
FIGURE 9.3: CLASS DIAGRAM SECTION (C) 246
FIGURE 9.4: CLASS DIAGRAM SECTION (D) 247
FIGURE 9.5: APPENDIX A.1, INPUT TEST DATA, FOUR COLUMNS 248
FIGURE 9.6: APPENDIX B.1, INPUT TEST DATA, THREE COLUMNS 248
FIGURE 9.7: APPENDIX C.1, TRAINING USERS' DATA 249
FIGURE 9.8: APPENDIX C.2, ARGUMENTS PARSER 249
FIGURE 9.9: APPENDIX C.3, EUCLIDEAN SCORE ALGORITHM 250
FIGURE 9.10: APPENDIX C.4, PEARSON SCORE ALGORITHM 250
FIGURE 9.11: APPENDIX C.5, THE MAIN FUNCTION 251
FIGURE 9.12: APPENDIX C.6, THE ARGUMENT PARSER 251
FIGURE 9.13: APPENDIX C.7, FIND SIMILAR USERS FUNCTION 251
FIGURE 9.14: APPENDIX C.8, SIMILAR USERS MAIN FUNCTION 252
FIGURE 9.15: APPENDIX C.9, THE MOVIE RECOMMENDATION PARSER 252
FIGURE 9.16: APPENDIX C.10, THE GET MOVIE RECOMMENDATION FUNCTION 252
FIGURE 9.17: APPENDIX C.11, THE MOVIE RECOMMENDATION MAIN FUNCTION 253

List of Abbreviations and Acronyms

OS	Operating System
ART	Android Runtime
HAL	Hardware Abstraction Layer
OEM	Original Equipment Manufacturers
AOSP	Android Open Source Project
AR	Augmented Reality
IMU	Inertial Measurement Unit
GPS	Global Positioning System
SLAM	Simultaneous Localization and Mapping
AI	Artificial Intelligence
ANN	Artificial Neural Network
KNN	K Nearest Neighbors
LLM	Large Language Model
IP	Internet Protocol address
VPN	Virtual Private Network
FFIEC	Federal Financial Institutions Examination Council
MFA	Multi Factor Authentication
OTP	One Time Passwords
JWT	JSON Web Tokens
SPA	Single Page Applications
UUID	Universally Unique Identifier
IDFA	Apple's Identifier for Advertisers
GAID	Google Advertising ID
GDPR	General Data Protection Regulation
CCPA	California Consumer Privacy Act

2FA Two Factor Authentication 2FA

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

ROI Return on Investment

FNN Feed-forward Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

GAN Generative Adversarial Network

NLP Natural Language Processing

XAI Explainable Artificial Intelligence

ReLU Rectified Linear Unit

DPP Deep Neural Networks

CPU Central Processing Unit

GPU Graphics Processing Unit

TPU Tensor Processing Unit

SGD Stochastic Gradient Descent

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

BERT Bidirectional Encoder Representations from Transformers

GPT Generative Pre-Trained

ViT Vision Transformer

GDP Gross Domestic Product

VR Virtual Reality

HMD Head Mounted Displays

UX User Experience

IV Intravenous

OHA Open Handset Alliance

MVC Model View Controller

UCD User-Centered Design

XSS Cross-Site Scripting

CSRF Cross-Site Request Forgery

HTTPS Hypertext Transfer Protocol Secure

SSL Secure Sockets Layer

UML Unified Modeling Language

JSON JavaScript Object Notation

ISP Internet Service Provider

RESTful Representational State Transfer

API Application Programming Interface

TLS Transport Layer Security

ANR Application Not Responding

RDBMS Relational Database Management System

CRUD Create, Read, Update, and Delete

GIS Geographic Information Systems

WAAS Wide Area Augmentation System

RTK Real-Time Kinematic

GNSS Global Navigation Satellite System

OSM OpenStreetMap

ODbL Open Database License

POI Point of Interest

OSRM Open Source Routing Machine

QR Quick Response

URL Uniform Resource Locator

TEE Trusted Execution Environment

SAR Spoof Acceptance Rate

FAR	False Acceptance Rate
TTS	Text-to-Speech
BLE	Bluetooth Low Energy
RSSI	Received Signal Strength Indicator
IoT	Internet of Things
UWB	Ultra-Wideband
RTT	Round Trip Time
BaaS	Backend-as-a-Service
FCM	Firebase Cloud Messaging
HIPAA	Health Insurance Portability and Accountability Act
UID	Unique Identifier

LIST OF APPENDICES

APPENDIX 1 CLASS DIAGRAM 245

APPENDIX 2 AI ALGORITHMS AND DATA 248

1 INTRODUCTION

1.1 The Tour Guide Operators Field and its Impact on the Economy

1.1.1 Overview of the Tour Guide Operators Industry

Tour guide operators form an essential segment of the tourism industry [1-3]. They plan, organize, and deliver experiences for travelers, ranging from historical city walks to wildlife safaris and cultural immersion programs. Tourism is one of the largest and fastest-growing industries in the world [4-6], and within this expansive sector, tour guide operators play a vital role. Tour guide operators are the facilitators of experience [7], culture [8,9], and learning [10,11]. They serve as the interface between the traveler and the destination, helping bridge language barriers, cultural gaps, and logistical challenges. The profession of tour guiding is as old as travel itself, evolving alongside technological advancement, changing traveler expectations, and global mobility trends.

Their role extends beyond merely showing places as they interpret culture, manage logistics, ensure safety, and enhance the visitor experience. The field of tour guiding is highly varied [12,14]. It includes freelance guides, small local operators, and large multinational companies. These operators work with tourists directly or through partnerships with travel agencies, hotels, and transport providers. Tour guide operators enhance the overall travel experience by providing curated, informed, and often entertaining tours that bring history, art, culture, and nature to life. However, their contribution extends beyond the tourist experience. They are major contributors to the economic ecosystems of destinations [15,16]. Through direct employment, support of

ancillary industries, and enhancement of local economies, tour guide operators serve as economic multipliers.

The economic impact of tour guide operators is significant but often underappreciated [16-18]. They generate direct income through their services, create employment opportunities, stimulate demand for local goods and services, and help distribute tourism revenue more evenly, especially in rural or developing regions. In this thesis we will try to explore the tour guide operators' field in detail, describing its structure, functions, and the wide-ranging economic impacts it has on local, regional, and national economies.

1.1.2 The Structure of the Tour Guide Operator Industry

Typically, the industry has a large number of small and micro-enterprises, especially in developing regions. In advanced economies, some large tour operators consolidate services to offer scaled packages, but local guides still play a vital role in delivering the personal, authentic aspect of the experience.

The tour guide operator sector is diverse and it mainly includes the following:

- Local Guides, (individuals who specialize in specific cities, heritage sites, or natural areas);
- Inbound Tour Operators, (companies organizing experiences for foreign visitors in their country);
- Outbound Tour Operators, (those creating packaged tours for domestic travelers to go abroad);
- Specialized Operators, (adventure tours, eco-tours, culinary tours, heritage tours, etc.).

The tour guide operator industry is highly diverse, encompassing a range of business types and service models. At its core, there are local tour guides who often work independently or with small companies to provide specialized, place-based experiences such as city walking tours, museum visits, or food tours. Then there are inbound tour operators who handle travel logistics for foreign visitors, organizing accommodation, transport, and curated excursions. Outbound operators, on the other hand, plan trips for local residents traveling abroad, coordinating with foreign partners to ensure a seamless experience. Domestic tour operators focus on local tourism within a country, catering to residents exploring their own regions, a segment that gained significant importance

during travel restrictions imposed by global crises. In addition to these broad categories, there are niche and specialty operators who design unique, highly targeted experiences. These can include adventure tourism companies offering trekking or rafting expeditions, eco-tourism operators emphasizing conservation and nature interpretation, or cultural and spiritual tourism providers catering to travellers seeking meaningful, immersive encounters. Such specialization allows businesses to differentiate themselves in a competitive market while appealing to the growing demand for authentic, responsible, and personalized travel experiences.

The employment and business models within this industry are equally varied. Many guides work as freelancers or subcontractors, especially in regions with large informal economies. Small, often family-run enterprises may operate within specific communities or regions, directly supporting local livelihoods. Meanwhile, large tour operators can be international corporations with vertically integrated services that include their own hotels, transport fleets, and local guides. This structural diversity enables wide economic participation, offering low-barrier entrepreneurial opportunities to local residents, including women and marginalized groups. Moreover, the fragmented nature of the industry allows for a high level of adaptability and inclusivity. In developing countries and agricultural regions, guiding often serves as an accessible form of formal employment that does not require large capital investments. It also creates opportunities for cultural preservation and storytelling by empowering local communities to share their heritage on their own terms. Altogether, this varied industry structure enables tour guide operators to serve as important drivers of inclusive economic development while responding to a wide range of traveller needs and market demands.

1.1.3 Roles and Responsibilities of Tour Guide Operators

Tour guide operators perform a range of roles [19,20], including:

- Planning journey design, (creating structured, appealing, and safe tours);
- Interpretation and storytelling, (explaining historical, cultural, environmental, and social contexts);
- Logistics and coordination, (managing bookings, transport, meals, and timing);
- Risk management, (ensuring safety, adhering to local regulations, and handling emergencies);

- Customer service, (addressing tourists' needs, providing support, and managing expectations).

Tour guide operators have a central role in designing and delivering travel experiences that are both enjoyable and meaningful. One of their core responsibilities is planning and developing itineraries that balance traveller interests with practical considerations such as timing, safety, and accessibility. This planning involves thorough research into attractions, understanding local customs and seasonal factors, coordinating bookings for transport and accommodation, and anticipating challenges that might arise along the way. A well designed journey not only maximizes tourist satisfaction but also supports local businesses and communities by distributing visitor spending across multiple services and sites. Another critical responsibility is interpretation and storytelling. Tour guides don't simply point out landmarks but they bring destinations to life through engaging narratives, historical context, and cultural insights. They often receive training in subjects like history, art, ecology, and anthropology to ensure they can answer questions and deepen visitors' understanding. This interpretive role is essential in creating authentic connections between travellers and the places they visit. By educating tourists about local heritage, guides also promote cultural appreciation and responsible behavior, helping reduce misunderstandings and disrespectful interactions with host communities.

In addition to storytelling, tour guides manage complex logistics [21], and safety [22]. They coordinate transportation, meals, and schedules to keep tours running smoothly and on time. This requires strong organizational skills and the ability to adapt quickly when plans change due to weather, local events, or unforeseen delays. Safety is a major concern also. This means that guides must know emergency procedures, basic first aid, and legal responsibilities, especially in adventure tourism or remote destinations. Their attentiveness ensures that travellers can relax and enjoy the experience, knowing they're in capable hands.

Finally, tour guides act as cultural mediators and customer service professionals. They help visitors navigate through language barriers, understand local customs, and interact respectfully with communities. This mediation role is vital for minimizing cultural friction and ensuring positive encounters for both tourists and residents. At the same time, guides must manage group dynamics, resolve conflicts or complaints, and

maintain a welcoming, enthusiastic attitude throughout the journey. Their ability to deliver high quality customer service often determines whether tourists leave with positive memories, recommend the destination to others, or return for future visits, which feature makes tour guides critical ambassadors for their regions and countries.

1.1.4 Direct Economic Impacts

The most obvious economic impact of tour guide operators is direct income generation [23]. Tourists pay fees for guided tours, and these revenues flow to local guides, companies, and support staff. For many destinations, especially in developing countries, guiding services provide one of the most accessible ways for local people to participate in the tourism economy.

Some examples are:

- Employment creation, (directly like guides or drivers and indirectly like administrative staff, marketers, logistics managers);
- Local business support, (guides often steer tourists toward local restaurants, shops, and artisans);
- Tax Revenue, (formal operators pay taxes, contributing to local government budgets);
- Foreign Exchange, (inbound tourism, foreign visitors bring in hard currency).

Tour guide operators generate direct economic benefits primarily through the in-come they earn from providing services to tourists. These earnings support not only the guides themselves but also a wide range of related roles, including office staff, coordinators, drivers, and translators. The industry creates formal and informal employment opportunities, particularly in regions where other job sectors may be limited. In developing countries, guiding often serves as a key entry point into the formal economy, offering livelihoods to individuals with knowledge of local culture and language skills. Additionally, tour guiding is a relatively accessible form of self-employment or entrepreneurship, enabling people and especially women and marginalized groups to start their own businesses with minimal capital. Beyond employment, tour guide operators contribute to national economies through the taxes and fees they generate. Licensed operators pay income taxes, business taxes, and often tourism specific levies that help and fund public services and tourism infrastructure. Inbound operators also

bring foreign currency into the economy, which supports balance of trade and contributes to overall economic stability. Furthermore, the training and skills development required in tour guiding sector such as language proficiency, customer service, and safety awareness enhance the overall human capital of a country, preparing individuals for broader roles in the tourism and hospitality sectors.

1.1.5 Indirect and Induced Economic Impacts

Beyond direct revenues, tour guide operators have broad indirect and induced impacts [24]. Studies often found tourism's multiplier effect is high. Because every dollar spent on a tour can generate several dollars in the wider economy. This ripple effect is crucial for agricultural field or marginalized communities with limited economic opportunities. Tour guiding also encourages infrastructure development. Roads, communication networks, signage, and public facilities often improve in response to tourism demand.

Some of the major indirect impacts are the following:

- Supply chains, (operators purchase transport services, fuel, meals, entrance tickets, equipment);
- Multiplier effects, (income earned by guides and staff is spent in the local economy, supporting shops, markets, and services);
- Seasonal stability, (in regions with strong tourism seasons, guides may sustain economic activity when other sectors slow down).

Moreover, tour guide operators have significant indirect economic impacts by stimulating demand for a wide range of supporting services and industries. When tourists book a guided experience, they also spend money on transportation (such as buses, taxis, or boats), accommodations (hotels, guesthouses, homestays), food and beverages (restaurants, street vendors, caterers), and entry to attractions like museums, parks, and cultural sites. Guides often introduce tourists to local craftspeople and markets, boosting sales of souvenirs and artisan products. This interconnected supply chain ensures that tourism revenue circulates throughout the local economy, supporting jobs and businesses far beyond the guiding service itself. The induced economic impacts arise from the spending of incomes earned through tourism-related activities. For example, the wages that guides, drivers, hotel staff, and restaurant workers earn are spent on everyday needs such as food, housing, education, and healthcare. This creates

additional demand in the local economy, generating further rounds of economic activity, a process often called the “multiplier effect”. Over time, this sustained spending can encourage broader development, such as improved infrastructure, better public services, and increased investment in new tourism products or facilities. Thus, the economic benefits of tour guiding ripple outward, amplifying its role in local and national development.

1.1.6 Cultural and Social Economic Benefits

Tour guide activities provide diversified income streams, helping reduce poverty and reliance on single sectors like agriculture or extractive industries [25]. Moreover, responsible operators often promote sustainable tourism principles, minimizing environmental impacts and educating tourists about local challenges and solutions.

Tour guide operators also drive important cultural economy impacts such us:

- Cultural preservation, (interpreting and promoting cultural heritage, guides support awareness and conservation);
- Revitalization of traditions, (demand for authentic experiences encourages traditional crafts, performances, and storytelling);
- Community engagement, (local communities can benefit directly by offering cultural programs, meals, or homestays).

Tour guide operators play a crucial role in preserving and promoting local culture, creating important cultural and social economic benefits. By interpreting historical sites, traditions, and stories for visitors, guides encourage appreciation and respect for cultural heritage, often helping fund its preservation through entrance fees and donations. They also act as vital links between tourists and local artisans, musicians, and performers, driving demand for traditional crafts and cultural experiences. This not only sustains livelihoods for local artists but also helps maintain and revitalize cultural practices that might otherwise be lost in the face of modernization or globalization. Beyond cultural preservation, tour guide operators promote social inclusion and community development. Community based tourism initiatives often rely on local guides to ensure tourism revenue reaches residents directly through homestays, local tours, and cultural workshops. Guiding provides accessible employment opportunities, especially for women and young people, empowering them with income, skills, and leadership

experience. This inclusive approach helps reduce economic inequalities within regions and strengthens community cohesion. By encouraging respectful, meaningful interaction between tourists and locals, guides also help build mutual understanding and social harmony, making tourism a tool not only for economic growth but also for cultural exchange and community well-being.

1.1.7 Environmental and Regional or National Economic Impacts

Well-managed tourism can help fund and justify conservation areas. This creates jobs while maintaining the ecosystem services, like clean water, biodiversity, and climate regulation that underpin long-term economic health [26]. Poorly managed tour guiding, however, can have the opposite effect, causing overcrowding, habitat destruction, and cultural commodification. Hence, professional, trained, and responsible operators are critical for maximizing positive impacts.

Sustainable tour guide operators have an important role in mitigating negative environmental impacts and generating environmental economic value based on:

- Education, (guides teach tourists about conservation, reducing litter, respecting wildlife);
- Revenue for conservation, (tours often include park fees or donations to conservation projects);
- Monitoring and stewardship, (guides act as eyes on the ground to report environmental damage or illegal activities).

On a regional scale, tour guide operators help distribute economic benefits beyond major urban centers such as:

- Decentralization of tourism, (tours can bring visitors to rural, remote, or less-developed areas);
- Balanced development, (this reduces pressure on overcrowded destinations and spreads economic opportunities);
- Resilience building, (diverse tourism offerings help regions withstand shocks like natural disasters or market shifts).

At the national level, tour guide operators support:

- GDP growth, (tourism is often a major GDP contributor since guided experiences are a core driver of value);
- Employment strategy, (labor-intensive, low-barrier-to-entry jobs help address un-employment);
- Foreign investment attraction, (a well-developed tourism sector signals stability and opportunity);
- Brand image, (high-quality tours shape visitors' perceptions and encourage repeat visits).

Tour guide operators play a vital role in regional development by helping distribute tourism activity beyond major cities and popular hubs. By designing and leading tours in rural, remote, or underserved areas, guides encourage travellers to explore less visited destinations. This helps spread tourism revenue more evenly, reducing pressure on overcrowded urban centers while bringing new sources of income to regions that might otherwise have limited economic opportunities. The resulting investment in local businesses, infrastructure, and community services can reduce regional inequalities and improve quality of life for residents.

At the national level, the tourism sector, including guiding services, can make a substantial contribution to Gross Domestic Product (GDP). In some countries, particularly small island or developing nations, tourism can account for a significant share of GDP, sometimes exceeding 50%. Tour guide operators are central to this contribution because they enhance the visitor experience, encourage longer stays, and boost overall spending. Their services make destinations more appealing and accessible to international tourists, increasing foreign exchange earnings that strengthen national economies and help balance trade dept.

Tour guiding also serves as an important employment strategy, especially in labor-rich but capital-poor countries. The industry offers accessible, low barrier job opportunities for people with local knowledge and language skills, supporting formal employment where other options may be difficult to be achieved. Governments often recognize this potential by including tourism development in national employment and poverty reduction plans. Beyond direct employment for guides themselves, the sector supports many related jobs in hospitality, transport, retail, and cultural services, making it a cornerstone of inclusive economic growth strategies.

Finally, tour guides act as ambassadors for their countries, shaping how visitors perceive the nation's culture, people, and values. Positive experiences delivered by skilled guides can improve a country's reputation abroad, boosting communication and encouraging diplomatic goodwill. Satisfied visitors are more likely to recommend destinations to others or return in the future, creating a cycle of sustained demand. In addition, quality tourism experiences attract investors interested in developing hotels, transport networks, and other infrastructure, further amplifying the sector's long-term economic impact at a national scale.

1.1.8 Challenges Facing Tour Guide Operators

Policy responses can include licensing regimes, training programs, infrastructure investments, marketing support, and crisis management planning [19]. Collaboration among government, private sector, and communities is essential to address these challenges and maximize the economic benefits.

Despite these benefits, the sector faces challenges:

- Seasonality, (many destinations have peak and low seasons, making income unstable);
- Informality, (in many places unlicensed or informal guiding undermines quality and tax revenue);
- Training gaps, (lack of professional development can hurt safety, quality, and sustainability);
- Over-tourism, (popular sites risk being overwhelmed, damaging resources and resident goodwill);
- External or unpredicted events, (pandemics, political instability, or natural disasters can devastate demand).

One major challenge facing tour guide operators is seasonality and demand fluctuation. Tourism demand is often highly seasonal, with peaks during holidays or favorable weather and dramatic slowdowns during off-seasons. This can make income for guides and operators unreliable and limit their ability to invest in training or business growth. In many regions, guides must take on other jobs during the low season or rely on savings, making it difficult to maintain consistent staff and service quality year round. Seasonality also complicates planning for governments and destination managers, who

must balance the need for tourism development with the risk of overdependence on a sector that is not steady throughout the year.

Another significant issue is economic vulnerability to external shocks. Political instability, pandemics, natural disasters, or global economic downturns can all disrupt tourism demand, leaving guides and operators suddenly without income. The COVID-19 pandemic, for example, devastated the tourism industry worldwide, pushing many guides into unemployment and causing small operators to close permanently. Because guiding often involves informal or freelance work, many guides lack social protections like health insurance or unemployment benefits, making them particularly vulnerable in times of crisis. These shocks highlight the need for better social safety nets and business resilience planning within the sector.

Regulatory challenges and barriers to formalization also pose problems for tour guide operators. In many countries, guiding is only partly regulated, with unclear or inconsistent licensing requirements, high fees, or complex bureaucracy discouraging operators from formal registration. This can lead to a large informal economy where guides operate without legal protections or access to training, quality standards, or marketing support. While formalization could improve service quality and worker protections, governments often struggle to enforce regulations or provide accessible paths to compliance, leaving guides in a legal grey area that undermines both their livelihoods and the reputation of destinations.

Competition and market pressures further complicate the work of tour guide operators. The rise of online booking platforms and peer-to-peer tourism services has made it easier for unlicensed or inexperienced guides to offer tours, sometimes undercutting prices and eroding quality. At the same time, travellers are increasingly demanding specialized, authentic, and sustainable experiences, requiring operators to invest in training, marketing, and partnerships to stay competitive. Balancing affordability with quality, while adapting to changing traveller expectations and technological shifts, can be especially challenging for small or community-based operators with limited resources.

Finally, environmental and social sustainability concerns present both a challenge and a responsibility for tour guide operators. Unmanaged tourism can contribute to environmental degradation, overcrowding at sensitive sites, and cultural commodification that harms local communities. Guides are often on the front lines of

managing visitor behavior, but they may lack training, resources, or authority to enforce sustainable practices. At the same time, operators must balance sustainability with business pressures to maximize tourist numbers and income. Navigating these tensions requires collaboration with governments, local communities, and other stakeholders to ensure tourism remains a force for long-term positive development rather than a source of harm.

1.1.9 The Future of the Tour Guide Operator Field

The future of the tour guide operator field is expected to be shaped by increasing demand for personalized, authentic, and meaningful travel experiences [27]. As tourists be-come more experienced and discerning, they seek deeper engagement with local cultures, traditions, and environments rather than generic sightseeing. This trend is pushing tour operators to specialize in niche markets like culinary tourism, adventure travel, heritage trails, and eco-tourism. Guides will need to develop richer storytelling skills, in-depth local knowledge, and the ability to customize experiences to individual interests, making their role even more central to delivering high-quality tourism products.

Technological innovation will also transform the field in multiple ways. Online booking platforms, review sites, and mobile apps have already revolutionized how travellers choose and book tours, increasing competition but also expanding market reach for local operators. Virtual and augmented reality tools may enhance pre-trip planning or even enable remote tours. At the same time, operators will increasingly use digital marketing and social media to connect with customers and build their brand. Tour guides themselves may use technology on tours to provide richer content, translations, or interactive experiences. Embracing these tools will be essential for operators who want to stay relevant and competitive in a rapidly changing industry.

Sustainability and responsible tourism will become even more important in shaping the future of guiding. As awareness grows about the negative impacts of over-tourism, environmental damage, and cultural exploitation, travellers are seeking operators who prioritize ethical practices. Future tour guide operators will need to adopt and promote sustainable tourism principles such as minimizing waste, respecting wildlife and cultural sites, and supporting local economies through fair partnerships. They will increasingly act as educators and advocates for responsible travel, encouraging tourists

to behave in ways that preserve destinations for future generations. Certification schemes and sustainability standards may become more widespread, creating both opportunities and obligations for operators.

The industry will also need to adapt to new forms of risk and uncertainty, such as pandemics, geopolitical tensions, or climate change. Resilient business models that can handle weather disruptions will be critical, potentially including diversification of services (for example, offering domestic or virtual tours during international travel restrictions). Operators may also need to rethink group sizes, health protocols, and insurance requirements to reassure travellers about safety. Governments and industry bodies will likely invest more in crisis preparedness and support for tourism workers to avoid the devastation seen during the COVID-19 pandemic.

Finally, the professionalization of guiding is likely to accelerate in the future. As tourism becomes more competitive and destinations focus on quality over quantity, there will be growing demand for trained, certified, and knowledgeable guides who can deliver exceptional experiences. This may involve expanded training programs in history, culture, languages, safety, customer service, and sustainability. Governments and industry associations may strengthen regulations and licensing to ensure consistent standards and consumer trust. For guides themselves, this evolution offers the chance to build respected, stable careers that contribute meaningfully to local economies and cultural preservation while offering travellers richer, more rewarding experiences.

1.2 Augmented Reality Guiding Systems

1.2.1 Introducing Augmented Reality Guiding Systems

A guiding system refers to any system designed to help users navigate, learn about, or interact with physical spaces or objects [28,29]. Classic guiding systems include maps, audio tour guides, signage, and even human guides. In modern contexts, guiding systems can be interactive apps, GPS navigation tools, or museum audio devices. Guiding systems aim to improve user orientation, provide relevant information at the right time and place, reduce confusion, and enhance the overall experience of moving through or learning about space, events, history, heritage, culture, etc.

Augmented Reality (AR) is a technology that superimposes computer-generated content onto a user's perception of the real world. Unlike Virtual Reality (VR), which immerses the user in a completely synthetic environment, AR blends digital elements with the real

environment in real time. Examples of AR include heads-up displays in cars that overlay navigation instructions onto the windshield, smartphone apps that show information about historical sites when viewed through the camera, and smart glasses that project work instructions onto physical machinery. AR relies on sensors, cameras, displays, and software to detect and interpret the user's environment and position, enabling accurate placement of digital objects. This real-time integration creates an "augmented" experience in which users can interact with both real and virtual elements seamlessly.

Traditional guiding systems often rely on static media (maps, signs) or require users to interpret abstract information and relate it to their surroundings. This can be challenging in complex environments such as large airports, industrial plants, museums with many exhibits, or outdoor cultural heritage sites.

AR-guiding systems solve this problem by overlaying context-sensitive information directly onto the user's view of the world by using:

- Arrows on the ground guiding a user to a gate at an airport;
- Labels appearing over museum artifacts with historical details;
- Maintenance instructions projected onto equipment.

The motivation for combining AR with guiding systems includes:

- Improved way finding (users see real-world paths augmented with digital arrows or markers, reducing errors and confusion);
- Contextual information (information appears exactly where and when it is needed, enhancing learning and engagement);
- Hands-free operation (with head-mounted displays or audio AR, users can receive guidance while keeping their hands free for other tasks);
- Personalization (AR systems can adapt content to the user’s preferences, language, or level of expertise);
- Engagement: Immersive experiences hold attention better than traditional signage or brochures.

1.2.2 History and Evolution of Augmented Reality Guiding Systems

The concept of Augmented Reality has its roots in early experiments with head-mounted displays (HMDs) and human-computer interaction in the 1960s [30,31]. Ivan Sutherland’s 1968 "Sword of Damocles" is often credited as the first true AR system: a head-tracked see-through display that overlaid wireframe graphics onto the real world. Although primitive, it established key principles of spatial registration and user viewpoint tracking. Throughout the 1970s–1990s, AR remained mostly in research labs. Milgram and Kishino (1994) formally described the "Reality-Virtuality Continuum" placing AR between real environments and fully virtual environments. The concept of combining real and virtual elements in a mixed reality space gained traction in academia and defense research. In the late 1990s and early 2000s, advances in computer vision and portable computing enabled early mobile AR prototypes. Examples include marker-based AR systems that used printed patterns to register virtual objects in camera feeds, paving the way for more sophisticated tracking approaches.

Meanwhile, guiding systems have a much older history. Human tour guides, maps, and signage are ancient technologies dating back thousands of years. The 20th century saw mass production of printed maps, standardized road signage systems, and audio guides in museums. Digital transformation began in earnest with the advent of GPS in the 1990s, enabling in-car navigation systems and handheld GPS devices for hiking. These systems greatly improved outdoor navigation but were limited indoors. By the early 2000s, smartphone apps emerged as a new generation of guiding systems. Location-based services used GPS, Wi-Fi, and later Bluetooth beacons for indoor positioning. Audio tours in museums evolved into multimedia smartphone guides, offering images, videos, and interactive maps.

The convergence of AR and guiding systems was driven by the need for more intuitive and context-sensitive navigation aids. Traditional maps and signage required users to interpret abstract information and translate it to their real surroundings, which is a cognitive burden that AR could alleviate. Early examples included research prototypes that overlaid arrows onto video feeds on PDAs or smartphones.

As cameras, processors, and displays improved, consumer-grade AR apps emerged, such as:

- Google’s AR walking directions in Maps;
- Museum apps overlaying labels or reconstructions onto exhibits;

- Tourism apps that point out landmarks when viewed through a phone camera.

Head-mounted displays like Google Glass (2013) and Microsoft HoloLens (2016) promised hands-free AR guidance for industrial maintenance, training, and professional use cases. These systems provided real-time instructions overlaid on the user's field of view. More recently, ARKit (Apple) and ARCore (Google) democratized AR development on smartphones, leading to a rapid increase in numbers of AR-guiding apps for navigation, tourism, education, and retail. The evolution of AR-guiding systems reflects broader trends in “found everywhere” computing, making information accessible anywhere, anytime, in the most natural and context-aware form possible.

1.2.3 Core Technologies of AR Guiding Systems

AR-guiding systems rely on a complex interplay of hardware, software, and algorithmic components [32,33]. The effectiveness of these systems depends not only on the quality of AR content but also on how well the underlying technologies work together to provide a seamless and context-aware user experience such as:

- Hardware components;
- Software platforms;
- Tracking and registration;
- Rendering;
- User interface and interaction.

Hardware components make AR guiding systems possible and fundamentally shape user experience. The discussion of display technologies, from handheld smartphones to head mounted displays, illustrates the trade-offs between accessibility and immersion. While smartphones are ubiquitous and affordable, they require users to hold them up, which can interrupt tasks or reduce safety in certain contexts. In contrast, head-mounted displays enable hands-free operation ideal for complex environments like factories or hospitals, but they introduce challenges around cost, comfort, and social acceptance. This contrast highlights how hardware decisions are not merely technical but directly influence adoption and usability.

Hardware components also focus on sensors and tracking components, which are vital for maintaining accurate alignment between digital overlays and the real environment. The inclusion of cameras, inertial measurement units (IMUs), GPS, and depth sensors reflects the complexity of achieving reliable, low-latency tracking across diverse settings. The analysis points out that each sensor type has strengths and limitations. For example, GPS works well outdoors but fails indoors, while vision-based tracking can be disrupted by poor lighting or repetitive surfaces. This complexity requires careful integration of multiple sensing types to ensure consistent performance. It demonstrates that robust hardware design is critical for meeting user expectations of seamless, intuitive guidance.

Processing units, connectivity, and power considerations, are often overlooked yet crucial to AR system viability. Processing power determines whether devices can handle real-time rendering, AI-based guidance, or computer vision tasks locally. While connectivity including Wi-Fi and emerging 5G networks, enables offloading to the cloud and collaborative AR experiences. However, all of this imposes significant power demands, making battery life a persistent constraint for mobile and wearable devices. This wisely underscores that successful AR guiding systems must balance performance with energy efficiency, emphasizing that the entire hardware stack must be optimized not just for technical capability but for sustained, practical use in real-world conditions.

On the other and, software platforms play crucial role in enabling AR guiding systems to deliver reliable, engaging, and context-aware experiences. By covering AR SDKs, operating system supports, and development frameworks, the software provides the backbone for features like spatial mapping, object recognition, and real-time rendering. This discussion makes clear that while hardware delivers the raw capability, software defines how effectively that capability is harnessed, determining ease of development, cross-platform compatibility, and the richness of user interactions. Importantly, it also points to the challenge of fragmentation, where developers must navigate different tools and standards across devices, underlining the need for robust, flexible, and well-supported software ecosystems to make AR guiding systems truly scalable and sustainable.

Tracking and registration are the technical heart of any effective AR guiding system, as they ensure that digital overlays align accurately with the user’s real-world view in real time. By explaining techniques like marker-based, marker less, and sensor fusion approaches, technology reveals both the sophistication and challenges of maintaining

precise, low-latency tracking in varied environments. This focus on the fact that without robust tracking and registration, even the most compelling AR content becomes confusing or unusable, breaking user trust and reducing effectiveness. It also points to an ongoing tension in AR design. The balancing accuracy and responsiveness with computational efficiency and adaptability to real-world conditions, making tracking and registration a key area for innovation and continuous improvement.

Rendering focuses on the process that translates raw data into coherent, immersive visuals in AR guiding systems. Rendering is not just about visual constancy but also about real-time performance, contextual relevance, and spatial coherence. It highlights that successful rendering must ensure digital elements appear anchored in the physical world, with proper lighting, scale, and occlusion to maintain realism and user trust. Moreover, it underscores the computational demands of high-quality rendering, particularly on mobile or wearable devices with limited resources. This analysis reveals that rendering is a critical link between technical capability and user perception, and any lag, jitter, or visual inconsistency can significantly degrade the AR guiding experience.

User interface (UI) and interaction design are central in making AR guiding systems intuitive, accessible, and effective. By exploring elements like gesture controls, voice commands, and gaze-based selection, the section highlights the need for natural, context-aware inputs that reduce cognitive load and keep users focused on their environment. There are challenges in balancing rich functionality with simplicity, ensuring that overlays provide just enough information without overwhelming or distracting the user. This focus underscores that even the most advanced AR technologies can fail if the interface is confusing or intrusive, making thoughtful interaction design critical to user adoption, safety, and overall experience quality.

1.2.4 Types of AR Guiding Systems

AR-guiding systems come in many forms, tailored to different use cases, environments, and user needs [34]. They vary in terms of the senses they engage (visual, auditory, haptic), the devices they run on, and the ways they deliver guidance.

Below are some key types:

- Visual Overlays;
- Audio AR Guidance;
- Haptic Feedback Systems;

- Multimodal Guidance;
- Marker-Based vs Markerless Systems;
- Indoor vs Outdoor Guidance.

Visual overlays is a defining feature of AR guiding systems, describing how they deliver essential information, like directions, instructions, or contextual cues directly onto the user's view of the real world. This highlights the strength of overlays in reducing cognitive load, as users no longer need to mentally translate between maps or manuals and their surroundings. By integrating information seamlessly into the environment, overlays enable faster decision-making and greater situational awareness. The analysis reveals that the effectiveness of visual overlays depends on clarity, alignment accuracy, and timing. Any misalignment or delay can quickly undermine trust and usability, turning helpful guidance into a source of confusion.

Some examples or visual overlays are:

- Arrows on the ground or walls showing the path to follow;
- Highlighted outlines or labels on objects of interest;
- Step-by-step maintenance instructions projected onto equipment;
- Museum exhibits with augmented reconstructions or labels.

Designing effective overlays is not simply a technical task but also a human centered challenge. It emphasizes the need to avoid clutter or information overload by carefully selecting and prioritizing the displayed content. Effective overlays must adapt to context. For example, offering more detail when needed while remaining minimal in dynamic or safety critical situations. This underscores the importance of balancing informativeness with simplicity to ensure that AR guiding systems remain genuinely helpful rather than distracting. Overall, visual overlays are both the most visible and arguably the most critical part of the AR guiding experience, serving as the interface where technology meets human perception and action.

Audio AR guidance can be a powerful and often underappreciated complement or alternative to visual overlays in AR guiding systems. By delivering information through spatial audio cues, spoken instructions, or sound effects, audio AR enables hands-free, heads-up navigation that keeps users engaged with their environment instead of fixating

on a screen. Audio guidance's has a particular value in situations where visual attention is limited or safety is paramount, such as pedestrian navigation in busy streets or assisting visually impaired users. The analysis reveals that audio AR expands accessibility and inclusivity, demonstrating that effective guidance need not to be purely visual but can leverage our multimodal sensory capabilities to create more natural, less distracting experiences.

Some examples of audio AR guidance are the followings:

- Turn-by-turn spoken directions in outdoor navigation;
- Spatial audio cues indicating the direction of the next waypoint;
- Museum tours where the audio adapts to the user's position and orientation.

The challenges of designing effective audio AR guidance are many. There is a need for clarity, timing, and context sensitivity to avoid cognitive overload or confusion, especially in noisy environments where auditory cues might be masked or misinterpreted. There are also opportunities for personalization. For example, adjusting voice styles, volume or languages to suit user's preferences. This makes clear that audio AR is not just a supplementary feature but a critical design dimension that can greatly enhance usability, inclusivity, and situational awareness in AR guiding systems when thoughtfully implemented.

Haptic feedback systems are valuable sensory channels in augmented reality guiding systems, especially when visual or auditory input is impractical or overloaded. Haptic feedback involves tactile cues such as vibrations, taps or pulses delivered through wearable devices like smart watches, gloves, or belts. These cues can guide users discreetly, such as by indicating direction through vibration patterns or confirming interaction with a virtual object. Haptic systems add an additional layer of interaction that is non-intrusive and can enhance awareness without demanding visual or auditory attention. This makes them particularly useful in safety critical environments or when supporting users with sensory impairments. Tactile cues must be carefully designed to be intuitive and distinguishable, as overly complex or ambiguous vibrations may confuse users rather than help them. The placement of actuators, intensity of feedback, and duration of signals all affect how effectively the information is conveyed. Additionally, haptic feedback must be synchronized with other modalities to maintain

consistency across the AR experience. Overall, while haptic feedback may not replace visual or audio guidance, it can significantly enrich the user experience when integrated thoughtfully, offering ingenious, context-aware reinforcement that improves both navigation and interaction.

Multimodal guidance in augmented reality systems refers to the integration of multiple sensory channels like visual, auditory and haptic to deliver instructions and information in a more natural, effective, and user-friendly way. This approach recognizes that users interact with their environment using all of their senses and that relying on a single mode of communication can be limiting or even unsafe in certain contexts. For example, visual overlays might be ideal for detailed navigation in clear conditions, but audio prompts can ensure attention remains on the environment while walking, and haptic cues can deliver silent, discreet directions in noisy or crowded areas. By combining these modes, multimodal guidance reduces cognitive load, improves comprehension, and enhances overall situational awareness. At the same time, designing effective multimodal systems requires careful consideration of how these channels work together. Overloading the user with redundant or conflicting signals can create confusion rather than clarity. Instead, each mode should complement the others, reinforcing critical information or offering alternative delivery when one channel is unavailable or impractical. For instance, an AR navigation app might show an arrow on a headset display, read out the next turn via audio, and use a minor vibration to confirm the turn direction. This layered approach enhances accessibility for users with sensory impairments or in challenging environments and personalizes guidance to individual preferences. Ultimately, multimodal guidance aims to make AR experiences more intuitive, inclusive, and resilient in real-world conditions.

Marker-based and marker less systems are two primary approaches to tracking and registration in augmented reality guiding systems, each with distinct advantages and limitations. Marker-based systems rely on predefined visual patterns or codes such as QR codes or confidential markers placed in the environment. These markers serve as reliable reference points for the AR system to recognize and track, enabling precise, stable alignment of virtual content even in challenging lighting or feature-poor settings. They provide clear, predictable anchors. Marker-based approaches are popular in controlled environments like museums, factories, or training labs where setup can be planned in advance, ensuring accuracy and ease of implementation. In contrast, marker less systems use natural features in the environment such as walls, objects, textures, or

spatial geometry to determine position and orientation without the need for special markers. Techniques like simultaneous localization and mapping (SLAM) or plane detection allow these systems to function dynamically in unfamiliar or changing spaces, making them ideal for large-scale navigation, outdoor use, or consumer applications where pre-placing markers isn't feasible. However, marker less systems can face challenges with accuracy in environments lacking distinct visual features, or in varying lighting conditions. While marker-based systems offer reliability and simplicity in known settings, marker less systems provide flexibility and scalability, and many modern AR applications aim to combine both approaches for the best of both worlds.

Indoor and outdoor guidance in augmented reality present distinct challenges and opportunities, reflecting the differences in environment, available technologies, and user needs. Outdoor AR guidance typically relies heavily on GPS and large-scale mapping data to provide navigation across cities, parks, or campuses. Because users often need to travel longer distances with fewer obstructions, outdoor guidance emphasizes broad geographic coverage, route optimization, and context-aware overlays like street names or landmarks. However, outdoor AR must also contend with variable lighting, weather, and signal interference, making robust tracking and clear visual design essential for maintaining usability in changing conditions. In contrast, indoor AR guidance [35] faces the challenge of GPS unreliability or unavailability, requiring alternative methods such as Wi-Fi fingerprinting, Bluetooth beacons, visual markers, or advanced SLAM (simultaneous localization and mapping) for accurate positioning. Indoor environments are often more complex, with multiple floors, corridors, and obstacles that demand highly precise, responsive guidance to avoid user confusion. Use cases like airport navigation, shopping malls, hospitals, and museums also place an overlay that must adapt to crowded spaces and personalized destinations. Overall, the distinction between indoor and outdoor guidance highlights the need for AR systems to tailor their tracking, mapping, and interface strategies to suit the unique constraints and expectations of each environment.

1.2.5 Fields of Applications for AR Guiding Systems

Augmented Reality guiding systems have found use in many industries and con-texts [36]. Their ability to deliver context-sensitive, spatially aligned and personalized information has made them a powerful tool for improving navigation, training, learning, and safety.

Below are some key application domains with illustrative examples:

- Tourism and Cultural Heritage;
- Museums and Exhibitions;
- Industrial Maintenance and Training;
- Navigation in Indoor and Outdoor Spaces;
- Education and Training;
- Healthcare;
- Retail and Customer Experiences.

Tourism and cultural heritage are areas where augmented reality (AR) guiding systems can significantly enhance visitor experiences by making history and culture more vivid, interactive, and accessible. Using AR, tourists can see reconstructions of ancient buildings overlaid on ruins, watch historical events re-enacted in their original locations, or access detailed stories about artifacts in museums simply by pointing their device. This transforms passive sightseeing into an engaging, educational journey, helping visitors connect more deeply with places and their histories. AR can also support multilingual guidance, personalized itineraries, and accessible content for people with disabilities, broadening the appeal of cultural sites to a diverse range of travelers. At the same time, effective AR design for tourism and heritage requires careful attention to authenticity, user experience, and context. Successful AR guidance needs to respect the integrity of real-world locations while adding meaningful, well-curated information that enhances understanding without overwhelming visitors. Additionally, AR can help manage visitor flows, promoting sustainable tourism by guiding people along less crowded routes or offering remote virtual visits. By thoughtfully integrating digital storytelling with cultural preservation, AR guiding systems can both enrich tourism experiences and help safeguard heritage for future generations.

In museums and exhibitions, augmented reality guiding systems offer powerful tools for enriching visitor engagement, learning, and enjoyment. By overlaying additional information, or 3D reconstructions onto physical exhibits, AR can bring static displays to life in ways that traditional labels and audio guides cannot. For example, a visitor might use a smartphone or AR headset to see how an ancient artifact was originally used, explore its internal structure, or watch a historical scene unfold around it. Such interactive storytelling can make complex topics more understandable and appealing to

diverse audiences, including children and those without specialist knowledge, transforming a passive viewing experience into an immersive exploration. At the same time, successful use of AR in museums and exhibitions depends on thoughtful integration with curatorial goals and the physical space. AR experiences need to be carefully designed to complement, not overshadow, the original artifacts and artworks, preserving the authenticity and atmosphere of the exhibition. Clear, intuitive interfaces are essential so that technology remains accessible to all visitors without requiring extensive instructions or disrupting the flow of exploration. Moreover, museums must consider practical factors such as device availability, maintenance, and visitor support. When well executed, AR guiding systems can expand the educational mission of museums, making cultural heritage more engaging, inclusive, and meaningful for all visitors.

In industrial maintenance and training [37], augmented reality guiding systems provide critical support by delivering step-by-step instructions, diagnostics, and visual cues directly onto equipment and workspaces. Technicians wearing AR headsets or using tablets can see overlays that identify components, highlight problem areas, or guide them through complex procedures with 3D animations. This hands-free, context-aware assistance reduces errors, speeds up maintenance tasks, and ensures consistent adherence to safety protocols and standards. In training scenarios, AR can simulate real-world tasks in a safe, controlled environment, giving workers practical experience without the risks or costs of live equipment, thereby improving learning outcomes and workforce readiness. At the same time, effective deployment of AR in industrial settings requires attention to usability, reliability, and integration with existing workflows. Industrial environments can be challenging, with varying lighting, noise, and connectivity that can interfere with tracking and content delivery. AR systems must be robust, intuitive, and comfortable for extended use, ensuring that technology aids rather than distracts the worker. Additionally, content must be kept accurate and up-to-date as equipment and procedures evolve. When thoughtfully designed and integrated, AR guiding systems can greatly enhance productivity, reduce downtime, and improve safety, representing a transformative tool for modernizing industrial maintenance and workforce training.

Navigation in indoor and outdoor spaces is one of the most practical and impactful applications of augmented reality guiding systems, helping users move confidently

through unfamiliar environments. Outdoors, AR navigation typically relies on GPS and large-scale map data to overlay directions, landmarks, or points of interest onto the user's real-world view via a smartphone or AR headset. This can make urban way finding more intuitive by showing arrows on sidewalks, labeling streets, or even providing real-time transit information directly in context. For tourists, delivery drivers, or pedestrians, such guidance reduces reliance on abstract maps and written directions, lowering cognitive load and improving safety by allowing users to keep their eyes on their surroundings. Indoors, navigation poses different challenges and opportunities, since GPS is often unreliable or unavailable. AR systems for indoor navigation rely on technologies like Bluetooth beacons, Wi-Fi fingerprinting, visual markers, or advanced computer vision techniques (like SLAM) to localize users accurately within complex spaces such as airports, malls, hospitals, or museums. Once localized, AR can overlay clear, intuitive paths through corridors, stairways, and multiple floors, offering accessible guidance for all users, including those with disabilities. This reduces confusion, saves time, and improves overall user experience in large or unfamiliar buildings. Together, indoor and outdoor AR navigation systems demonstrate the power of blending digital directions seamlessly with the physical world to create a more connected, understandable, and user-friendly environment.

Augmented reality guiding systems are revolutionizing education and training by creating immersive, interactive learning experiences that go beyond traditional methods. In classrooms, AR can bring abstract concepts to life and students can visualize complex scientific processes, historical events, or anatomical structures in 3D, making learning more engaging and easier to understand. In professional training, AR offers hands-on practice in a controlled, risk-free environment, allowing learners to simulate real-world tasks, receive immediate feedback, and build confidence before performing actual operations. This experiential approach not only improves retention but also provides diverse learning styles by combining visual, auditory, and kinesthetic elements. At the same time, the successful use of AR in education requires careful design to ensure content aligns with course of study goals and learning outcomes. The technology must be accessible, easy to use, and seamlessly integrated into lesson plans or training programs to avoid becoming a distraction. Additionally, AR can support personalized learning by adapting to individual progress and offering tailored guidance, making education more inclusive and effective. When thoughtfully implemented, AR guiding systems enhance both formal and informal education by fostering deeper

understanding, increasing motivation, and preparing learners for practical challenges in real-world contexts.

Augmented reality guiding systems are increasingly being adopted in healthcare to support medical professionals and improve patient outcomes. In surgical settings, AR can overlay critical information such as anatomical structures, real-time imaging, or preoperative plans, directly onto the surgeon's field of view, enhancing precision and reducing risks during complex procedures. For medical training, AR provides realistic simulations that allow students and practitioners to practice techniques on virtual patients, gaining valuable experience without exposing real patients to risk. Beyond surgery, AR can assist with patient rehabilitation by guiding movements or exercises, improving compliance and monitoring progress through interactive feedback. However, deploying AR in healthcare demands strict attention to accuracy, reliability, and safety, given the high stakes involved. The technology must integrate seamlessly with existing medical equipment and workflows, providing intuitive and unobtrusive support that enhances rather than distracts healthcare providers. Patient privacy and data security are also paramount considerations. Despite these challenges, AR guiding systems hold significant promise for transforming healthcare delivery by improving training, enhancing surgical outcomes, and empowering patients through personalized, real-time guidance, ultimately contributing to more effective and efficient medical care.

Augmented reality guiding systems are reshaping retail and customer experiences by blending digital information with physical shopping environments, creating more interactive and personalized journeys. In stores, AR can help customers visualize products in real time, whether it's trying on virtual clothes, previewing furniture in their homes, or exploring detailed product features through interactive displays. This level of engagement reduces uncertainty, enhances decision-making, and often leads to higher satisfaction and sales. AR also supports way finding within large retail spaces, guiding shoppers directly to items on their lists and improving convenience. At the same time, AR enables retailers to craft unique, memorable brand experiences that differentiate them from competitors. By incorporating gamification, storytelling, and immersive promotions, AR can attract and retain customers in an increasingly digital marketplace. However, designing these systems requires a balance between innovation and usability. Overlying complex interfaces or intrusive overlays may frustrate users or detract from the shopping experience. When thoughtfully integrated, AR guiding systems offer

retailers powerful tools to deepen customer engagement, personalize service, and bridge the gap between online and offline shopping.

Summarizing, AR-guiding systems are incredibly versatile, spanning tourism, museums, industry, healthcare, education, retail, and more. Their strength lies in adapting information to the user's context and delivering it precisely where and when it's needed, enhancing usability, safety, learning, and engagement.

1.2.6 Design Considerations for AR Guiding Systems

Designing effective AR-guiding systems is far more than just overlaying graphics on the real world [38,39]. It requires careful attention to user needs, technological limits, safety, and usability. A well-designed AR-guiding system minimizes user frustration, enhances task performance, and delivers value without causing confusion or distraction.

Below are key design considerations to address when creating such systems:

- User Experience (UX) and Interface Design;
- Safety and Situational Awareness;
- Interaction Design;
- Environmental Adaptation;
- Accessibility;
- Technical Performance;
- Content Management;
- Privacy and Security;
- Cultural and Social Context;
- Evaluation and Iteration.

User Experience (UX) and Interface Design are critical elements in the success of augmented reality guiding systems, shaping how effectively users can access and interact with information in real-world contexts. Unlike traditional screens, AR interfaces must seamlessly blend digital overlays with the physical environment, requiring careful design to avoid clutter, confusion, or distraction. Clear, intuitive visuals, well-timed prompts, and context-aware guidance are essential to ensure that users can easily understand and act on the information presented. Good AR UX prioritizes accessibility and inclusivity, providing options for different languages, sensory needs, or levels of technical familiarity so that a wide range of users can benefit

from the system. At the same time, interface design for AR guiding systems must consider practical factors like user comfort, device limitations, and environmental variability. For example, overlays must remain legible in changing light conditions, and interactions should work reliably even when users' hands are full or their attention is divided. Designers need to ensure that controls are simple and ergonomic, minimizing cognitive load while supporting complex tasks when necessary. When thoughtfully executed, strong UX and interface design transform AR from a novel technology into a genuinely helpful tool that enhances navigation, learning, and decision-making in real-world settings, ensuring that users feel empowered rather than overwhelmed by the digital layer added to their experience.

Safety and situational awareness are central concerns in the design and deployment of augmented reality guiding systems, especially because these systems operate directly in users' physical environments. AR has the potential to enhance safety by providing timely, context-sensitive information, such as hazard warnings, navigation cues, or real-time instructions without requiring users to divert their attention to a separate screen. For example, workers on an industrial site can receive overlaid alerts about dangerous equipment zones, while drivers might see turn-by-turn directions projected onto their windshield to reduce distraction. By aligning guidance with the user's real-world view, AR can help maintain awareness of surroundings and support safer decision making. However, these benefits depend on careful design to avoid introducing new risks. Poorly placed or overly complex overlays can obscure critical real-world details, cause cognitive overload, or distract users at the wrong moment. Designers must ensure that AR content is clear, minimal, and adaptive to context. For example, reducing visual clutter when users are moving quickly or in high-risk settings. Systems should also account for environmental factors like lighting or weather that might affect visibility and tracking accuracy. Ultimately, prioritizing safety and situational awareness in AR guiding systems is not just about adding more information but about delivering the right information at the right time, in a way that genuinely supports users in staying alert, oriented, and secure in their environment.

Interaction design in augmented reality guiding systems focuses on creating intuitive, efficient, and enjoyable ways for users to engage with digital content seamlessly layered onto the real world. Unlike traditional interfaces confined to screens, AR inter-action design must account for the spatial and physical context of users, enabling natural

gestures, voice commands, eye tracking, or even haptic feedback to control and manipulate virtual elements. For instance, a user might point at a building to get directions, use a voice command to request information, or tap virtual buttons floating in space. The goal is to make these interactions feel fluid and instinctive, reducing the barrier between user intent and system response. At the same time, effective interaction design must carefully balance richness of functionality with simplicity and safety. Overly complex gestures or crowded interfaces can confuse users or distract them from their surroundings, especially in contexts like navigation, maintenance, or driving where attention must remain on the real world. Designers need to prioritize clarity, responsiveness, and context-awareness, ensuring that controls adapt to the user's environment and task. This means interactions should be easy to learn but robust enough to handle varied use cases and conditions. When done well, interaction design transforms AR from a technological novelty into a practical, user-centered tool that enhances understanding, decision-making, and overall experience in daily life.

Environmental adaptation is a critical aspect of augmented reality guiding systems, ensuring that digital content remains usable, relevant, and effective across diverse real world conditions. AR applications need to function reliably in changing lighting, from bright sunlight to dim indoor settings and handle obstacles, reflections, or cluttered backgrounds without losing tracking accuracy. For example, an AR navigation system must adjust overlays to remain visible in glare or adapt guidance when a planned route is blocked. This adaptability enhances usability by maintaining a consistent, clear experience regardless of where or when the system is used, supporting confident and stress free interaction even in unpredictable environments. At the same time, environmental adaptation goes beyond technical tracking to include the contextual relevance of information delivered. AR systems should be context-aware, presenting content tailored to the user's current surroundings and needs. For instance, in a museum, the system might switch between detailed explanations and concise summaries based on crowd density or user preference, while an outdoor guide might highlight landmarks only when they come into view [40,41]. Such sensitivity ensures that AR enhances rather than disrupts the user's situational awareness and comfort. Ultimately, strong environmental adaptation reflects thoughtful, human-centered design that respects the complexity of the real world, making AR guiding systems truly practical and beneficial for everyday use.

Accessibility in augmented reality guiding systems is about ensuring that these technologies are usable and beneficial to as many people as possible, regardless of their abilities or limitations. Well-designed AR can actually improve accessibility by providing alternative ways to perceive and interact with information. For example, visual overlays can include text-to-speech features for users with low vision, while audio cues can support those who struggle with reading or navigating complex spaces. AR can also offer sign language interpretations, adjustable font sizes, or customizable color contrasts, tailoring the experience to individual needs. By making guidance intuitive and multi-sensory, AR has the potential to reduce barriers that traditionally exclude people from independently navigating public spaces, museums, workplaces, and more. At the same time, true accessibility requires careful design choices and testing to avoid introducing new obstacles. Overly complex interfaces, small text, or gesture controls that require fine motor skills can exclude users with certain disabilities. Designers must prioritize simplicity, clarity, and flexibility, offering multiple input and output modes so users can choose what works best for them. Additionally, AR systems should be compatible with assistive technologies that people already use, ensuring seamless integration into their daily routines. By embedding accessibility as a core principle rather than an afterthought, developers can create AR guiding systems that not only meet legal or ethical obligations but also unlock richer, more independent experiences for everyone.

Technical performance is a crucial factor in the effectiveness of augmented reality guiding systems, determining how smoothly and reliably users can interact with digital overlays in real-world contexts. High-quality AR experiences require precise tracking, fast and stable rendering, and real-time responsiveness to user movements and environmental changes. For example, if an AR navigation system lags or misaligns directions, it can confuse users or even create safety hazards. Similarly, training or maintenance guidance that freezes or jitters breaks immersion and reduces trust in the system. To deliver reliable performance, developers need to optimize software for low latency, ensure compatibility with various hardware platforms, and design for diverse conditions such as variable lighting, connectivity, and processing power. At the same time, maintaining strong technical performance requires balancing resource demands with usability and accessibility. AR applications can be computationally intensive, draining battery life or overheating devices if not optimized properly. Systems must be designed to function smoothly even on lower-end smartphones or headsets, broadening

access while ensuring a consistent user experience. Moreover, developers need to address challenges like data security, bandwidth for streaming content, and offline functionality where network access is limited. By prioritizing robust technical performance, AR guiding systems can offer dependable, high-quality experiences that users can trust in critical scenarios, from navigating busy streets to performing complex industrial tasks.

Content management is a vital component of augmented reality guiding systems, as it ensures that the digital information users see is accurate, up-to-date, and relevant to their context [42,43]. Unlike static applications, AR guidance often relies on dynamic, location-specific, or highly detailed content, such as annotated 3D models, maps, multilingual text, or multimedia explanations. Effective content management involves organizing, storing, and delivering this data efficiently so users always receive the right information at the right time. For example, a museum's AR guide needs to update exhibit details as collections change, while an industrial maintenance system must reflect the latest procedures or equipment updates to ensure safety and compliance. At the same time, managing AR content requires robust tools and workflows that support easy creation, editing, and distribution. Curators, trainers, or local businesses need intuitive platforms to add or modify AR experiences without requiring advanced technical skills. Additionally, systems must ensure consistency across devices and platforms while supporting localization and personalization for diverse user groups. Data integrity and security are also critical, especially when content includes sensitive information or user generated inputs. By prioritizing strong content management practices, developers can keep AR guiding systems relevant, scalable, and sustainable, ensuring they continue to deliver meaningful experiences as user needs and environments evolve.

Privacy and security are critical concerns in augmented reality guiding systems, as these technologies often rely on collecting and processing large amounts of real-world data, including location, surroundings, user preferences, and behavioral patterns. This data can enhance user experience by enabling personalized and context-aware guidance, but it also raises significant risks if not properly managed. For example, an AR app that accesses a user's camera and location could potentially be exploited to track movements or gather sensitive information without consent. In public or shared spaces, the risk extends to bystanders, whose images or environments might be captured unintentionally. These concerns make it essential for developers to implement clear data

policies, informed consent mechanisms, and options for users to control what is collected and shared. On the security side, AR guiding systems must protect against unauthorized access, data breaches, and malicious manipulation of content. A compromised AR system could mislead users with false overlays, direct them into unsafe areas, or disrupt training and operational processes in high-stakes environments like healthcare or industrial maintenance. Strong encryption, secure data transmission, and regular software updates are essential components of a secure AR infrastructure. Additionally, developers must consider device-level protections, especially for mobile and wearable platforms that are more vulnerable to physical tampering or theft. Ultimately, building trust in AR guiding systems requires a strong commitment to privacy and security by design, ensuring that users can benefit from immersive, intelligent experiences without compromising their personal rights or safety.

Cultural and social context is a crucial consideration in the design and deployment of augmented reality guiding systems, as these technologies do not exist in a vacuum, they shape, and are shaped by, the societies in which they are used. AR guidance must respect local customs, languages, and norms to ensure it is meaningful and inclusive for diverse user groups. For example, an AR tour guide in a historical site should present culturally sensitive interpretations, avoiding stereotypes or offensive content while offering options for multiple languages or perspectives. Similarly, gestures or interface symbols that are intuitive in one culture might be confusing or even inappropriate in another. By understanding and accommodating these differences, designers can create AR experiences that resonate with users and foster cross-cultural understanding rather than alienation. At the same time, social dynamics influence how people perceive and adopt AR technology. In some contexts, using AR headsets or smartphones in public may be seen as intrusive, antisocial, or even threatening to privacy. Designers must consider how to make AR interactions discreet, socially acceptable, and respectful of bystanders who may be unintentionally recorded or observed. There’s also the question of equity: ensuring AR guiding systems are accessible and affordable so they do not widen existing digital divides. By engaging with local communities, incorporating user feedback, and respecting cultural values, AR developers can create guiding systems that enhance social connection, support cultural heritage, and provide genuinely helpful, context-aware guidance that fits naturally into users’ everyday lives.

Evaluation and iteration are essential processes in developing effective augmented reality guiding systems, ensuring they truly meet user needs and perform well in real-world contexts. Unlike static products, AR systems involve complex interactions with the physical environment, so careful testing is required to verify tracking accuracy, usability, and contextual relevance. This means observing users as they navigate with AR guidance, identifying pain points such as confusing overlays, delayed responses, or poor visibility in different lighting conditions. Structured evaluation through usability studies, field trials, and feedback sessions provides critical insights into what works and what doesn't, grounding design decisions in actual user experience rather than assumptions. At the same time, iteration is about using this feedback to continuously refine and improve the system. Rather than treating design as a one-time task, successful AR development embraces cycles of prototyping, testing, and updating to address evolving needs and technologies. For example, new hardware capabilities might enable smoother tracking or richer interactions, while user feedback might reveal a need for clearer instructions or better accessibility features. By prioritizing evaluation and iteration, developers ensure that AR guiding systems remain reliable, intuitive, and genuinely useful over time. This user-centered approach not only enhances satisfaction and adoption but also helps AR realize its promise as a practical, meaningful tool that integrates seamlessly into daily life.

1.2.7 Benefits and Technical Challenges

AR-guiding systems hold enormous promise to improve navigation, learning, training, and user experiences [44-46]. But they also come with important challenges that need to be addressed for successful deployment. This section examines both sides, the key benefits that drive adoption, and the challenges that must be overcome.

1.2.7.1 Benefits of AR Guiding Systems

Augmented reality guiding systems offer a wide range of transformative benefits that fundamentally change how people access, interpret, and use information in the real world. By seamlessly layering digital content onto the physical environment, these systems make information immediately available and contextually relevant, removing the friction of switching between screens or searching through manuals. For instance, travelers in unfamiliar cities can follow overlaid arrows or contextual markers that make navigation intuitive and stress-free, while museum visitors can see historical scenes or reconstructions appear around them, deepening their understanding and appreciation of

cultural heritage. This ability to anchor guidance to real-world locations and objects creates a powerful sense of immediacy and relevance that traditional media cannot replicate.

The main benefits can be considered to be:

- Enhanced Spatial Understanding;
- Contextual, Just-in-Time Information;
- Increased Engagement and Motivation;
- Improved Accessibility;
- Hands-Free Operation;
- Operational Efficiency;
- Personalization.

Beyond navigation, AR guiding systems excel in delivering step-by-step instructions and real-time feedback in professional and educational settings, supporting safer, more efficient, and more accurate work. Technicians performing complex maintenance can see overlays indicating which parts to inspect or replace, reducing errors and training time. In healthcare, surgeons might view real-time imaging or procedural cues superimposed on a patient, enhancing precision and outcomes. For students and trainees, AR enables immersive, hands-on learning experiences that promote better retention and understanding compared to static lectures or printed materials. By integrating guidance directly into real-world contexts, AR empowers users to act confidently even in high stakes or unfamiliar scenarios.

Another important benefit is the ability of AR guiding systems to create more engaging, personalized, and memorable experiences that drive user satisfaction and loyalty. In retail, customers can try on clothes virtually, see how furniture fits in their homes, or access detailed product information through interactive overlays, bridging the gap between online and in-person shopping. Tourism and entertainment industries can use AR to tell richer stories that captivate visitors, encouraging repeat visits and word of mouth promotion. By supporting multiple modalities, visual overlays, audio guidance, or haptic feedback, AR can also tailor experiences to individual needs and preferences,

increasing accessibility for users with diverse abilities and ensuring that everyone can benefit equally from the technology.

Finally, AR guiding systems offer significant societal and economic benefits by unlocking new business opportunities, enhancing productivity, and supporting inclusivity. Cities and transportation hubs can improve way finding and reduce congestion by guiding travelers more effectively. Industries can lower training costs and improve worker safety, while educational institutions can offer cutting edge learning experiences without needing expensive physical models or field trips. Moreover, by providing accessible, multimodal guidance, AR can help remove barriers for people with disabilities, making public spaces and services more inclusive. In sum, when designed thoughtfully and deployed responsibly, AR guiding systems don't just deliver information—they transform the way people experience, navigate, and interact with the world around them, offering profound benefits across personal, professional, and societal dimensions.

1.2.7.2 Challenges of AR Guiding Systems

While the benefits are compelling, deploying AR-guiding systems at scale faces substantial challenges like:

- Technical Limitations;
- Usability Issues;
- Cost and Infrastructure;
- Privacy and Security Concerns;
- Social Acceptance;
- Accessibility Gaps;
- Environmental Constraints.

Despite their many advantages, augmented reality guiding systems face significant technical challenges that can limit their effectiveness and adoption. One of the most fundamental issues is achieving robust and accurate tracking and registration in real-world environments. AR systems must continuously align digital content with the user's physical surroundings, which is complicated by variable lighting, reflections, clutter, or fast user movement. Poor tracking can lead to misaligned overlays or jittery visuals that break immersion and reduce trust in the guidance provided. Additionally, ensuring real-

time responsiveness without lag requires significant processing power, which can strain device hardware, shorten battery life, and limit use on lower-end smartphones or headsets. Developers must work continuously to improve tracking algorithms, optimize performance, and support a wide range of devices and environments to deliver a consistently reliable experience.

Another major challenge lies in designing user interfaces and interactions that are intuitive, safe, and usable in dynamic contexts. AR guiding systems must balance rich, context-sensitive information with the need to avoid overwhelming or distracting users, especially in high-stakes settings like driving, industrial maintenance, or healthcare. Poorly designed overlays can obscure important real-world features or lead to cognitive overload, reducing situational awareness and potentially causing accidents. Gesture controls, voice commands, and gaze tracking offer natural input methods but can be unreliable or difficult to use in noisy, crowded, or visually complex environments. Effective interaction design requires deep understanding of user needs and contexts, extensive testing, and the flexibility to adapt to different situations and user preferences.

Content management also presents significant challenges for AR guiding systems. Unlike static signage or apps, AR guidance often depends on dynamic, location-specific, and frequently updated content. Museums must continually revise exhibit details, cities may change traffic patterns, and businesses update inventory or layouts. All of which require efficient systems to create, manage, and distribute new AR content quickly and accurately. Ensuring consistency and quality across languages and cultures adds complexity, as does maintaining data security and integrity when content is edited or shared by multiple stakeholders. Without robust content management tools and workflows, AR guiding systems risk delivering outdated, incorrect, or culturally insensitive information that undermines their credibility and usefulness.

Finally, broader societal challenges must be addressed to ensure AR guiding systems are adopted responsibly and inclusively. Privacy and security concerns are paramount, as these systems often rely on continuous camera and location data that could be misused for surveillance or tracking without proper safeguards. Social acceptance is another hurdle. Wearing AR headsets or holding up smartphones in public can feel intrusive or antisocial, and may not be equally embraced across cultures or age groups. There is also the risk of exacerbating digital divides if AR guidance depends on expensive devices or high-speed connectivity, limiting access for economically

disadvantaged communities. Developers and policymakers need to consider these ethical, cultural, and economic dimensions carefully, embedding privacy protections, inclusive design, and equitable access into the core of AR guiding systems to ensure they truly benefit everyone.

1.2.8 Case Studies and Real-World Examples of AR Guiding Systems

To better understand the impact and practical implementation of augmented reality guiding systems, it helps to look at real-world examples across different industries [47,48]. These case studies show how AR is transforming navigation, training, tourism, and more, while also highlighting lessons learned from actual deployments.

One of the most prominent examples is the Google Maps Live View, which integrates AR guiding technology to assist pedestrians in navigating complex urban environments. By overlaying arrows and street signs onto the smartphone camera view, users can see exactly where to turn or which building to look for. This system leverages a vast database of Street View images combined with visual positioning to achieve accurate localization. It has proven especially useful for tourists or people unfamiliar with a city, eliminating the need to constantly shift attention between a traditional map and the physical world. The success of this system underscores the importance of seamless integration between AR content and reliable localization technologies.

In the industrial sector, Boeing has successfully employed AR-guided assembly systems in its aircraft manufacturing lines. Technicians use AR headsets to receive step-by-step instructions overlaid directly onto aircraft parts, significantly reducing errors and assembly times. These AR instructions replace bulky manuals and complex paperwork, allowing workers to keep their hands free and focus on the task. The Boeing case demonstrates how AR can improve operational efficiency and worker safety by providing precise, context-aware guidance. However, it also highlights challenges such as the need for highly accurate tracking and ergonomically designed headsets to ensure user comfort over extended periods.

Museums and cultural heritage sites have also embraced AR guiding systems to enrich visitor experiences. The British Museum experimented with AR applications that provide layered information and historical reconstructions directly on exhibits. Visitors using smartphones or tablets can see additional details about artifacts, such as the original appearance of damaged objects or 3D models illustrating how items were used in the past. This approach increases engagement and learning outcomes but also

requires continuous content updates and maintenance to stay relevant. It emphasizes the importance of intuitive interfaces and minimizing cognitive overload to ensure visitors remain immersed in both real and augmented worlds.

Healthcare is another domain where AR guiding systems have begun to make an impact. The AccuVein device projects vein maps onto patients' skin to assist nurses and doctors in locating veins for injections or intravenous (IV) placements. This AR guidance reduces failed attempts and patient discomfort, illustrating how AR can enhance precision and improve outcomes in critical tasks. Similarly, surgeons have adopted AR overlays during operations to visualize underlying anatomy based on imaging data, improving accuracy while reducing invasiveness. These applications highlight both the promise and stringent safety requirements for medical AR systems.

In the retail environment, IKEA's Place app has revolutionized how customers interact with furniture shopping by allowing users to place virtual models of furniture items in their own homes via AR. This system helps customers visualize scale, style, and fit before purchasing, reducing returns and increasing satisfaction. It combines AR-guided positioning with product information, representing a powerful example of how AR guides decision-making. However, it also reveals challenges around device compatibility, lighting conditions in homes, and the accuracy of virtual object placement.

Finally, the Soundscape app developed by Microsoft provides a compelling case for audio-based AR guidance, designed especially to assist visually impaired users. Instead of relying on visual overlays, Soundscape uses 3D spatial audio cues to inform users about nearby points of interest and navigation directions. This approach demonstrates the potential of multimodal AR guiding systems to cater to diverse user needs and promote accessibility, while also highlighting the importance of environment-aware audio design to avoid confusion or information overload.

These case studies collectively illustrate the transformative potential of AR guiding systems in real-world contexts. They show how successful implementations rely on robust localization technologies, user-centered design, and thoughtful integration with existing workflows. At the same time, they underscore persistent challenges including hardware limitations, environmental variability, and the need for ongoing content management. Understanding these successes and obstacles provides valuable insights for future development and adoption of AR-guiding technologies.

1.2.9 Trends and Future Directions in AR Guiding Systems

As augmented reality technology continues to evolve, AR guiding systems are poised for significant advancements that will further enhance their utility, usability and reach content [49,50]. Emerging trends indicate that future AR guides will become more intelligent, adaptive, and seamlessly integrated into everyday life.

One of the key future developments lays in the integration of artificial intelligence (AI) and machine learning. By leveraging AI, AR guiding systems will be able to better understand user behavior, preferences, and context, enabling them to deliver highly personalized guidance. For instance, AI can predict user intent based on past actions or environmental cues, dynamically adjusting navigation routes or instructional steps. This means AR guides will no longer be static overlays but adaptive companions that learn and evolve with the user, offering smarter and more intuitive support.

Another important trend is the advancement of sensor technologies and hardware factors. Future AR devices are expected to become lighter, more comfortable, and less obtrusive, facilitating longer use in diverse scenarios. Improvements in eye-tracking, gesture recognition, and haptic feedback will allow more natural and fluid interactions without the need for traditional controllers or touchscreens. Meanwhile, advances in sensors such as LiDAR, depth cameras, and inertial measurement units will dramatically improve tracking accuracy and robustness, enabling AR guidance in environments previously considered challenging, like featureless indoor spaces or outdoor areas with dynamic elements.

The rise of 5G connectivity and edge computing is also set to transform AR guiding systems by providing low-latency, high-bandwidth data transmission. This will support richer, cloud-based content streaming and real-time collaboration between users and remote experts. For example, a field technician could receive immediate AR annotations from a specialist located miles away, facilitating complex repairs or inspections without the need for travel. Moreover, edge computing will allow intensive data processing to happen closer to the user, reducing lag and preserving battery life on mobile or wearable devices.

In parallel, there is growing interest in multimodal AR guidance that combines visual, audio, and haptic feedback. Such systems can help to diverse user preferences and accessibility needs while minimizing cognitive overload. For example, a pedestrian navigation system might use vibrations in a smart wristband to indicate turns while

visually displaying arrows, making the experience safer and less distracting in busy urban environments. This holistic approach recognizes that not all users interact with AR the same way and that context plays a critical role in how information should be conveyed.

The development of standardized platforms and authoring tools is another promising direction. Currently, creating AR-guiding content often requires specialized skills and custom development, which can limit scalability and widespread adoption. As more intuitive, user-friendly platforms emerge, businesses and organizations will be able to design, deploy, and update AR guidance with less technical overhead. This democratization of AR content creation will accelerate innovation and broaden the range of applications.

Finally, ethical considerations and privacy preserving technologies will become increasingly important as AR guiding systems proliferate. Future systems are expected to incorporate privacy by design principles, such as on-device processing, anonymization of location data, and transparent user consent mechanisms. As AR devices become more embedded in everyday life, addressing societal concerns about surveillance and data security will be essential to building trust and encouraging adoption.

In summary, the future of AR guiding systems looks promising, characterized by smarter AI-driven guidance, enhanced hardware capabilities, seamless multimodal interactions, faster connectivity, and ethical design. These developments will help AR guiding systems move from niche applications to ubiquitous tools that enrich how people navigate, learn, work, and explore the world around them.

1.2.10 AR Guiding Systems Conclusions and Final Thoughts

Augmented reality guiding systems represent a significant leap forward in how humans interact with and navigate the physical world. By seamlessly overlaying digital information onto real environments, these systems reduce cognitive effort, enhance spatial awareness, and provide context-aware support that adapts to users' immediate needs. Across industries, from urban navigation and industrial maintenance to healthcare, tourism and retail, AR guiding systems have demonstrated their potential to improve efficiency, safety, accessibility, and user engagement.

Throughout this exploration, it has become clear that the success of AR guiding technologies depends on a thoughtful balance of multiple factors. Technological advancements in tracking, display hardware, and AI must be matched with human centered design principles that prioritize clarity, usability, and safety. Privacy and ethical considerations must be embedded from the outset to foster trust and widespread acceptance. Moreover, continuous iteration informed by real world testing ensures that AR guides remain relevant, accurate, and aligned with users' evolving expectations.

While challenges remain, including hardware limitations, environmental variability, cost barriers, and social acceptance, ongoing research and innovation are steadily addressing these issues. Emerging trends in AI, sensor technology, connectivity, and content authoring promise to make AR guiding systems more adaptive, natural, and accessible than ever before.

Looking ahead, the potential for AR-guided experiences extends beyond simple navigation. As AR becomes more integrated with the Internet of Things, digital twins, and smart environments, guiding systems will evolve into holistic assistants that support complex decision making, collaboration, and learning in real time. They will empower individuals to interact with their surroundings in ways that were once the realm of science fiction.

In conclusion, augmented reality guiding systems are assured to transform how we move through and understand the world, bridging the gap between digital information and physical reality. Their continued development and adoption will not only change practical tasks but also enrich human experiences—making navigation safer, learning more engaging, and everyday life more connected.

1.3 The Android Operating System

1.3.1 Introducing Android OS

The Android operating system is the world's most widely used mobile platform [51-53]. Developed by Google, Android powers billions of devices globally, from smartphones and tablets to TVs, cars, and wearables. Its open-source nature, broad device support, and customizable user experience have made it a dominant force in consumer electronics. Android's development began with Android Inc. in 2003, later acquired by Google in 2005. In 2008, the first commercial Android device launched, marking the start of an extraordinary growth trajectory that transformed the mobile industry.

Android Inc. was founded in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White. Their initial goal was to build an advanced operating system for digital cameras, but they quickly pivoted to mobile phones. In 2005, Google acquired Android Inc., recognizing the potential to disrupt the mobile space. Under Google, Android was positioned as an open-source platform to counter the dominance of proprietary systems like Symbian, BlackBerry OS, iPhone OS and Windows Mobile. Google formed the Open Handset Alliance (OHA) in 2007, a consortium of hardware manufacturers, software companies, and carriers committed to advancing open standards for mobile devices. In 2008, HTC released the first commercial Android device, the HTC Dream (also known as the T-Mobile G1). The early OS featured basic smartphone capabilities, integration with Google services, and support for third-party apps via the Android Market (now Google Play Store). Since then, Android has gone through numerous major version updates. Each version was traditionally named after a dessert in alphabetical order (Cupcake, Donut, Eclair, etc.) until version 10, when Google switched to a simple numeric naming convention.

As key versions we can consider the followings [53]:

- Android 1.5 Cupcake (2009): First to support third-party widgets;
- Android 2.3 Gingerbread (2010): Better UI and NFC support;
- Android 4.0 Ice Cream Sandwich (2011): Unified phone/tablet UI;
- Android 5.0 Lollipop (2014): Introduction of Material Design;
- Android 6.0 Marshmallow (2015): App permissions overhaul;
- Android 8.0 Oreo (2017): Background app limits;
- Android 10 (2019): System-wide dark mode;
- Android 12 (2021): Material You theming;
- Android 14 (2023): Performance and privacy improvements.

Android’s open nature helped manufacturers customize it for their devices, fueling its adoption across the globe. As of 2025, Android powers over 70% of smartphones worldwide.

1.3.2 The Architecture of Android

Android’s layered architecture is designed to balance flexibility, modularity, and performance, enabling diverse hardware manufacturers to build devices on a common platform while maintaining customization options [54,55]. At its core lies a modified Linux kernel, which provides fundamental security, process management, and hardware abstraction through device drivers. Above this, the Hardware Abstraction Layer (HAL) standardizes interactions between the OS and hardware components, reducing vendor fragmentation [56]. The Android Runtime (ART) [57] and native C/C++ [58] libraries deliver efficient application execution and essential services such as graphics rendering and media playback. The application framework layer exposes consistent APIs to developers, fostering a robust app ecosystem, while the top application layer ensures process isolation and sandboxing for security. This modular design enables Google to deliver system updates more flexibly via initiatives like Project Mainline, while allowing manufacturers to innovate on device features without altering core system stability, illustrating Android’s balance between openness and centralized control.

Android’s architecture is layered, designed for modularity, flexibility, and hardware abstraction and it consists of the following components:

- Linux Kernel;
- Hardware Abstraction Layer (HAL);
- Native Libraries and Android Runtime (ART);
- Application Framework;
- Users Applications.

Android's core functionality relies on the Linux kernel, which includes device drivers that facilitate communication between the operating system and hardware. These drivers act as translators, enabling the OS to interact with various components like the display, touch screen, camera, and more. Android leverages a modified version of the Linux kernel, specifically tailored for mobile devices, and incorporates Android specific patches. The Linux kernel serves as the foundation for Android, managing essential functions like process management, memory management, security, and networking. Android builds upon the existing Linux kernel, adapting it for mobile devices and adding its own features and security measures. Each Android version typically utilizes a specific version of the Linux kernel.

The Android Hardware Abstraction Layer (HAL) acts as a crucial intermediary between the Android framework and the device's hardware. It provides a standardized interface that allows higher-level Android services to interact with various hardware components (like cameras, Bluetooth, etc.) without needing to understand the specific details of each device's hardware. This modular design simplifies development and ensures compatibility across different hardware configurations.

Native C/C++ libraries provide core functionality for graphics rendering (OpenGL ES), media playback, and database management (SQLite) [59]. Android Runtime (ART) is the application runtime environment used by the Android operating system, replacing the older Dalvik virtual machine [60]. ART is responsible for compiling byte code into native machine code that the device's CPU can execute.

Application framework layer offers APIs for common system services, including window management, content providers, and resource handling. Developers use this layer to build applications in Java or Kotlin. And finally at the top are user-facing applications, each running in its own virtual machine instance to ensure process isolation and security.

1.3.3 Android’s Key Features and Capabilities

Android’s features and capabilities are central to its global success, offering a highly customizable user experience that distinguishes it from competing platforms [61]. The user interface is designed to be flexible, supporting multi-touch gestures, dynamic widgets, and home screen personalization through launchers and themes. This adaptability empowers users to tailor their devices to individual preferences, while also enabling manufacturers to differentiate products through custom skins and branding. Such flexibility, however, can introduce usability inconsistencies across devices, posing challenges for developers who must optimize applications for varied screen sizes, resolutions, and Original Equipment Manufacturers (OEM) customizations.

Multitasking and connectivity are additional pillars of Android’s design. Android’s architecture supports true multitasking through background services, foreground notifications, and split-screen modes, allowing users to perform multiple tasks seamlessly. Connectivity options are equally comprehensive, with support for Wi-Fi, Bluetooth, NFC, and 5G enabling diverse use cases ranging from mobile payments to high-speed data transfers. These capabilities make Android devices versatile tools for

both personal and professional contexts, contributing to their widespread adoption in emerging and developed markets alike.

Google service integration further enhances Android's appeal, offering deep connections with widely used applications such as Gmail, Maps, and the Google Play Store. This integration provides a cohesive ecosystem that simplifies account management, data synchronization, and content delivery. However, reliance on Google services has drawn criticism over privacy concerns and market dominance, leading to the development of alternatives like the Android Open Source Project (AOSP) and efforts by regions such as the EU and China to promote Google-free Android distributions. This dynamic underscores the tension between ecosystem benefits and user autonomy.

Moreover, security features in Android have evolved significantly to address concerns coming from its open nature. The platform employs a layered security model that includes application sandboxing, verified boot, file-based encryption, and a permissions system that grants users fine-grained control over app access to sensitive data. Google Play Protect adds another layer by scanning apps for malware. Despite these measures, Android's openness can make it more vulnerable to fragmentation-driven delays in security updates, underscoring the need for continued improvements in update delivery and OEM cooperation to maintain user trust and platform integrity.

1.3.4 Main Advantages and Challenges

One of Android's most significant advantages lies in its open-source nature, which allows device manufacturers, developers, and even users to freely modify and distribute the operating system [62]. This openness has fostered a vast, diverse hardware ecosystem, ranging from budget smartphones to premium devices and specialized platforms like smart TVs and automotive systems. Consequently, Android has achieved a dominant global market share, especially in emerging markets where cost effective hardware is crucial. Moreover, the open-source model encourages innovation by enabling third parties to contribute and customize the platform according to their unique needs.

Android also excels in terms of flexibility and user customization. Unlike more closed platforms, Android permits users to change launchers, modify UI elements, install third-party apps from outside the Play Store, and personalize virtually every aspect of their device. This flexibility extends to developers as well, who benefit from a rich set of

APIs, a robust development environment (Android Studio), and support for multiple programming languages such as Java, Kotlin, and C++. Google's integration of cloud services like Google Drive, Maps, and Assistant, further enhances functionality, making Android not just a mobile OS, but a hub for interconnected digital services.

From a technological perspective, Android supports a wide array of hardware features and connectivity standards, including 5G, NFC, and biometric sensors. Its multitasking capabilities and scalable UI makes it suitable for various device types, including foldables, tablets, and wearables. Additionally, Google's investment in AI and machine learning has introduced smart features such as voice recognition, predictive text, and real time translation, which elevate user experience and add competitive value to Android powered devices.

Despite its numerous strengths, Android faces persistent challenges foremost among them is fragmentation. Due to the diverse range of manufacturers and devices using Android, software updates are often delayed or inconsistent. Many devices con-tinue running outdated versions of the OS, limiting access to the latest features and leaving users vulnerable to security risks. Although Google has attempted to address this issue through initiatives like Project Treble (a major re-architecting of the Android operating system, introduced with Android 8.0 Oreo, designed to make it easier and faster for manufacturers to update Android devices) and Project Mainline (introduced in Android 10, modularizing OS components so feature and security updates could be delivered through Google Play), the problem persists, largely because OEMs control the update rollout process and prioritize newer devices over older ones.

Security is another critical concern. While Android implements robust security measures, such as app sandboxing, permission controls, and Play Protect, its open nature makes it more susceptible to malware, particularly through sideloaded apps and third-party app stores. The flexibility that allows customization also creates potential vulnerabilities, especially when manufacturers add poorly maintained software layers or fail to issue timely patches. These concerns have led enterprises and some governments to approach Android with caution, favoring more tightly controlled ecosystems for sensitive use cases.

Additionally, the close integration of Google services has sparked debates over data privacy and monopolistic practices. Although users benefit from a unified ecosystem, critics argue that it gives Google undue influence over user data and limits competition.

Regulatory scrutiny in regions such as the European Union has prompted Google to offer alternatives and greater transparency, but these measures are still evolving. Furthermore, the vast diversity of devices means that developers often face compatibility challenges, requiring extra effort to ensure applications run smoothly across different screen sizes, chipsets, and custom ROMs.

1.3.5 Future Trends and Innovations in Android

Android's future development is shaped by rapid technological shifts, with artificial intelligence (AI) integration standing out as a major driver of innovation [63,64]. AI-powered features such as on-device language translation, advanced image processing, and predictive text have already improved the user experience. Google's introduction of custom AI-focused hardware, such as the Tensor chip in Pixel devices, enables faster, more private processing of machine learning tasks without relying on cloud services. As AI capabilities continue to evolve, we can expect even more personalized and context-aware interactions, making smartphones indispensable tools for productivity, accessibility, and entertainment.

Another prominent trend is Android's adaptation to foldable and large-screen de-vices. The market for foldable smartphones and tablets is expanding, prompting Google to enhance Android's support for adaptive layouts, resizable activities, and improved multitasking interfaces. The development of Jetpack Window Manager and other UI libraries is intended to help developers create seamless experiences across form factors. These innovations reflect a broader industry move toward device convergence, where smartphones can transform into tablets or laptops, offering users greater flexibility and new modes of interaction.

Modular system updates represent another key innovation aimed at overcoming Android's long standing fragmentation challenge. Initiatives such as Project Mainline enable Google to deliver updates to core system components directly through the Play Store, bypassing manufacturer delays. This approach promises faster delivery of security patches and bug fixes, improving overall platform stability and user trust. As Google expands the scope of modular updates, the Android ecosystem may achieve a more consistent baseline of security and features across diverse devices, addressing one of its most persistent criticisms.

Environmental and ethical considerations are also influencing Android's trajectory. Growing awareness of electronic waste and sustainability pressures have led Google

and its partners to commit to longer software support cycles and more efficient recycling programs. By extending device lifespans through regular updates and repairability initiatives, the Android ecosystem can reduce its environmental impact. These sustainability goals are increasingly important as regulators and consumers demand greater accountability from technology companies, shaping Android's strategic priorities in the years ahead.

1.4 Novelty and Contributions

This work introduces a comprehensive and empirically validated approach to smart bio- and cultural-guiding systems that integrates artificial intelligence driven personalization, augmented reality-based guidance, and mobile computing within a unified and scalable platform. The novelty of the proposed system lies not merely in the technological integration of these components, but primarily in the systematic and quantitative demonstration of their individual and combined benefits through a rigorously designed evaluation framework.

In contrast to much of the existing literature, where cultural and tourism guiding applications either rely on simple proximity based recommendations or present augmented reality interfaces without controlled empirical comparison, this work provides measurable evidence that both AI-based personalization and AR-assisted navigation deliver tangible performance and usability improvements over established baseline approaches.

The first major contribution is the design and implementation of a neural-network-based personalization mechanism for recommending points of interest in bio- and cultural-tourism contexts. Unlike static or rule-based systems, the proposed model dynamically adapts to user interaction patterns and contextual signals. Its effectiveness is quantitatively validated using standard information-retrieval metrics, demonstrating substantial improvements in recommendation relevance and ranking quality compared to proximity based and rule based baselines. These results confirm that learned personalization models offer a decisive advantage in dynamic tourism scenarios where user preferences evolve over time.

The second contribution is the empirical validation of augmented reality as a functional navigation aid rather than a purely illustrative interface. While prior work often assumes

that AR intuitively improves spatial understanding, this study provides quantitative evidence that AR guidance significantly increases task success rates and reduces task completion times in real-world point-of-interest localization tasks. These findings indicate a meaningful reduction in cognitive load when users translate digital guidance into physical navigation, directly supporting the practical value of AR in mobile guiding systems.

The third contribution is the adoption of a strictly separated dual-methodology evaluation framework, combining synthetic data based technical and algorithmic evaluation with controlled observational usability studies involving real users. This separation ensures scientific validity by preventing inappropriate conflation of system performance metrics with human-centered usability measurements. Through this methodology, the proposed system is shown to achieve efficient resource utilization on commodity Android devices while simultaneously delivering superior user experience outcomes.

Finally, the work demonstrates that computationally demanding AI and AR functionalities can be deployed on mid-range mobile hardware without compromising responsiveness or usability, thereby addressing a critical barrier to real-world adoption. Collectively, these contributions advance the state of the art in intelligent mobile guiding systems by moving beyond conceptual or demonstrative implementations toward evaluation-driven, deployable, and empirically justified solutions for bio and cultural tourism applications.

2 System's Architecture and Design

2.1 Introduction

The success of any digital platform whether a website or a smartphone application, relies heavily on its underlying system architecture and design. System architecture provides the structural framework that defines how components interact, while design focuses on the usability, efficiency, and adaptability of the system. Together, they form the foundation upon which functionality, performance, and user experience are built.

In an era of rapid technological change and growing user expectations, well-designed system architecture is not only a technical necessity but also a strategic advantage. Websites and mobile applications must be scalable, secure, reliable, and intuitive to maintain competitiveness and user satisfaction.

2.2 Key Principles of System Architecture

Modularity

Breaking the system into smaller, independent components makes it easier to develop, test, and maintain [65,66]. For example, websites often separate the front-end (user interface) from the back-end (server and database) to allow independent upgrades and scalability.

Scalability

A good architecture anticipates growth in users, traffic, and data. Cloud-based solutions, load balancing, and micro-services are widely used in both web and mobile systems to handle fluctuating demand without sacrificing performance.

Security

Security is essential for protecting sensitive information, whether it is user data on a mobile banking app or customer profiles in an e-commerce website. Incorporating encryption, secure APIs, and proper authentication protocols at the architectural level ensures a safer user experience.

Interoperability

Websites and smartphone apps often integrate with external systems such as payment gateways, third-party APIs, or enterprise solutions. A flexible architecture that supports interoperability allows seamless data exchange and smoother workflows.

2.3 Website System Architecture

Websites are typically built using a multi-tier architecture [67-69], which separates the system into logical layers:

1. **Presentation Layer (Front-End):** This is the user interface, designed using technologies like HTML, CSS, and JavaScript frameworks (e.g., React, Angular, Vue). It focuses on visual design, usability, and responsiveness across devices.
2. **Application Layer (Back-End):** Handles business logic, user authentication, data processing, and server-side operations. Common technologies include Node.js, Django, Ruby on Rails, or Java Spring Boot.
3. **Data Layer (Database):** Manages structured and unstructured data. Relational databases (MySQL, PostgreSQL) or NoSQL options (MongoDB, Cassandra) are selected based on scalability and performance requirements.
4. **Infrastructure Layer:** Websites often rely on cloud services (AWS, Azure, Google Cloud) for hosting, load balancing, and content delivery networks (CDNs) to ensure high availability and low latency.

2.4 Smartphone App System Architecture

Mobile applications follow different architectural patterns based on platform (iOS, Android, or cross-platform) [70,71]. Typical architectures include:

1. **Client Layer (UI/UX):** Designed using platform-specific tools such as SwiftUI for iOS or Jetpack Compose for Android, or cross-platform frameworks like Flutter and React Native. This layer focuses on intuitive navigation, accessibility, and visual appeal.
2. **Business Logic Layer:** Encapsulates the app's core functionality, ensuring separation between data handling and user interaction. It may involve patterns like MVC (Model-View-Controller), MVVM (Model-View-ViewModel), or Clean Architecture.
3. **Data Layer:** Mobile apps often require offline functionality, so local storage (SQLite, Room, Core Data) complements server-side databases. Synchronization between local and cloud data is managed through APIs and background services.
4. **Integration Layer:** Smartphone apps rely heavily on APIs to interact with external services, sensors (GPS, camera), and third-party libraries. A well-structured architecture ensures smooth integration without compromising security.

2.5 Design Considerations

While architecture defines structure, design ensures usability and efficiency [72]. Key design considerations for both websites and smartphone apps include:

- **User-Centered Design (UCD):** Interfaces should prioritize user needs, offering intuitive navigation and clear visual hierarchies.
- **Performance Optimization:** Fast loading times, efficient caching, and minimal resource consumption improve the user experience.
- **Accessibility:** Systems should be inclusive, following guidelines such as WCAG for websites and platform-specific accessibility standards for mobile.
- **Consistency:** A consistent design language across platforms ensures familiarity and trustworthiness.
- **Maintainability:** Well-documented code, modular design, and adherence to standards allow easier updates and feature expansions.

The architecture and design of websites and smartphone apps are inseparable elements that directly influence functionality, scalability, and user satisfaction. While websites demand robust multi-tier structures and global availability, mobile apps require efficient local data management and integration with device capabilities. Both must prioritize security, interoperability, and responsive design. Ultimately, a carefully planned system architecture paired with thoughtful design ensures that digital platforms remain competitive, adaptable, and valuable to users in an ever-evolving technological environment.

2.6 Personal Tour Guide Application's and Web Platform's Architecture

The platform consists of three parts as shown in Figure 2.1. Three basic packages of which the first one called External Services will include all necessary source code and APIs to communicate with remote services. The second one called Libraries will include all important libraries which are crucial for our application. The third main part of the hierarchy is the android application itself.

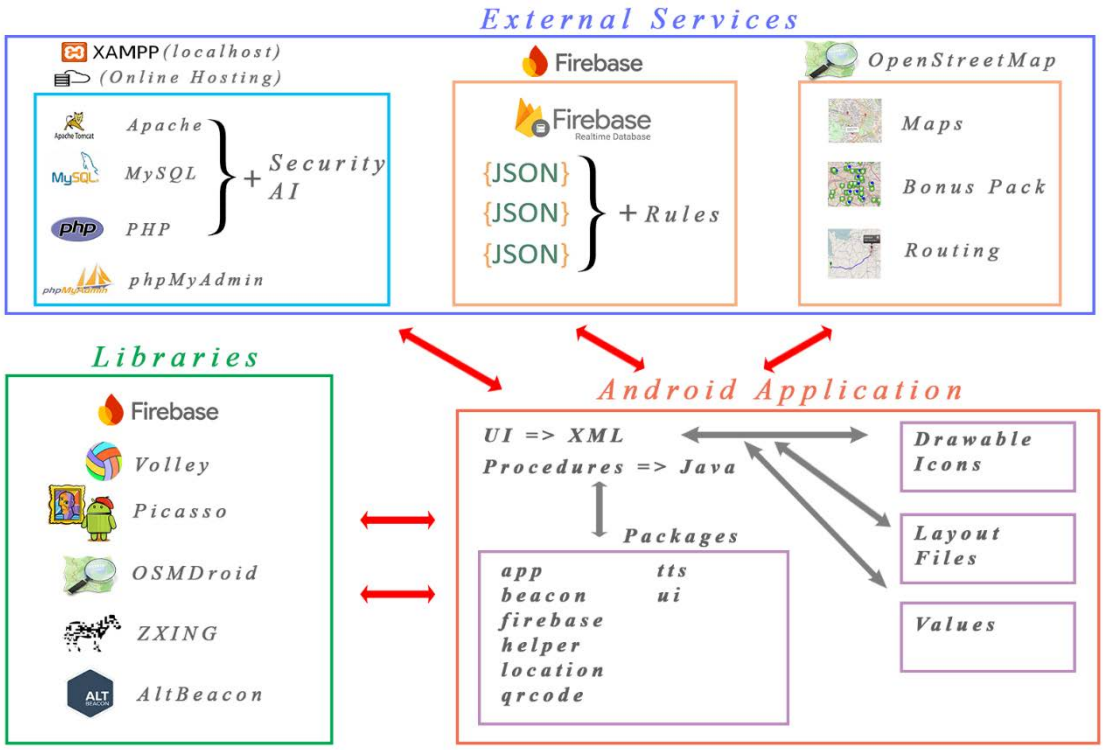


Figure 2.1: Personal Tour Guide Platform's Architecture

2.6.1 External Services

In details, External Services hold the key to functionality and interoperability. First, we choose to use a local host environment using XAMPP [73] (for testing and development purposes) implementing source code and functionalities with full compatibility to online hosting services provided currently by shared or dedicated hosting companies on these days. The local or online hosting service (containing Apache web server [74], MySQL database [75], PHP [76] installation and phpMyAdmin [77]) will provide the necessary tools to host the main App's Users tables to store all information that deals with the users' accounts and their preferences. It will also be able to host the necessary RESTful API [78] implemented in PHP language for Android applications to be able to communicate safely with the MySQL database.

Secondly, Firebase Realtime Database [79], is chosen as our main operational functioning database. Firebase is considered to be the main, most preferred and most commonly used database from Google LLC, Android OS and generally Android applications. In this case, Firebase will be able to store all available data that users can exploit using our application and security issues are solved using the according rules.

Thirdly, since many users may not be indigenous we have to provide a digital map service. This is why we choose to use Open Street Map through its Android's "sibling", the osmdroid extension. Also, because of the fact that users may also request a path from their current position to some destination we choose to use the osmbonuspack, which is a library to interact with OpenStreetMap data inside an Android application and moreover to perform real time on demand routing.

2.6.2 Android Libraries

Android libraries [80,81], are collections of pre-written code, resources, and tools that developers can use to speed up the app development process. Instead of building features from scratch, developers can integrate libraries to handle common tasks such as networking, image loading, data storage, or UI design. For example, libraries like Retrofit simplify API calls, Glide or Picasso make image loading efficient, and Room provides a robust database management solution. These libraries not only save time but also ensure reliability, since they are often maintained by large developer communities and thoroughly tested in real world projects.

Key benefit of using Android libraries is the ability to keep projects modular and maintainable. By leveraging these ready-made solutions, developers can focus more on building unique features instead of reinventing the wheel. Libraries also help with ensuring compatibility across different Android versions and devices, which can be a challenge in the fragmented Android ecosystem. Additionally, many libraries are open-source, giving developers the flexibility to customize them according to their project needs. This collaborative ecosystem makes Android development more efficient, innovative, and accessible for developers of all levels.

The necessary libraries to achieve our goals and the desired results are described in the previous figure in section Android Libraries. In order our application to be able to perform correctly its functionalities we need the according external libraries. These libraries are the Firebase extension which will provide the ability to connect to the Firebase Realtime Database, the Volley library in order to perform http requests to online servers, the Picasso library to load image from online hosting providers, the Osmdroid library to load maps and perform routing, the ZXING library is the most famous extension for usage in autonomous QR Code readers and last the AltBeacon library which is able to provide available algorithms for Android’s Bluetooth Detectors.

2.6.3 Android Application

The goal of the Android application is to offer users a complete digital tour guiding experience as long as user is on sight using mostly voice as the preferred mean of communication. The preferred language for graphic design in Android is XML and for computational logic we prefer Java instead of Kotlin. The hierarchy makes use of packages to divide on conquer and in our implementation we are trying to keep things simple and stupid in order to achieve agile approach. This means that we are using self-sustained classes and parameterized variables to build components. Application contains eight main packages operating as the back bone of the system and using the according services users will be able to be guided to points of interest using only their mobile phone without the need or extra “expensive” cost of the physical presence of a guide being necessary.

2.7 Personal Tour Guide Application’s Design

Application design and mockups play a crucial role in the development process because they serve as the blueprint for the final product. Before a single line of code is written,

designers and developers collaborate to visualize how the application will look and function. The design phase focuses on defining the layout, color schemes, typography, icons, and overall user experience (UX). It ensures that the application not only looks appealing but also provides intuitive navigation, making it easy for users to accomplish tasks. A well-structured design reduces confusion and enhances usability, which directly impacts user satisfaction and retention.

Mockups are detailed static representations of an app’s interface. Unlike rough sketches or wireframes, mockups include realistic design elements such as colors, fonts, and imagery, giving stakeholders a clear preview of the end product. They help developers, clients, and team members align on the visual direction and functionality before moving into coding. Mockups can also be used to test design concepts, gather feedback, and make adjustments early, saving time and resources in the long run. Tools like Figma, Sketch, and Adobe XD made creating mockups faster and more collaborative, allowing multiple team members to work simultaneously on a design.

Beyond visualization, mockups play a vital role in communication. They act as a bridge between designers and developers, ensuring that ideas are translated into functional features accurately. They also provide a foundation for prototyping, where interactivity is added to simulate user flows and actions. This step is especially useful for identifying usability issues early. In short, application design and mockups are not just about aesthetics. They are essential for aligning vision, reducing risks, and ensuring that the final product meets user needs effectively.

Login and registration windows are essential components in many Android applications, as they provide users secured access and personalized experiences. The login window typically includes input fields for a username, email, or phone number along with a password field. Additional features such as "Forgot Password," "Remember Me," and support for biometric authentication (like fingerprint or face recognition) are often included to enhance usability and security. A well-designed login blueprint ensures that users can access the app quickly and safely while minimizing friction. Clear error messages, input validation, and user-friendly layouts are also crucial to prevent frustration and improve the overall experience.

The registration window, on the other hand, focuses on creating new user accounts by collecting necessary information such as name, email, password, and sometimes phone numbers or profile details. Modern registration blueprints often include password

strength indicators, confirmation fields, and optional integrations with social media or Google accounts for quicker sign-ups. The design should remain simple and uncluttered, guiding users step by step through the process without overwhelming them. Security measures, such as email verification or CAPTCHA, are commonly integrated to prevent spam and unauthorized access. Together, login and registration window blueprints form the foundation of user authentication, balancing security with convenience to create a smooth onboarding experience.

The proposed design for our application's Login and Registration windows is shown in the Figure 2.2. Login and Registration windows are equipped with EditText fields which are able to retrieve given inputs for inner processing and an activation button which triggers communication with remote MySQL.

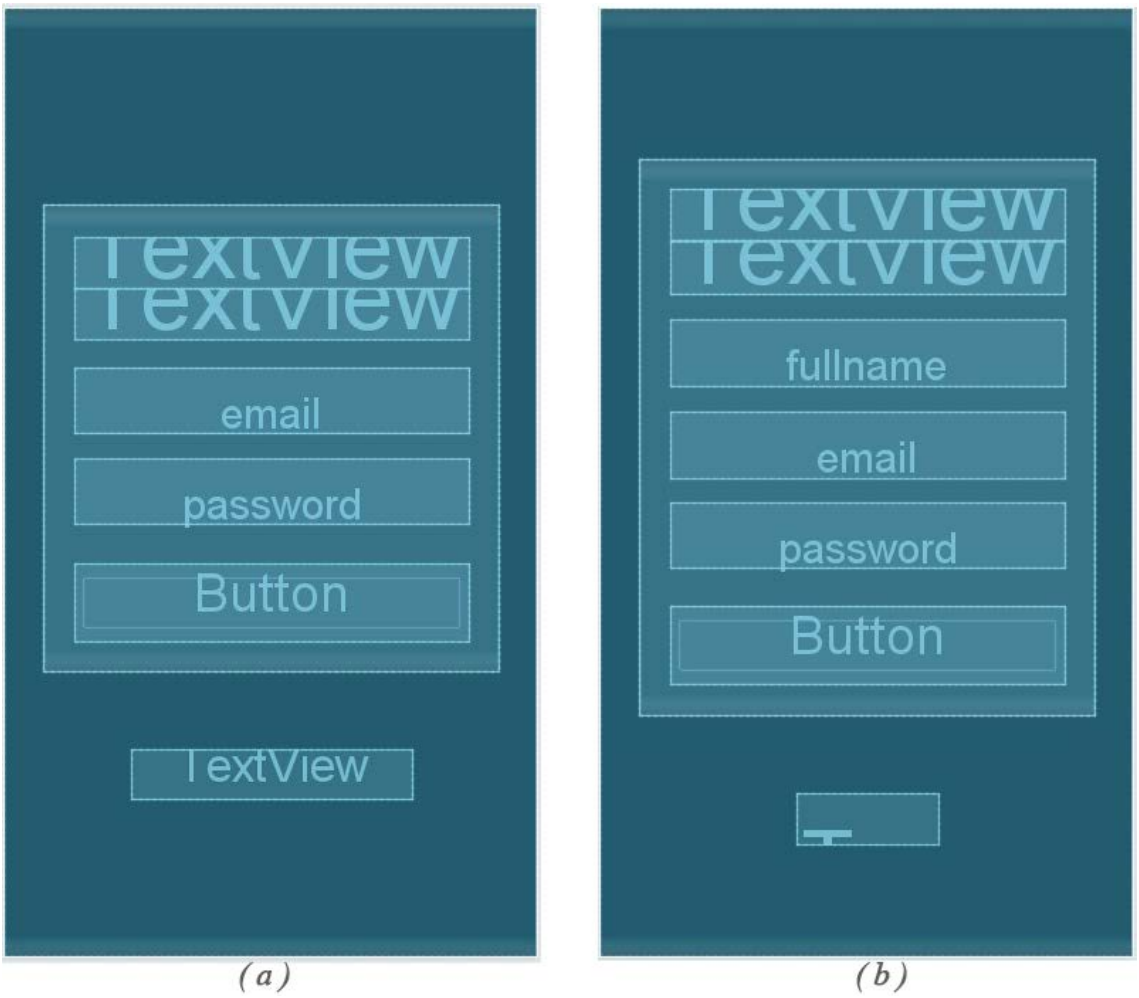


Figure 2.2: (a) Login Window and (b) Registration Window mockups

Moreover, the design for application's Main and Guide window is presented in the Figure 2.3. In the Main window a GridLayout is used to divide space into equal parts.

Each cell contains a CardView with an ImageView and a TextView beneath. Each CardView will be responsible for one category from the available guides in our system. On the other hand, the Guide window provides all available information for each guide to users. The design starts with an ImageView (the main image for the selected guide) and then provides information lite the title, intro, ratings and guidance itself on the left side. Next to the title and the intro sections, on the right side, we can host the two necessary options which are the favorite and location icons. Each of these windows will also provide a basic toolbar on top of the design (included from an external file), holding main features and operations.

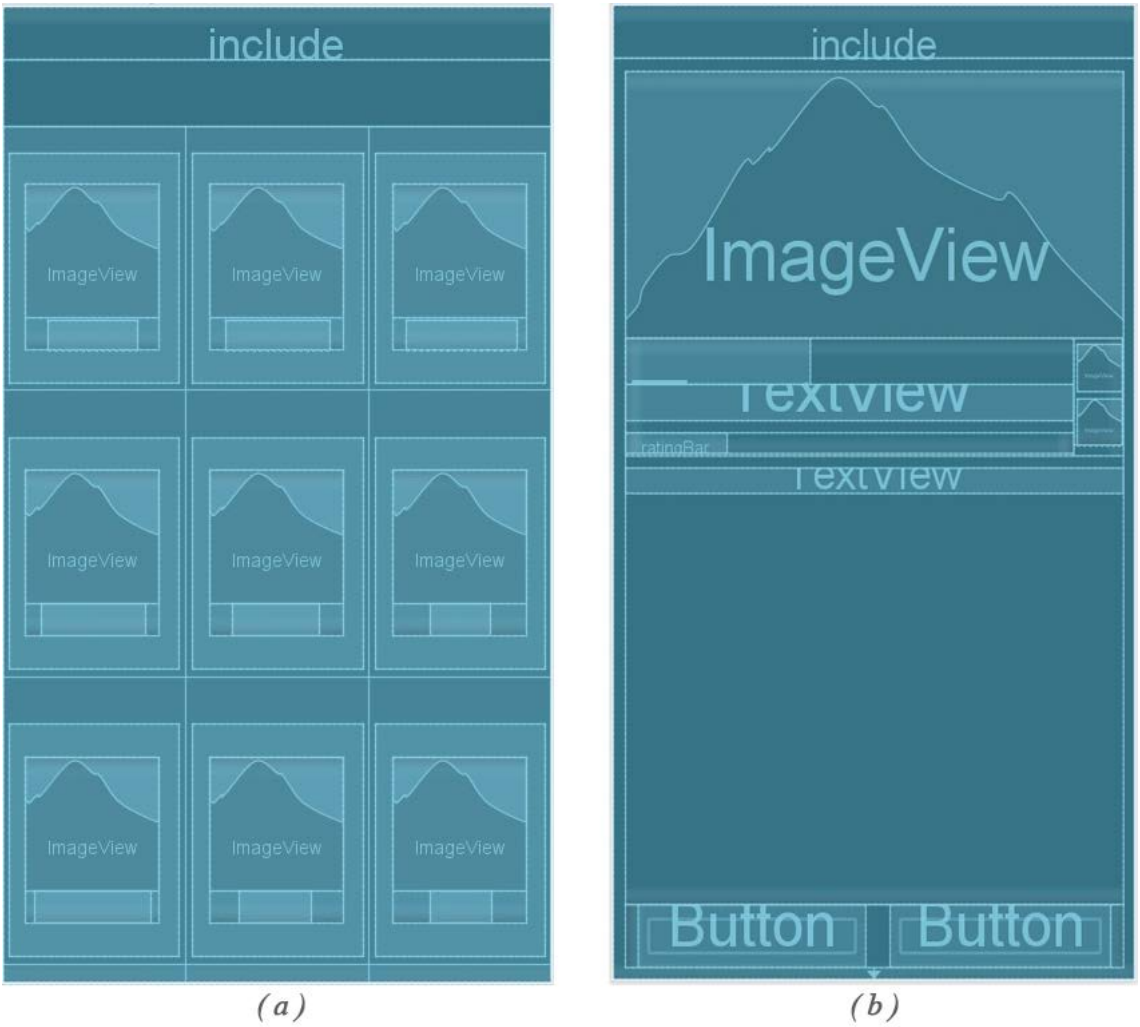


Figure 2.3: (a) Main Window and (b) Guide Window mockups

2.8 Firebase UML Diagram

Firebase databases, whether it’s the Realtime Database or Cloud Firestore, are NoSQL systems that follow a document-oriented model. Instead of tables, data is organized into collections of documents, with each document structured like a JSON file. These documents can also include nested collections or other documents.

In theory, every document within a collection can differ entirely from the others. Because of this, using a UML class diagram wouldn’t be appropriate, as classes are based on shared attributes and behaviors.

In practice, typically, the documents within a collection consist of objects that are quite similar or at least related. Some objects may have a few extra or fewer properties, but overall, they represent the same type of entities.

In this situation, UML class diagrams can be effectively applied, since UML is class-oriented rather than table-oriented. In this case we must pay attention to the followings:

- A class represents a specific type of entity. In reverse engineering, you would typically begin with the known collections.
- Minor structural variations between documents of the same type can be shown in the UML diagram as optional properties (e.g., multiplicity [0..1], like lastLogin in the users example).
- Larger differences between documents that still share common traits are usually modeled with specialized subclasses.
- The main challenge lies in modeling embedded documents and sub-collections, which often correspond to additional or composite classes.

In principle, UML class diagrams are not meant solely to represent the database schema (data only). They are intended to design the object model of our application (data and behavior) independently of the database. This object model can then guide the database design according to the types of objects required to meet your users’ needs.

In our case, as shown in Figure 2.4, Firebase Realtime Database will include six types of Classes with each one containing the entries for its field of interest. These six types are Nature, Aquariums, Religion, Museums, Cities and Ruins. Each one of these classes will contain a list of instances of the Guide Class. The Guide Class will contain all necessary information about a tour guide such as the unique auto generated identifier of

each entry “id”, its title “title”, an introduction text “intro”, a URL path to an image hosted online “image”, the text content of the guidance itself “quidance”, a unique self-generated key to be used in case of a e.g. qr code “key”, the language of the presentation “language”, the latitude and the longitude of the place in maps “latitude and longitude”, the users’ average rating value “rating”, a URL path to the webpage of the selected entry “link” and last but not least the uuid of the Bluetooth beacon device that may exist in the area “uuid”. All values for the entries above will be String type to simplify communication from Android device to Firebase Realtime Database.

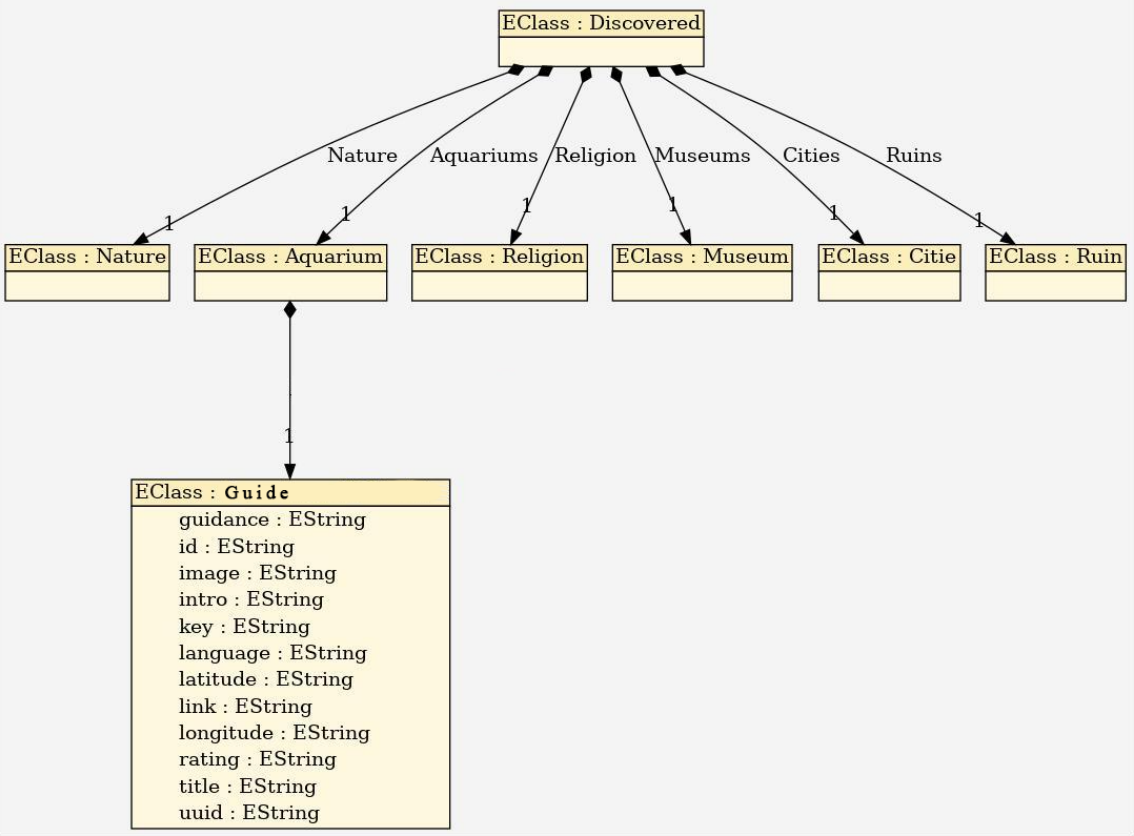


Figure 2.4: Firebase JSON UML Diagram

2.9 Websites and Applications Security

In today’s digital age, websites and applications have become central to business operations, communication, and personal interactions. However, this heavy reliance on digital platforms has also made them prime targets for cyber threats. Website and application security refers to the collection of practices, tools, and processes designed to

protect these systems from unauthorized access, misuse, and attacks that can compromise data and functionality.

2.9.1 Common Security Threats

- **SQL Injection:** Attackers manipulate input fields to access or corrupt databases.
- **Cross-Site Scripting (XSS):** Malicious scripts are injected into web pages to steal user data or hijack sessions.
- **Cross-Site Request Forgery (CSRF):** Exploits user authentication to trick them into performing unintended actions.
- **Malware and Ransomware:** Harmful software is introduced to disrupt services or extort money.
- **Brute Force Attacks:** Repeated attempts are made to guess user credentials and gain unauthorized access.

2.9.2 Security Best Practices

- **Strong Authentication and Authorization:** Use multi-factor authentication (MFA), enforce strong password policies, and apply role-based access control to limit user privileges.
- **Input Validation and Sanitization:** Prevent malicious code injection by validating and sanitizing user inputs.
- **Encryption:** Protect sensitive data in transit (using HTTPS/SSL) and at rest through encryption methods.
- **Regular Updates and Patch Management:** Keep servers, libraries, frameworks, and applications updated to close known vulnerabilities.
- **Secure APIs:** Since websites and apps often rely on APIs, they should be protected with tokens, rate limiting, and encryption.
- **Monitoring and Incident Response:** Continuous monitoring and well-defined response plans help detect and mitigate attacks quickly.

2.9.3 Importance of Security

Maintaining security in websites and applications is essential for:

- **Protecting User Data:** Safeguarding sensitive information such as personal details, payment credentials, and health records.
- **Building Trust:** Users are more likely to engage with platforms they perceive as secure.
- **Legal and Regulatory Compliance:** Adhering to data protection laws like GDPR, HIPAA, or PCI DSS to avoid penalties.
- **Business Continuity:** Preventing downtime and financial loss from cyber-attacks.

Websites and applications serve as gateways to critical data and services, making them prime targets for cyber threats. A comprehensive security strategy that includes preventive measures, continuous monitoring, and responsive action is vital to protect digital assets. By prioritizing security, organizations not only defend against evolving threats but also foster user trust, compliance, and long-term success.

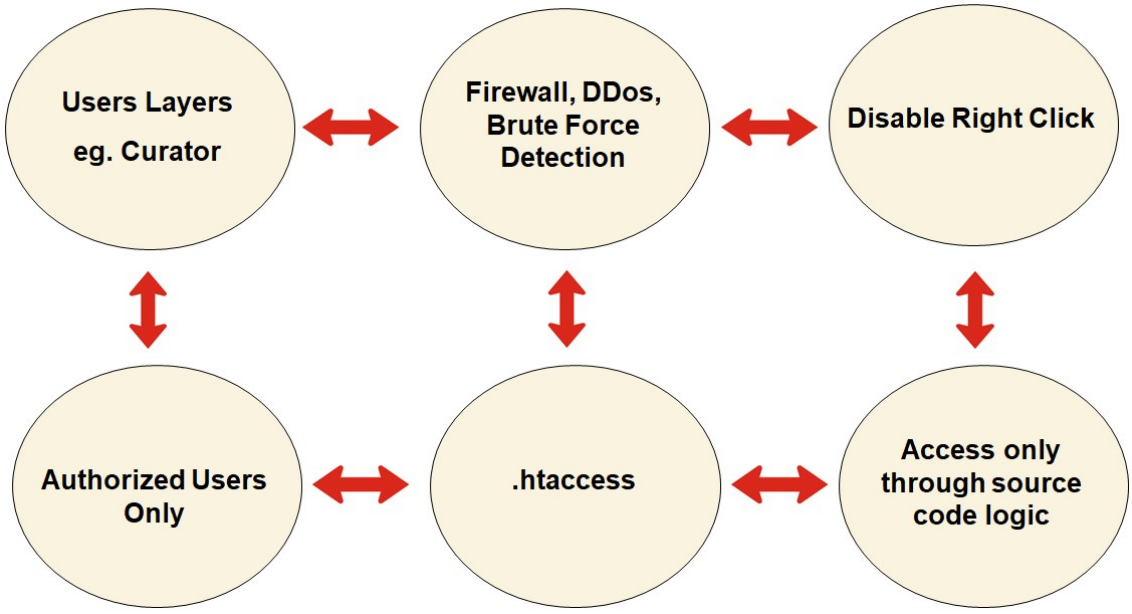


Figure 2.5: Online Security Package Implementation

As Figure 2.5 shows the online platform is secured using the presented six points. These addressed points can be easily implemented in a WordPress CMS. The platform is able to distinguish users and keep them in different layer with different permissions. Plugins which are able to detect and prevent DDos and Brute Force attacks are installed on the

server side. Simple techniques such as disabling right click or showing data only to authorized users can help and prevent copying information. Moreover, files accessing permission can be edited using the .htaccess files. Last, all communications with the Android devices is secured using safe and encrypted SSL when transmitting and receiving data and all sensitive information stored in database uses MD5 encryption.

2.10 UML Class Diagram

A UML (Unified Modeling Language) [82-84] class diagram [85] is one of the most widely used diagrams in software engineering for representing the static structure of a system. It provides a blueprint of the system by visually describing its classes, attributes, operations (methods), and the relationships between objects. Class diagrams help developers, designers, and stakeholders understand the architecture of a system at a conceptual, logical, or implementation level.

2.10.1 Components of a Class Diagram

A typical class diagram consists of the following elements:

- **Classes:** Represented as rectangles divided into three compartments – the top contains the class name, the middle lists attributes (variables or properties), and the bottom lists operations (methods).
- **Attributes:** Define the characteristics or properties of a class. For example, in a User class, attributes might include username, email, or password.
- **Operations (Methods):** Define the behavior or functions of the class, such as login(), register(), or updateProfile().
- **Visibility:** Attributes and operations can have visibility indicators:
    - + Public (accessible from anywhere)
    - - Private (accessible only within the class)
    - # Protected (accessible by the class and its subclasses)

2.10.2 Relationships in Class Diagrams

Class diagrams also capture how classes interact and depend on each other through different types of relationships:

- **Association:** A basic link between classes (e.g., a Customer places an Order).

- **Multiplicity:** Defines how many objects of one class can be associated with another (e.g., one Customer can have many Orders).
- **Inheritance (Generalization):** Shows a parent-child relationship where a subclass inherits attributes and behaviors from a superclass (e.g., Car and Bike inherit from Vehicle).
- **Aggregation and Composition:** Special types of associations. Aggregation represents a "whole-part" relationship where parts can exist independently (e.g., a Library has Books), while composition represents stronger ownership where parts cannot exist without the whole (e.g., a House contains Rooms).
- **Dependency:** Indicates that one class uses or depends on another temporarily (e.g., a PaymentService depends on a CreditCard class).

2.10.3 Importance of Class Diagrams

Class diagrams are essential during the design phase of system development. They help in:

- Visualizing system architecture before implementation.
- Identifying reusable components and reducing redundancy.
- Supporting communication between technical and non-technical stakeholders.
- Serving as documentation for maintenance and future enhancements.

In summary, a UML class diagram is a powerful tool for modeling the structure of object-oriented systems. By defining classes, attributes, operations, and relationships, it creates a clear representation of how different parts of a system interact. This clarity supports efficient development, reduces complexity, and ensures scalability of both websites and mobile applications.

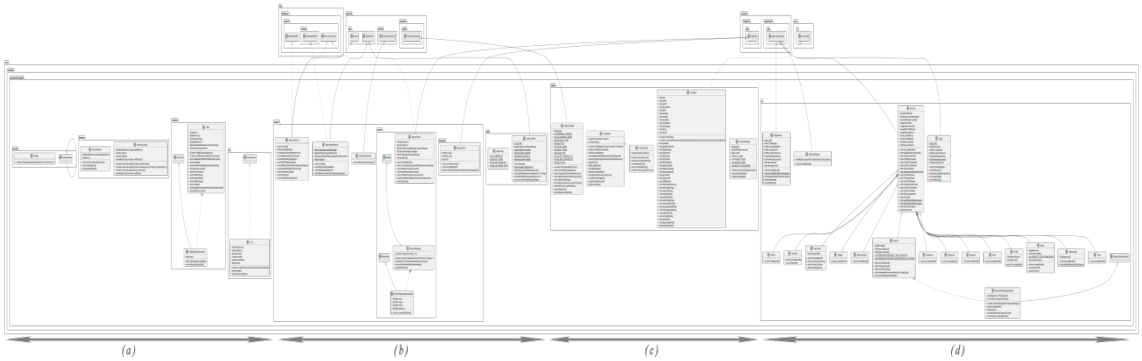


Figure 2.6: Personal Tour Guide Application Class Diagram

As above class diagram shows in Figure 2.6, we try to use the technique of divide and conquer separating classes in packages. Analyzing class diagram figure from left to right, the package qrcode will hold the classes responsible for reading and decoding image with qr codes printed on them. The firebase package contains classes responsible for communicating with the online Firebase Realtime Database and synchronizing when it is necessary. The location package contains classes responsible for communication with the online repository of the open street map service and finding routing paths. The tts package holds classes responsible for the embedded text to speech service provided by the Android OS. The beacon package provides algorithms able to support Bluetooth Beacons protocols. The app package contains classes responsible for the correct application functionality and setup. The helper package contains classes crucial for apps operation and provides the necessary data templates. Last, the ui package contains all necessary classes that provide basic user interface functionalities.

Because of the size and the complexity of the class diagram we have separated its figure in four different pieces named with the letters a, b, c and d. These figures are shown individually in the Appendix 1 Class Diagram section.

2.11 UML Sequence Diagram

A UML (Unified Modeling Language) sequence diagram [86] is a type of interaction diagram that models the dynamic behavior of a system by showing how objects communicate with each other over time. Unlike class diagrams, which focus on static structure, sequence diagrams emphasize the flow of messages between system components, making them valuable for visualizing system processes and use-case scenarios.

2.11.1 Components of a Sequence Diagram

Sequence diagrams use several elements to illustrate object interactions:

- **Objects (Participants):** Represented at the top of the diagram as boxes, each object is typically named according to its role (e.g., User, System, Database).
- **Lifelines:** Dashed vertical lines extending downward from each object, indicating the existence of that object over time.
- **Activation Bars:** Narrow rectangles placed on a lifeline to show when an object is active or performing a process.
- **Messages:** Horizontal arrows exchanged between objects that represent communication. Messages can be synchronous (waiting for a response), asynchronous (not waiting), or return messages.
- **Loops and Conditions:** Represented using frames, they indicate repeated interactions (loops) or alternative flows (conditions).

2.11.2 Purpose of Sequence Diagrams

Sequence diagrams are widely used in software engineering for:

- **Modeling use-case scenarios:** They show how a particular functionality, such as user login or order processing, unfolds step by step.
- **Clarifying system behavior:** Developers and stakeholders can visualize how objects interact in real-time, reducing ambiguity in requirements.
- **Design validation:** By mapping out processes, designers can verify that the system logic matches user needs before implementation.
- **Communication:** Sequence diagrams serve as a clear communication tool between business analysts, developers, and testers.

In summary, UML sequence diagrams are powerful tools for modeling time-ordered object interactions within a system. They not only help designers and developers understand the step-by-step flow of processes but also serve as valuable documentation throughout the software development lifecycle. By focusing on communication between components, sequence diagrams bridge the gap between requirements and technical implementation.

2.11.3 Main Window and Tour Guide Scenarios using Sequence Diagrams

The following two figures (Figure 2.7 and Figure 2.8) show the sequence diagrams for the Main Activity and the Tour Guide activity. For instance, in a Main Activity sequence:

- The onCreate event calls BasicUI class.
- BasicUI class initializes all necessary tools of the app.
- If user is not Logged In then activity returns.
- Users can call FavoriteTours or Firebase RecordsLoader on demand.

This flow, represented visually, helps ensure that all required interactions are considered and correctly implemented.

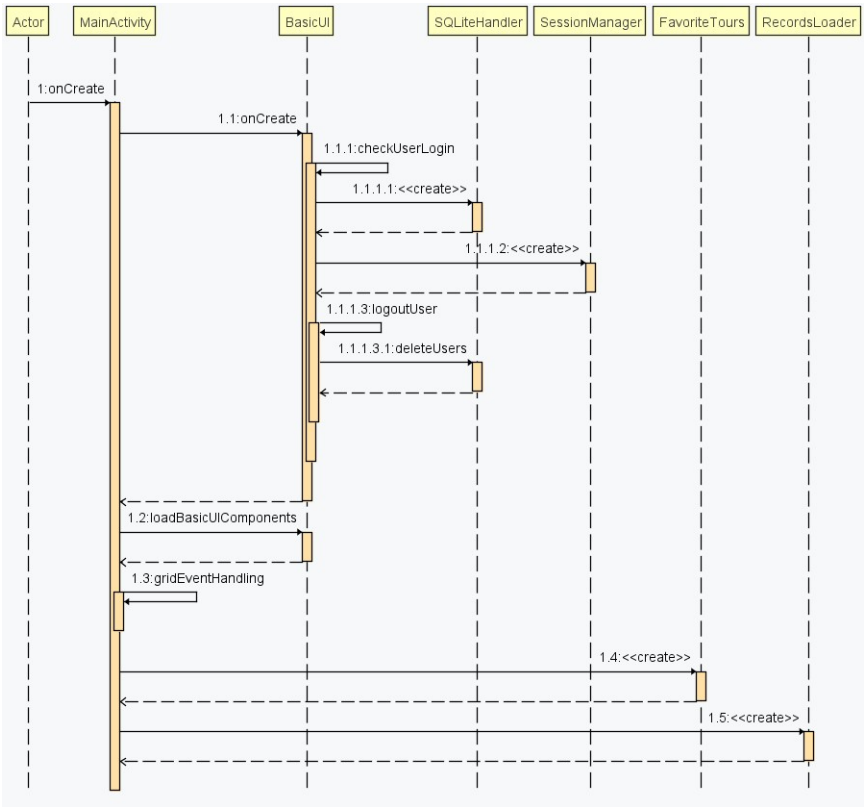


Figure 2.7: Main Window Sequence Diagram

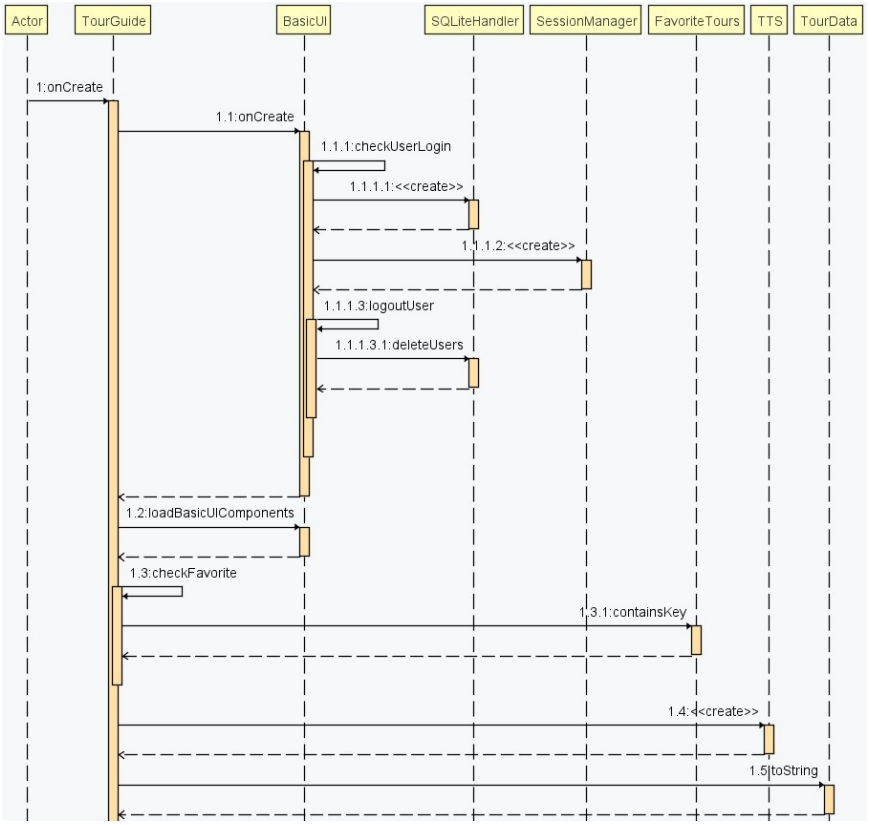


Figure 2.8: Tour Guide Window Sequence Diagram

3 Artificial Intelligence Neural Networks Algorithms in the Field of User Detection and Users' Preferences Prediction

In this thesis we study, develop and evaluate the use of Artificial Intelligence algorithms to identify the identity of the user's account as well as find and present possible preferences options analyzing data that can be stored by web pages or applications operated by users. User detection algorithms can be considered crucial, since there are many uses for targeted advertising and product recommendations in electronic commerce and product recommendations fields. Using single-layer or multi-layer Artificial Intelligence Neural Networks implementations, we study the use of algorithms that are implemented for example for classification, regression, summarization, association, clustering, applying them to data related to the personal application accounts of different users or to the choices they make during the use of the functions or services provided. Our study shows that we can achieve high rates of users' identity detection as well as high rates of preferences prediction.

3.1 Introduction

In the modern digital era, the ability to identify the user's identity on websites and applications as well as their preferences is particularly important, since thanks to the targeted promotion of products or services, we can see a significant increase in sales and revenue for companies. We have seen mechanisms applied for targeted promotion to specific users in many areas such as search engine marketing, social media targeting, television, mobile devices, content and contextual targeting, technical targeting and time targeting. Given the significant increase of information today, thanks to the development of information technology, systems use more approaches and technologies that are able to collect more and more data about the user's movements or choices [87]. The result of this is the fact that websites and the corporations behind them are able to collect significant amounts of personal data about each user by using mechanisms such as User Registration or monitoring movements at specific IP addresses [88, 89], etc.

Using common techniques such as storing the user's IP address on the server can lead to incorrect results in the mechanisms for detecting or locating the user in our time by websites or applications, as for example many users now use technologies such as VPN (Virtual Private Network) to hide their real address on the internet [90-93]. IP targeted identification will be less accurate when the case scenario comes to the large scale problem with multiple users sharing the same VPN provider. In this case, the server's detection algorithms could not be able to distinguish IP addresses from different users. Therefore, identifying web users based solely on current standard identification mechanisms rather than using more sophisticated tracking techniques becomes a really difficult problem.

In order to prevent single channel authentication systems from being exploited, the Federal Financial Institutions Examination Council (FFIEC) promotes the use of multiple channels in authentication procedures [94-97]. If properly developed, behavioral pattern-based user identification systems could offer one more channel in a multi-channel strategy. The rationale is that relying on a single authentication channel, such as a password or biometric measure, may expose systems to security vulnerabilities. To address this, this thesis explores the use of behavioral pattern-based user identification as an additional, complementary authentication mechanism in a multi-channel framework.

This approach involves analyzing and modeling users' behavior patterns, particu-larly in the context of web browsing activity, to create unique users' sample profiles. These behavioral profiles reflect repeated actions or habits, such as consistently visiting specific sections or options on websites, choosing the same options in smartphone applications, or spending a similar amount of time on specific activities during in app or in site navigation. Such behaviors are described as behavioral patterns, and their recurrence or consistency is referred to as the strength of the pattern. The more consistently a behavior is observed across sessions, the stronger it is considered. Techniques such as frequent pattern mining algorithms are used to identify these behaviors.

User identification can either be the main objective or a step toward improving personalization and targeting. Even if a user isn't logged in, recognizing them can enable more tailored recommendations. On large-scale websites, relying only on a single identification method becomes impractical, so combining behavioral data with other tracking technologies can help boost effectiveness.

There are also possible client-side uses that can gain advantages from profiling data focused on users. For example, affiliated websites can exchange user data, enabling them to collaborate on product recommendations more effectively. Additionally, users might choose to install software from a trusted third party that monitors client-side behavior to create user identification models. These models could then support services like behavioral authentication on the user's behalf.

Although previous research has focused on the use of well-known techniques such as cookies and IP addresses to achieve user identification and recognition [98], in this article we will study different approaches that will involve the use of history and users' actions recorded in the system. Also, although our research does not include a specific section that deals with the utilization of our algorithms in the field of biometric data such as fingerprint, iris recognition or facial recognition, we have no reason to believe that similar algorithms to those proposed in this work cannot be used in those fields.

The contribution of this thesis in the field and computer science arises from the fact that by using Artificial Intelligence Neural Network algorithms and Machine Learning mechanisms, the online platform - website or mobile device application can recognize possible indications of user identification without the user having even been authenticated by the basic user authentication mechanism of the service such as login or

the existence of an available pre-installed cookie. By observing the user's behavior, the platform can be able to perform a probabilistic identification of his identity as well as a prediction of his choices and preferences before he is even officially authenticated by the service.

Using Neural Network algorithms, we will try a variety of approaches and study different code examples based on the Python programming language using different source code libraries that support free access and non-profit use. These examples will be implemented in a different way and approach but will have a common goal. Initially, the recognition and possible detection of the user's identity as well as his categorization before he is even authenticated by the platform thanks to the real time monitoring of his movements and his choices. And then the automatic prediction of possible new options of interest and the extraction of these options in a descending order starting from the most likely option of interest of the user and settling on the least likely option of interest.

It is worth mentioning that the automatic mechanism for finding possible future preferences and points of interest of the user, which mechanism is able to suggest and indicate new options to the user that are relevant to his interests, is able to work in real time for a currently unknown user of the platform but also for an already certified person. In the case where the person is certified and already known on the platform, it means that there is significantly greater and more valuable amount of data for him with a increased penetrating character, thanks to his significantly longer history, which data can help us to significantly increase the degree of successful and quick prediction of his possible future preferences.

3.2 Related Work

3.2.1 The Data Mining Procedure

In the field of data mining, user profiling is commonly explored for purposes such as personalization and fraud detection. Meanwhile, in information retrieval, user profiles are primarily used to tailor information retrieval or filtering to individual users. A major challenge in creating personalized applications is developing accurate and detailed user profiles. These profiles typically consist of various attributes, as well as sets of conjunctive or classification rules. Beyond individual profiling, research has also focused on creating aggregated user profiles, which represent groups of users with

shared interests [99]. Clustering techniques are used to form these group profiles, which are then assessed within recommendation systems as part of broader personalization strategies aimed at identifying user preferences for certain pages or content.

Other approaches to user profiling rely on web usage data without involving personalization. These methods create user profiles by analyzing various web log files from a website. In such cases, it's assumed that the website owner categorizes each page under predefined topics. A user's profile is then built based on the amount of time they spend on pages within each topic category [100]. Similarly, there are efforts to develop profiles using server logs from career websites, where a user's profile is constructed based on how many times they click on job listings [101].

The utilization of behavioral profiles is fundamental to effectively detect fraud, particularly due to the impracticality of processing and evaluating all related transaction records in real time to identify anomalous activity. In both fraud and intrusion detection systems, profiles are leveraged through two primary methodologies: signature-based detection and anomaly-based detection [102].

In signature-based detection, a repository of predefined malicious behavior patterns, referred to as signatures, is maintained within a dedicated signature database. Incoming user traffic is profiled and systematically compared against this repository to identify matches indicative of known fraudulent or malicious activities.

Conversely, anomaly-based detection constructs a behavioral baseline model for each user or entity, derived from historical activity data. This individualized profile serves as the reference point for monitoring real-time behavior. Any significant statistical deviation from this established baseline may suggest an abnormality, such as potential account takeover or misuse [103]. The core assumption is that legitimate behavior exhibits consistent patterns, and deviations beyond a defined threshold can be flagged for further inspection. To achieve this, user profiles are created based on a series of web sessions linked to identifiable user IDs. These profiles are then utilized to infer the identity of users' in future anonymous web sessions, enabling user identification [104]. Specifically, it is considered a web browsing dataset that includes N distinct users $U=\{u1,u2,\ldots,uN\}$. It is chosen time T to represent the current moment. Prior to T, user ui has completed web browsing sessions $S(ui)=\{ei\ 1,ei\ 2,\ldots,ei\ x\}$. Sessions are arranged chronologically. The task of constructing user profiles for identification involves

learning user behaviors from sessions recorded before a specific time T, in order to recognize users in anonymous sessions that occur after T.

3.2.2 Users' Identification Techniques

User identification in websites and mobile applications refers to the process of recognizing and verifying users to provide secure, personalized, and consistent experiences. This is typically achieved through login credentials such as usernames, email addresses, and passwords, often enhanced by multi-factor authentication or social login methods like Google or Facebook. On websites, identification is commonly managed using sessions and cookies that store unique user IDs, while mobile apps rely on secure tokens, device IDs, or biometric data like fingerprints or facial recognition. In addition, analytics tools and device fingerprinting may be used to track users' behavior even without explicit login, helping improve services and security. Regardless of the method, user identification must comply with privacy laws like GDPR and CCPA to protect user data and ensure transparency.

In more details, user identification in websites and mobile applications is a fundamental component of digital security and personalized user experience. It involves the ability of a system to recognize who the user is, verify their identity, and tailor interactions based on their profile, history, or preferences. The most common method of identification is through user authentication, typically using a combination of a username or email address and a password. Many platforms enhance this with multi-factor authentication (MFA), such as one-time passwords (OTPs), authenticator apps, or biometric verification like facial recognition or fingerprint scans. Third party verification services like Social login options, using common companies services like Google, Facebook, or Apple, are also widely implemented for faster and more secure identification although this mechanism is also often avoided because it adds an extra external third party dependency to websites or applications which cannot always be fully managed and also may also add an extra operational cost to the necessary procedures that an only service has to provide.

On websites, once a user is authenticated, their session is often tracked using HTTP cookies or session tokens, which allow the server to maintain state between page loads. Modern web applications may also use JWT (JSON Web Tokens) stored in local storage to manage identity in single-page applications (SPAs). In contrast, mobile

applications often use secure, persistent tokens that are stored in encrypted form within the app's storage or device keychain. Additionally, mobile platforms may rely on unique device identifiers, such as UUIDs or platform-specific advertising IDs (e.g., Apple's Identifier for Advertisers (IDFA) for iOS, Google Advertising ID (GAID) for Android), for identifying users across sessions.

Even without explicit login, users can be identified or tracked through techniques like browser fingerprinting, which collects various data points (such as device type, screen resolution, installed fonts, and browser plugins) to create a unique profile. Simi-larly, analytics and advertising tools assign anonymous user IDs to track behavior and performance across sessions. These identification strategies help applications deliver personalized content, enforce access control, and monitor for suspicious activity or fraud.

However, with great power comes great responsibility. User identification must be implemented with strong data protection measures, including encryption, secure data storage, and limited data retention. It also requires compliance with international privacy regulations such as the General Data Protection Regulation (GDPR) in the EU and the California Consumer Privacy Act (CCPA) in the U.S. These laws mandate clear user consent, the right to access or delete personal data, and transparency in how user data is collected and used. Properly implemented, user identification not only enhances user experience and security but also builds trust between users and digital platforms.

3.2.3 Current state of Technology on User Identification in Websites or Smartphone Applications

Summarizing the information above and analyzing the current state of the art techniques we derive (as a general result) the fact that the basic user authentication mechanisms based on the technologies used today on websites or applications are the following:

- User Authentication on Login/Password;
- Cookies and Local Storage;
- IP Address Tracking;
- Browser or Device Fingerprinting;
- Biometric Authentication;
- Mobile Device Identifiers;
- Behavioral Analytics;

- Session Management;
- Third-party Trackers - Social Login;
- Server-side Logs;
- Multi-Factor Authentication.

User Authentication ensures that a user is who they claim to be, allowing secure access to accounts, data, or personalized features. It mainly uses username and password fields (which is the most common method) to identify users. Its basic strength depends on the password complexity. It is usually stored in a secure encrypted storage (hashed and salted in a database). It also uses techniques like rate limiting and lockouts to prevent brute-force attacks. Its weaknesses are the fact that depends heavily and it is susceptible to reuse attacks while usernames and passwords can be phished or leaked in breaches.

Cookies are small encrypted text files that websites store on our browser. When we visit a website, it might send a cookie to our browser saying something like, "User123 came here." Next time we visit the same site (within a standard period of time), our browser sends that cookie back to the website, which lets the site know it’s us again.

IP Address is the ID of our device on the internet, assigned by our Internet Service Provider (ISP). Websites can see our IP when we connect on them. This can sometimes be used to identify users or determine our general location (like our city or region). For example Netflix uses our IP to show content available in our country. IPs can also help us Geolocate a user (e.g., city/country), Detect VPN/proxy use, limit access by region.

Browser or Device Fingerprinting is a simple way to "fingerprint" your browser or device based on its unique configuration and it is considered to be a kind of user detection like identifying a person by their unique handwriting. Websites or apps gather info like your operating system, screen size and resolution, browser type and version, language, Timezone, fonts installed, plugins installed, apps installed and more. The combination of all this creates a near-unique signature that provides important information which can help services detect anomalies in a probabilistic way. One interesting effect of Browser Fingerprinting is the fact that even if our computer blocks cookies, websites might still recognize us based on our browser fingerprint.

Biometric Authentication is a new approach mainly used by applications on smartphones or supported browsers. It usually implements techniques like Face ID,

Touch ID, fingerprint, retina scan. On smartphones it is usually layered on top of device security or passcodes. It provides high convenience, total confidentiality and it is really difficult to forge. On the other hand it is limited to devices with biometric hardware sensors and software support and last but not least there are serious privacy and ethical concerns if mishandled.

Mobile device identifiers are unique codes assigned to smartphones and tablets that help apps recognize and track individual devices. The most commonly used identifiers are Apple's IDFA (Identifier for Advertisers) and Google's GAID (Google Advertising ID), which are primarily used for advertising and analytics. These identifiers allow app developers and marketers to measure user behavior, deliver personalized ads, and track performance across different apps and sessions. Unlike permanent hardware IDs, these identifiers can typically be reset or limited by users through device settings, offering a degree of privacy control. With growing concerns around data privacy, modern mobile platforms now require apps to request explicit user permission before accessing these identifiers, especially for tracking purposes.

Behavioral analytics involves collecting and analyzing data on how users interact with a website or app to understand patterns, preferences, and potential risks. This includes monitoring actions such as mouse movements, clicks, scroll behavior, typing speed, and navigation paths. These insights help businesses improve user experience, personalize content, and detect anomalies that may indicate fraud or automated bot activity. For example, sudden changes in interaction speed or unusual navigation paths might trigger security measures. Behavioral analytics is also used in tools like CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) systems to distinguish humans from bots without requiring explicit challenges. While powerful, this technique raises privacy considerations, as it can track users silently and build detailed behavioral profiles over time.

Session management is the process of maintaining a user's state and identity across multiple interactions with a website or application. When a user logs in, a session is created and typically represented by a unique session ID stored in a cookie or passed as a token that allows the server to recognize the user across different pages or actions without requiring repeated authentication. Effective session management ensures a seamless user experience while also maintaining security, such as by expiring sessions after a period of inactivity or requiring re-authentication for sensitive actions. Poor

session handling can lead to vulnerabilities like session hijacking, where attackers gain unauthorized access to a user's session. Secure implementation involves techniques such as using HTTPS, setting secure cookie flags, and implementing mechanisms like token rotation or idle timeouts.

Social Login or OAuth uses third-party identity providers like Google, Facebook, Apple, Microsoft, etc. The main benefits of this technique are the facts that it is considered to be a fast and user-friendly experience for the users since there is no need to manage passwords. Often includes a built-in two-factor authentication (2FA). On the other hand, the main negative results are the facts that it ties users to third-party services which means that this technique may leak metadata to those external companies' services. Furthermore, some users distrust the tracking implications that arises in these cases since many users have the feeling that the external authenticator services may be able to track their overall online activity.

Server-side logs are records automatically generated by a website or application's backend systems that capture details about every request made to the server. These logs typically include information such as IP addresses, timestamps, requested URLs, user-agent strings, response codes, and error messages. They are essential for monitoring system performance, debugging issues, detecting suspicious activity, and conducting security audits. For example, repeated failed login attempts from the same IP may signal a brute-force attack, while analyzing traffic patterns can help identify bottlenecks or high-traffic periods. Although server-side logs are powerful for diagnostics and analytics, they must be managed carefully to avoid exposing sensitive user data and to comply with privacy regulations like GDPR. Proper log retention policies and access controls are essential to balancing operational insight with data protection.

The Two-Factor Authentication (2FA) or Multi-Factor Authentication (MFA) as it is called in general adds an extra layer of security [105]. Typically it uses a password + one-time code generator (e.g SMS, email, authenticator app) or a combination of password + biometric detector in order to identify the user's identity. Its benefit is that it Improves resistance to phishing, credential leaks or session hijacking. One common problem for users (that they must comply) is the fact that this techniques increases complexity and the overall time needed to identify users while users are also obligated to retain the same phone numbers or email accounts. Never the less, this remains the

most commonly used technique for the online e-banking infrastructure or the general money transaction system worldwide.

3.3 Benefits from Targeted and Selective Promotion to Specific Users

3.3.1 Demonstrating Main Benefits

Targeted and selective promotion refers to the strategic marketing approach in which promotional messages, offers, or campaigns are directed at a clearly defined segment of users rather than the general public. This method leverages data analytics, user behavior insights, and segmentation tools to deliver tailored content to specific individuals or groups. The benefits of this approach are substantial, enhancing both business performance and user experience. Moreover, targeted and selective promotion to specified users is a powerful tool for modern businesses. It enables smarter resource allocation, fosters deeper customer relationships, and drives superior outcomes by delivering precisely what the user needs when and where they need it.

The main benefits from targeted and selective promotion can be categorized as follows:

- Increased Marketing Efficiency;
- Improved Customer Engagement and Loyalty;
- Higher Conversion Rates;
- Data-Driven Insights and Continuous Improvement;
- Competitive Advantage.

Dealing with Increased Marketing Efficiency, targeted promotion ensures that resources are allocated to audiences most likely to respond. By focusing only on users who meet certain criteria such as age, location, purchase history, or browsing behavior, companies reduce wasted spending on uninterested or irrelevant audiences. This leads to higher return on investment (ROI) and more efficient use of marketing budgets.

Supporting Improved Customer Engagement and Loyalty, when users receive content and offers that align with their needs or interests, they are more likely to engage. Personalized promotions foster a sense of recognition and relevance, which enhances customer satisfaction and builds long-term loyalty. Selective outreach also reduces the risk of overwhelming users with irrelevant messages, contributing to a better overall

experience. Moreover, acquiring users through targeted promotions tends to be cheaper than broad campaigns, especially when using lookalike audiences or retargeting. Also, targeted promotions can be used to re-engage inactive users or reward loyal ones with special offers.

Higher Conversion Rates can be achieved when promotions are targeted at users who have already shown interest in similar products or services and the result is naturally more effective. By delivering the right message at the right time to the right person, targeted marketing significantly boosts the chances of conversion. Whether the goal is a purchase, sign-up, or any other call-to-action, personalization drives results.

Moreover, Data-Driven Insights and Continuous Improvement can be the result of targeted campaigns which can generate rich data that helps businesses better understand user preferences and behavior. These insights can be used to fine-tune future promotions, develop new products, or refine segmentation strategies. Over time, this leads to smarter marketing decisions and continuous performance improvement.

Last but not least, companies can have a Competitive Advantage because thanks to effective targeted and selective promotion they can differentiate themselves in crowded markets. By delivering more relevant and recent content than competitors, they can capture customer attention, strengthen brand perception, and drive customer acquisition more effectively.

3.3.2 Selective or Targeted Promotion Areas and Categorizations

Targeted promotions can be applied across a variety of industries to enhance customer engagement and drive sales by delivering personalized offers based on user behavior, preferences, and demographics. In retail and e-commerce, businesses use targeted promotions through email marketing, personalized website experiences, and loyalty programs to boost conversion rates. Digital advertising leverages platforms like social media and search engines to serve tailored ads to specific audience segments. In sectors like telecommunications and banking, companies use customer data to promote relevant plans, services, or financial products. Travel and hospitality industries offer customized deals based on travel history or location, while media and entertainment platforms recommend content or subscription upgrades aligned with user preferences. Even in healthcare, food services, and automotive sectors, targeted promotions are used to

suggest relevant products, services, or maintenance offers, making marketing efforts more efficient and impactful.

In general the main areas where selective or targeted promotion can be applied are the following:

- Retail and E-Commerce (email marketing, on-site promotions, push notifications, loyalty programs);
- Digital Advertising (social media ads, search engine marketing, display ads);
- Telecommunications (plan upgrades, churn prevention, add-on services);
- Travel and Hospitality (flight/hotel deals, upsell offers, geo-targeted promotions);
- Banking and Financial Services (credit card promotions, loan offers, cross-selling);
- Media and Entertainment (content recommendations, subscription upgrades, event promotions);
- Healthcare and Wellness (health plans and services, medication reminders and offers, fitness apps);
- Food and Beverage (restaurant offers, product bundling, loyalty discounts);
- Automotive (service reminders and discounts, new model promotions, lease / finance offers);
- Education and Online Learning (course recommendations, discounted bundles, trial offers).

Personalized promotions focus on delivering specific offers and messages to distinct customer segments. This approach leverages various targeting methods to ensure promotions are relevant and engaging for the intended audience, potentially leading to higher conversion rates and increased revenue.

In general the main different categorizations where targeted promotions can be applied are the following:

- Demographics (age, gender, income, education);
- Geographic Location (geo-targeting);
- Purchase History and Behavior (behavioral targeting, customer segmentation);
- Online Behavior and Interests (interest-based targeting, cross-device targeting);

- Retargeting;
- Lookalike Audiences.

In more details, tailoring promotions based on demographic characteristics allows businesses to target specific age groups, genders, or income levels with relevant offers. For example, luxury car ads might target higher income brackets, while college-related services could focus on younger age groups. Geo-targeting involves delivering promotions based on a user's physical location, whether it's a country, city, or even a specific neighborhood. Behavioral targeting analyzes user activity on a website or app to identify their preferences and purchase intent, allowing for personalized recommendations and offers. Customer segmentation is targeting promotions based on customer segments (e.g., frequent shoppers, new customers) and allows for tailored offers that resonate with each group. In interest-based targeting platforms like Facebook, Instagram, and Twitter allow advertisers to target users based on their expressed interests and activities on those platforms while in cross-device targeting strategy connects a user's activity across multiple devices, allowing for consistent messaging and promotions regardless of the device being used. Retargeting involves showing ads to users who have previously interacted with a website or app, reminding them of the products they viewed or added to their cart. Last but not least, lookalike audience targeting method identifies users who share similar characteristics with a company's existing customer base, expanding reach to potential new customers.

3.4 Artificial Intelligence Neural Networks and its Principles

3.4.1 Introducing Artificial Intelligence Mechanisms and Principles

Artificial Intelligence (AI) is transforming the landscape of technology, with neural networks standing as one of its most influential components. Neural networks, inspired by the structure and function of the human brain, are the backbone of many modern AI systems. From powering recommendation systems and voice assistants to enabling self-driving cars and medical diagnoses, these networks are essential to achieving machine learning and deep learning goals. Understanding the principles behind neural networks is essential for grasping how AI systems learn, adapt, and perform complex tasks.

In recent decade, the rapid advancement of technology has driven us in a new era defined by the capabilities of Artificial Intelligence (AI). Among the many components that power AI, neural networks have emerged as a foundational architecture, enabling machines to learn from data, recognize patterns, and make intelligent decisions. These networks are loosely modeled after the human brain (as noticed previously), but consisting of interconnected nodes (artificial neurons) that process information in a manner that reminds of biological neural activity. While the inspiration may be biological, the implementation is mathematical and computational, allowing neural networks to perform highly complex tasks that were once thought to be exclusive to human intelligence.

Neural networks are central to the development of machine learning and deep learning, two subfields of AI that have experienced explosive growth due to their success in handling unstructured data such as images, audio, and natural language. Their applications are widespread, ranging from voice recognition systems like Siri and Alexa to more advanced systems such as autonomous vehicles and AI-powered medical diagnostics. The ability of neural networks to learn directly from raw data and improve their performance over time has made them indispensable tools in solving problems across diverse industries.

Despite their growing ability, the inner workings of neural networks remain a mystery to many. Understanding how these systems function, how they are trained, and the principles that govern their design and optimization are crucial for anyone interested in AI, data science, or computer science in general. Neural Networks are not just sophisticated algorithms. They represent a paradigm shift in how we design software, moving away from explicitly programmed rules and toward systems that learn and adapt. However, ethical considerations, transparency, and responsible development must guide the continued evolution of neural networks to ensure they benefit society as a whole.

This thesis aims to provide a comprehensive overview of neural networks and the principles behind them. We will explore their structure, learning mechanisms, core theoretical concepts, and practical applications. Additionally, we will highlight the different types of neural networks and discuss current research trends that are shaping the future of AI.

3.4.2 Basic Foundations of Neural Networks

A neural network is a computational model designed to simulate the way biological neurons interact and process information. The basic building block is the artificial neuron, also known as a perceptron. A perceptron receives inputs, processes them through a mathematical function (usually involving weights and biases), and outputs a value (as Figure 3.1 shows).

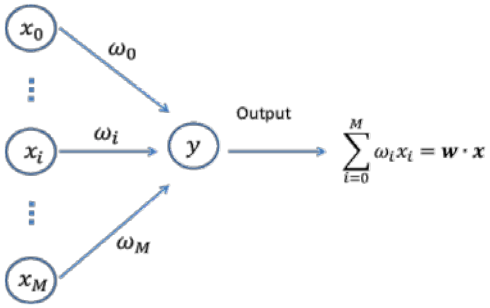


Figure 3.1: Visual representation of a Perceptron and its mathematical function

Neural Networks, as shown in Figure 3.2, consist of multiple layers of interconnected Perceptrons as follows:

- Input layer (receives the initial data e.g., image pixels, numerical values);
- Hidden layers (process inputs through complex transformations);
- Output layer (produces the final result or prediction).

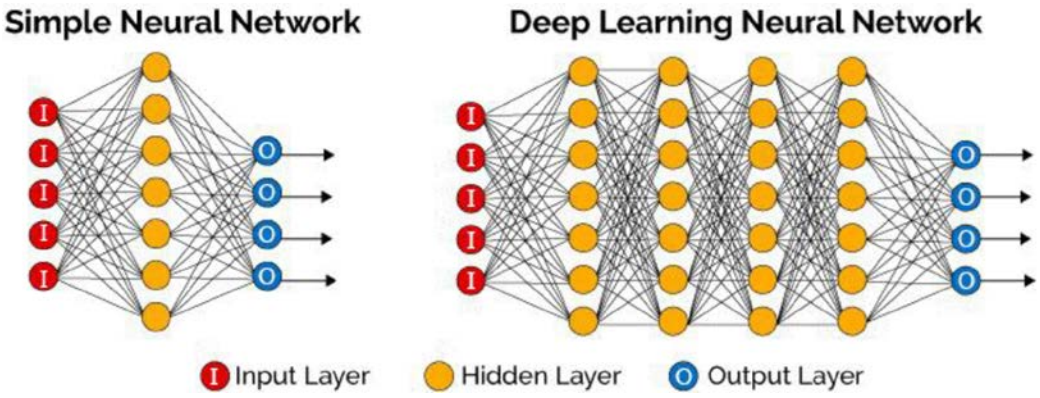


Figure 3.2: Simple versus Deep Neural Networking.

Each connection in the network has an associated weight that determines the influence of one neuron on another. Through learning algorithms, especially back-propagation, the network adjusts these weights to minimize error and improve accuracy.

3.4.3 Learning Mechanisms in Neural Networks

The learning process in Neural Networks is driven by data. The primary learning procedures are classified as:

- Supervised Learning;
- Unsupervised Learning;
- Reinforcement Learning.

In supervised learning, as shown in Figure 3.3, the network is trained on labeled data. Each input has a known output, and the network learns to map the input to the correct output by adjusting its weights. The error is calculated using a loss function, and gradients are computed through back-propagation to update the weights via an optimization algorithm, most commonly gradient descent.

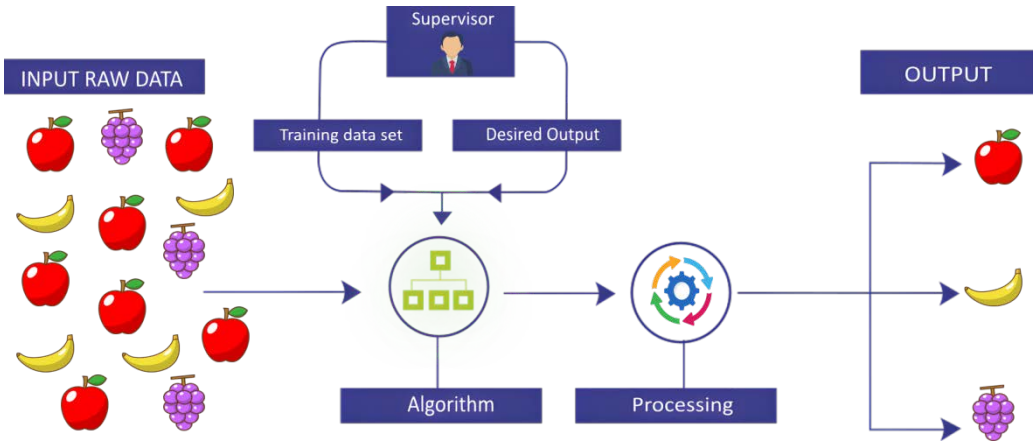


Figure 3.3: Visualization of a supervised Neural Network learning example.

In unsupervised learning (as shown in Figure 3.4), the network deals with unlabeled data. It attempts to discover hidden patterns or groupings in the data. Techniques like auto-encoders and clustering algorithms are used in unsupervised learning. In reinforcement learning, though not exclusive to neural networks, we train agents to make a sequence of decisions by rewarding desirable behaviors. Neural Networks often serve as function approximations in reinforcement learning systems.

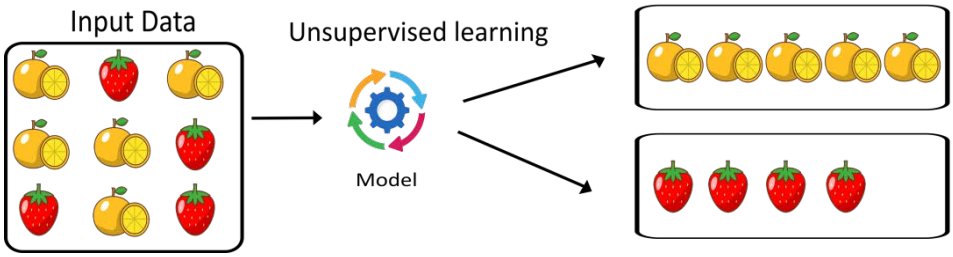


Figure 3.4: Visualization of an unsupervised Neural Network learning example.

3.4.4 Key Principles of Neural Networks

Several core principles guide the design and operation of neural networks:

- Non-Linearity;
- Hierarchy of Features;
- Generalization;
- Parallel and Distributed Processing;
- Gradient-Based Optimization.

In more details, one of the most fundamental principles in neural networks is non-linearity, which allows these systems to model complex relationships in data. If a neural network used only linear transformations, regardless of how many layers it had, the final output would still be a linear combination of the input. This limitation would make it impossible to solve problems where outputs do not follow a straight-line relationship with inputs. To overcome this, activation functions like ReLU (Rectified Linear Unit), sigmoid, and tanh are introduced. These non-linear functions are applied after each neuron's weighted sum calculation, enabling the network to stack layers and model with complex behaviors. For instance, ReLU helps to mitigate the vanishing gradient problem in deep networks and introduces sparsity, which improves learning efficiency. Without non-linearity, neural networks would be incapable of distinguishing fined patterns in images, audio, or text, rendering them ineffective for real-world AI applications.

Another critical concept is the hierarchical feature extraction capability of Deep Neural Networks (DPP), shown previously in Figure 3.2. Shallow networks with only one or two layers can learn basic patterns but struggle with complex abstractions. Deep

Networks, in contrast, can extract features at multiple levels of abstraction. For example, in a complicated Neural Network, used for image recognition, early layers might detect edges and textures, while deeper layers recognize shapes, objects, or even entire scenes. This hierarchical processing mirrors how the human visual cortex operates and allows machines to interpret raw sensory data more effectively. The depth of a neural network contributes not only to its expressive power but also to its capacity to perform transfer learning, where features learned in one task can be adapted for another, reducing the need for massive labeled datasets in every new application.

Concerning generalization and over-fitting, the Neural Network's ultimate goal is not to memorize training data, but to generalize and to perform well on unseen data (as shown in Figure 3.5). This is especially important in real-world applications where models must deal with noisy, incomplete, or novel inputs. However, due to their high capacity, neural networks are likely to suffer from over-fitting, where they learn the training data too precisely, including noise or outliers, and fail to generalize. To combat this, several regularization techniques are employed. Dropout, for instance, randomly disables a subset of neurons during training, forcing the network to develop redundant and more robust features. L2 regularization penalizes overly large weights, and early stopping halts training once validation performance begins to decline [106]. These strategies ensure that the network does not become overly complex and retains its ability to function in dynamic, unpredictable environments.



Figure 3.5: Representation of an underfitted vs overfitted and a generalized Neural Network.

Dealing with parallel and distributed processing, due to the vast number of computations involved in training deep neural networks, especially with large datasets, parallel and distributed processing has become a necessity. Modern neural networks, particularly in fields like natural language processing or computer vision, may have billions of parameters. Training such models on a single CPU would be computationally

infeasible. Graphics Processing Units (GPUs) and specialized hardware like TPUs [107] (Tensor Processing Units) are optimized for matrix operations and allow for the rapid computation of thousands of operations simultaneously [105]. Additionally, distributed systems split the workload across multiple machines or data centers. This scalability is what has enabled recent breakthroughs in AI, such as language models like GPT and image generation tools, to become practical and widely available.

Finally, Neural Networks rely on gradient-based optimization to learn from data. Using a method like back-propagation, the network calculates the gradient of the loss function with respect to each weight. This process reveals how each weight contributes to the error, and optimization algorithms such as stochastic gradient descent (SGD) [108], or Adam algorithm [109], adjust the weights to minimize this error. Gradient-based learning is efficient and scalable, enabling networks to gradually improve performance over many iterations. Adam is an adaptive optimization algorithm commonly used in training neural networks. It adjusts the learning rates of individual parameters, effectively minimizing the error by adapting to the "steepness" of the error surface for each parameter. This means that Adam can converge faster and more efficiently than simpler optimization methods like stochastic gradient descent, especially in complex models. However, the success of this approach depends heavily on factors like learning rate, weight initialization, and data quality. Poor choices in these areas can lead to slow convergence or getting stuck in local minima. Therefore, understanding and tuning the optimization process is essential for building high-performing neural networks.

3.4.5 Types of Neural Networks

Several types of neural networks exist, each suited to specific tasks:

- Feed-forward Neural Networks (FNNs), offering Simplicity and Versatility
- Convolutional Neural Networks (CNNs), the Masters of Visual Perception
- Recurrent Neural Networks (RNNs), Learning Through Time
- Generative Adversarial Networks (GANs), for Creative Intelligence
- Transformers, the Powerhouse of Language Understanding

Feed-forward Neural Networks (FNNs) are the most basic type of artificial neural network and serve as the foundation for more advanced models. In these networks, information flows in one direction, from input to output, without loops or cycles. Each

layer is fully connected to the next, and neurons are activated based on weighted sums of their inputs. FNNs are highly versatile and can be used for a variety of tasks such as classification, regression, and pattern recognition. Their main strength lies in their simplicity, making them easy to understand and implement. However, because they lack memory and do not process temporal or spatial relationships explicitly, they are not ideal for time-series data, sequences, or structured spatial data like images. This limitation has led to the development of specialized architectures better suited for complex data structures.

Convolutional Neural Networks (CNNs) have become the prime architecture for image-related tasks due to their ability to capture spatial hierarchies in data. CNNs use convolutional layers to detect local patterns such as edges, textures, or colors by sliding filters across the input. This operation preserves spatial relationships and reduces the number of parameters compared to fully connected layers. Pooling layers further con-dense information by down-sampling, improving computational efficiency and reducing the risk of over-fitting. Because of their spatial awareness, CNNs excel at tasks like image classification, facial recognition, medical image analysis, and even real-time object detection. Their structure also makes them interpretable to some degree and filters in earlier layers often correspond to specific visual features. CNNs demonstrate how architectural innovation can tailor a neural network to a specific data type, vastly improving its performance.

Unlike FNNs and CNNs, Recurrent Neural Networks (RNNs) are specifically de-signed to handle sequential data. This includes time-series data, natural language, and any input where context matters. RNNs have connections that form cycles (as shown in Figure 3.6), allowing the output of one step to influence the next, essentially giving the network a form of short-term memory. However, traditional RNNs suffer from issues like vanishing and exploding gradients, which hinder their ability to capture long-range dependencies. To overcome these challenges, advanced variants like Long Short-Term Memory (LSTM) [110] networks and Gated Recurrent Units (GRUs) [111] were developed. These architectures include gating mechanisms that regulate the flow of information, enabling the network to "remember" or "forget" specific inputs over time. As a result, RNNs and their variants are indispensable for tasks like language modeling, speech recognition, machine translation, and sentiment analysis.

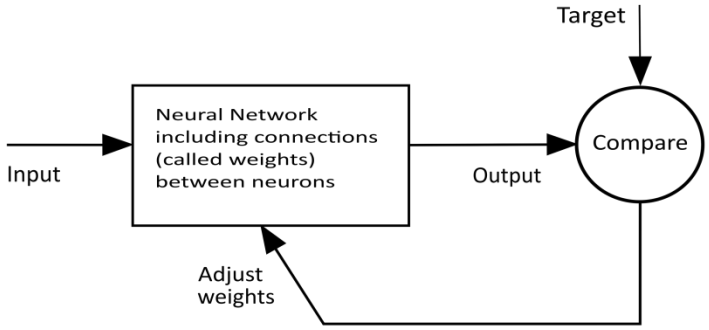


Figure 3.6: Recurrent Training on Neural Networks.

Generative Adversarial Networks (GANs) are among the most groundbreaking innovations in deep learning, especially in the field of generative modeling. A GAN consists of two networks, the generator and the discriminator, that are trained simultaneously in a game-theoretic setup. The generator tries to produce data indistinguishable from real data, while the discriminator evaluates whether a given input is real or generated. Over time, the generator improves until its outputs become convincingly realistic. GANs have opened the door to applications like Deepfake creation, art generation, image-to-image translation, and synthetic data generation. However, training GANs is notoriously unstable and requires careful balancing. Despite these challenges, GANs have pushed the boundaries of what neural networks can do, moving from classification to creative synthesis which is an area previously thought to be uniquely human.

Transformers represent a major leap in Neural Network design, especially in natural language processing (NLP). Unlike RNNs, which process inputs sequentially, transformers use self-attention mechanisms to evaluate all parts of the input simultaneously. This parallelism enables significantly faster training and better performance on long sequences. Models based on transformers, such as Bidirectional Encoder Representations from Transformers (BERT) [112-116], Generative Pre-trained (GPT) [115-117], and T5 [118-120], have achieved state-of-the-art results in tasks like text generation, translation, summarization, and question answering. Transformers are not only more efficient but also better at capturing long-range dependencies and contextual relationships. They have also been adapted for non-text domains, including vision (e.g., Vision Transformers, or ViTs) and reinforcement learning. Their flexibility

and performance have made transformers the dominant architecture in modern AI research and applications.

3.4.6 Neural Networks' Applications

Neural networks are central to a vast array of applications:

- Healthcare (diagnosing diseases from images, predicting patient outcomes);
- Finance (fraud detection, algorithmic trading, risk assessment);
- Autonomous Systems (self-driving cars, robotics);
- Natural Language Processing (machine translation, sentiment analysis, question answering);
- Creative Industries (AI-generated art, music, and literature).

Neural networks have dramatically reshaped entire industries by automating tasks once thought to require human expertise. In healthcare, for example, convolutional neural networks can analyze medical images with accuracy rivaling or even exceeding that of human radiologists, enabling early detection of conditions such as cancer or diabetic retinopathy [121-124]. Predictive models help anticipate patient outcomes and optimize treatment plans, leading to more personalized and effective care. Meanwhile, financial institutions deploy neural networks to detect fraudulent transactions in real-time, assess credit risk more accurately, and automate trading strategies [125-129]. In manufacturing, AI-powered quality control systems identify defects on production lines faster and more consistently than human inspectors [130-132]. These examples illustrate how neural networks are not merely academic curiosities but they are practical, transformative tools that deliver measurable value.

Beyond traditional industries, neural networks are critical enablers of autonomous systems and machines that can perceive, reason, and act in complex, dynamic environments. Self-driving cars [133-135], for instance, rely on deep neural networks to interpret sensor data, recognize objects, predict other agents' behaviors, and plan safe routes. This integration of perception and decision-making is crucial for navigating the unpredictable nature of real-world driving. In robotics, neural networks allow machines to learn motor skills through reinforcement learning, adapting to new tasks without explicit programming [136-140]. Such capabilities have profound implications not only for transportation but also for logistics, defense, and search-and-rescue operations.

However, deploying these systems in the real world raises critical questions about safety, accountability, and ethical design, which the field must continue to address.

One of the most visible impacts of Neural Networks is in natural language processing (NLP), where architectures like transformers have revolutionized how machines understand and generate human language [141-148]. Voice assistants, chat bots, and translation systems now provide far more fluent and context-aware interactions than ever before. Large language models can summarize documents, answer questions, and even generate creative text, enabling new forms of human-computer collaboration. This progress has democratized access to information and made technology more inclusive by bridging linguistic barriers. Yet these advances also introduce challenges, such as the potential for generating misinformation, biases encoded in training data, and the difficulty of making AI systems truly explainable [148,149]. Addressing these issues is essential to ensuring that language technologies are used responsibly and ethically.

Neural Networks are not only tools of analysis and prediction but they are also engines of creativity. Generative Adversarial Networks (GANs) and transformer-based generative models have enabled machines to create realistic images, compose music, write stories, and even design new molecules for drug discovery [150-153]. These capabilities expand human creative potential by providing new forms of inspiration, accelerating design cycles, and enabling experimentation at a scale previously impossible. However, they also raise important questions about authorship, originality, and misuse. Deepfakes [154-156], for example, demonstrate how generative technologies can be used maliciously to spread disinformation. Balancing the enormous creative promise of these tools with safeguards against abuse is a pressing concern for researchers, policymakers, and society at large.

3.4.7 Neural Networks and Future Directions

The future of Neural Networks lies in making them more efficient, interpretable, and generalizable. Current research focuses on:

- Neuro-symbolic systems (combining neural networks with symbolic reasoning);
- Few-shot and zero-shot learning (learning with minimal data);
- Explainable AI (XAI) (making neural network decisions understandable);
- Neuromorphic computing (building hardware that mimics the brain's architecture).

Looking ahead, the development of Neural Networks is focused on making them more capable, generalizable, interpretable, and efficient [138,157,158]. Research into Neuro symbolic systems [159,160] aims to combine the pattern recognition power of neural networks with the logic and reasoning of symbolic AI, potentially overcoming limitations in generalization and explainability. Few-shot and zero-shot learning seek to enable models to learn new tasks with minimal data, making AI accessible for domains where large annotated datasets are unavailable [161-164]. Efforts in Explainable AI (XAI) [165-168], aim to open the "black box" of deep learning so that human users can understand, trust, and verify AI decisions which are critical for high-stakes domains like medicine, law, and finance. Meanwhile, advances in neuromorphic hardware promise to make AI systems more energy efficient by mimicking the brain's architecture [169-172]. These directions represent an exciting and necessary evolution where neural networks become not only more powerful but also safer, more transparent, and more aligned with human values.

3.5 User Classification Neural Networking Algorithms

3.5.1 Single Layer Neural Network

User classification with neural networks refers to the process of automatically categorizing users into predefined groups based on their behavior, attributes, or interactions. For example, an e-commerce site might classify users as "new," "returning," or "high-value" based on purchase history, session length, and click patterns. Neural networks are well-suited for this task because they can learn complex, non-linear relationships in the data without requiring manual feature engineering. By training on large amounts of user data, these models can generalize to new users, predicting their class with high accuracy.

Neural networks used for user classification often include architectures like fully connected (dense) networks, recurrent neural networks (RNNs) for sequential data (e.g., browsing sessions), or even transformers for more sophisticated modeling of user behavior. The input features might include demographic data, transaction history, or time-series logs of user actions. During training, the network learns to map these features to user classes by minimizing a loss function, such as categorical cross-entropy. Once trained, the model can rapidly classify new users in real-time applications.

In this case we will be using a single layer neural network. A single Perceptron is limited to linearly separable data. With an ensemble of cooperating Perceptrons, we can overcome this limitation. In the Figure 3.7, we show the algorithm for the according Python application.

```
import numpy as np
import matplotlib.pyplot as plt                                    (1)
import neurolab as nl

# Load input data
text = np.loadtxt('data_simple_nn.txt')

# Separate it into datapoints and labels                           (2)
data = text[:, 0:2]
labels = text[:, 2:]

# Plot input data
plt.figure()                                                       (3)
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')

# Minimum and maximum values for each dimension
dim1_min, dim1_max = data[:,0].min(), data[:,0].max()              (4)
dim2_min, dim2_max = data[:,1].min(), data[:,1].max()

# Define the number of neurons in the output layer
num_output = labels.shape[1]                                       (5)

# Define a single-layer neural network
dim1 = [dim1_min, dim1_max]                                        (6)
dim2 = [dim2_min, dim2_max]
nn = nl.net.newp([dim1, dim2], num_output)

# Train the neural network                                         (7)
error_progress = nn.train(data, labels, epochs=100, show=20, lr=0.03)

# Plot the training progress
plt.figure()
plt.plot(error_progress)                                           (8)
plt.xlabel('Number of epochs')
plt.ylabel('Training error')
plt.title('Training error progress')
plt.grid()
```

Figure 3.7: User's Classification Algorithm.

In more details, in section 1, we import the necessary libraries. Neurolab is a simple and powerful Neural Network Library for Python. Contains based neural networks, train algorithms and flexible framework to create and explore other neural network types. In section 2, we use the input data from the file data_simple_nn.txt (see Appendix A.1). We separate the data into the first two columns which contains data points and the last two columns which contains labels that represents the user's type/classification in binary form with four distinguish values in this case. In section 3, as Figure 3.8 (a) shows, we plot the points on a graph. In section 4, we determine the maximum and minimum values that each dimension can take. In section 5, we specify the number of neurons in the output layer, which is two in this case. In section 6, we define a single-

layer neural network using the above parameters (two inputs, two outputs). In section 7, we train the Perceptron with the data from the training set, we repeat the procedure hundred times and we show intermediate data every 20 cycles. In section 8, as Figure 3.8 (b) shows, we plot the training progress using the error metric (error_progress).

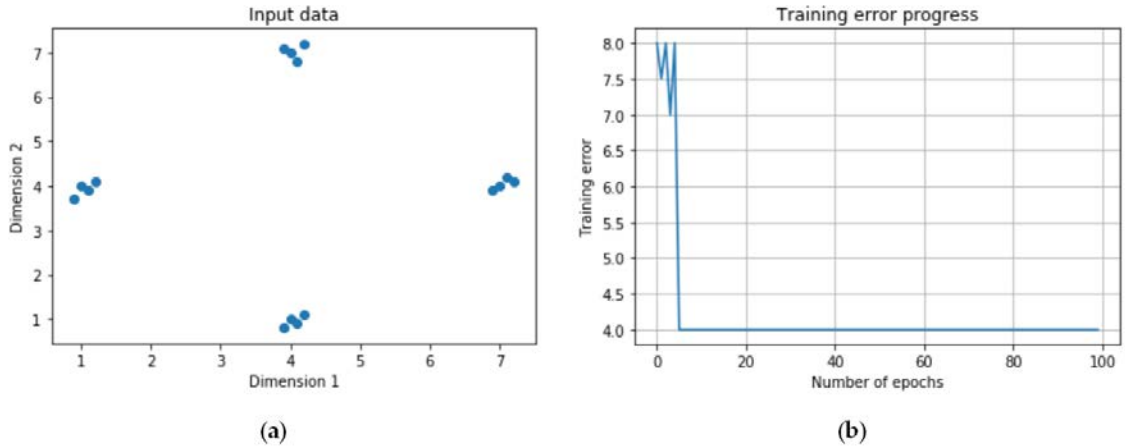


Figure 3.8: (a) Visual representation of users' data from input file; (b) The Training error progress.

Next, let's define some new sample test data points and run the network on these points. These new data points can simulate the according information of new system's users. The nn Python variable now contains our trained and configured Neural Network. To get this chart, simply add the following lines to your code as shown in Figure 3.9 (a). In Figure 3.9 (b), we can see the training data in blue dots and the new users' test data in orange color.

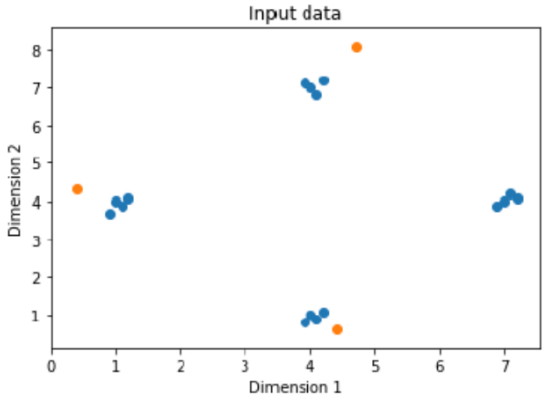
```
# Run the classifier on test datapoints
print('\nTest results:')
data_test = [[0.4, 4.3], [4.4, 0.6], [4.7, 8.1]]

data_test = np.array(data_test)

# Plot input and test data
plt.figure()
plt.scatter(data[:,0], data[:,1])
plt.scatter(data_test[:,0], data_test[:,1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input & Test data')

plt.show()
```

(a)



(b)

Figure 3.9: (a) The Python algorithm to classify new users; (b) Visual representation of the new users info entries.

Running the above test with the according users' data the algorithm will correctly report the following results:

- Input data {0.4, 4.3} 1 => {0, 0} as output label for user's classification
- Input data {4.4, 0.6} => {1, 0} as output label for user's classification
- Input data {4.7, 8.6} => {1, 1} as output label for user's classification

3.5.2 Multi Layer Neural Network

A multi-layer Neural Network, also known as a Deep Neural Network, consists of an input layer, one or more hidden layers, and an output layer. Each layer is made up of interconnected nodes (neurons), and each connection carries a weight that adjusts during training to improve the network's performance. The hidden layers enable the network to learn and represent complex patterns by applying nonlinear transformations to the input data. These networks are trained using algorithms like back-propagation, which updates the weights based on the error between predicted and actual outcomes. Multi-layer neural networks are foundational to deep learning and are widely used in tasks such as image recognition, natural language processing, and autonomous systems.

In our case, to allow for greater accuracy, we need to give the Neural Network more freedom and flexibility. More than one layer of Perceptrons is required to extract the underlying patterns in the training data. In this example, we will see how to use a multilayer neural network as a regressor (value predictor). In Figure 3.10 (a), we show the according Python algorithm. In section one, we show how we can create some training data using the numpy package. In section two, we initialize the y variable using the sample data points previously created. The transformation is based on the equation y = 3x2 + 5 and then we use the norm function to perform normalization. The according equation was choosen just because of its simplicity and in a real situation the y variable must be extracted through real users' monitoring. Finally in section 3, we create a visual representation of the generated training data, as Figure 3.10 (b) shows.

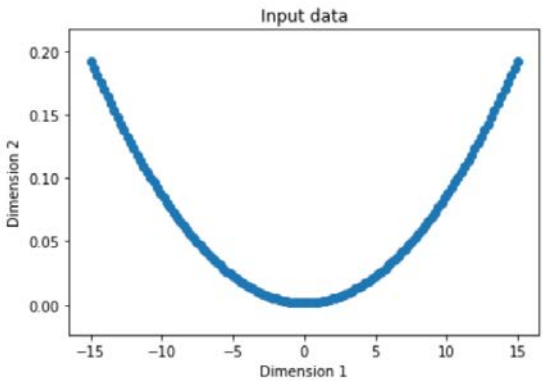
```python
import numpy as np
import matplotlib.pyplot as plt
import neurolab as nl

# Generate some training data
min_val = -15                                                    (1)
max_val = 15
num_points = 130
# Returns evenly spaced numbers over a specified interval.
x = np.linspace(min_val, max_val, num_points)
y = 3 * np.square(x) + 5                                         (2)
y /= np.linalg.norm(y)

# Create data and labels
data = x.reshape(num_points, 1)
labels = y.reshape(num_points, 1)

# Plot input data
plt.figure()                                                     (3)
plt.scatter(data, labels)
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Input data')
```

(a)



(b)

Figure 3.10: (a) Creating training data as input for a Multi Layered Neural Network; (b) Visual representation of the generated training data.

After creating the training data, we can continue with the Neural Networking section as Figure 3.11 shows. In more details, in section one we define the neural network architecture. In this case, we define a multilayer neural network with 2 hidden layers. The First hidden layer consists of 10 neurons, while the Second hidden layer consists of 6 neurons and the Output layer consists of 1 neuron. In section two, we set the training algorithm to gradient descent and train the model. In section three, we execute the Neural Network on the training data points. In section four, we demonstrate the progress of training. In section 5, we visualize the predicted values by the algorithm along with actual training data to help readers better understand the results.

```
# Define a multilayer neural network with 2 hidden layers;
# First hidden layer consists of 10 neurons
# Second hidden layer consists of 6 neurons
# Output layer consists of 1 neuron
nn = nl.net.newff([[min_val, max_val]], [10, 6, 1])                    (1)

# Set the training algorithm to gradient descent
nn.trainf = nl.train.train_gd
                                                                       (2)
# Train the neural network
error_progress = nn.train(data, labels, epochs=5000, show=500, goal=0.001, lr=0.001)

# Run the neural network on training datapoints
output = nn.sim(data)                                                  (3)
y_pred = output.reshape(num_points)

# Plot training error
plt.figure()
plt.plot(error_progress)                                               (4)
plt.xlabel('Number of epochs')
plt.ylabel('Error')
plt.title('Training error progress')

# Plot the output
x_dense = np.linspace(min_val, max_val, num_points * 2)                (5)
y_dense_pred = nn.sim(x_dense.reshape(x_dense.size,1)).reshape(x_dense.size)

plt.figure()
plt.plot(x_dense, y_dense_pred, '-', x, y, '.', x, y_pred, 'p')
plt.title('Actual vs predicted')

plt.show()
```

Figure 3.11: The Multi Layer Neural Network for Users' Classification.

The result of the execution of the sections 4 and 5 in Figure 3.11 is shown in Figures 3.12 (a) and (b) accordingly. We must notice that due to the nature of the developed Multy Layered Neural Network and the inner imported library modeling and the operational or algorithmic mechanism, the exported results may not always be the same. This means that different runs of the same algorithm can produce different results. Due to the nature of the Neural Networks, some libraries may use temporal or random features to perform their actions. Moreover, we can produce probably more accurate results if we provide different setups to our algorithm. For example we can expect better predicting performance from the Neural Network if we reduce the error limit, or increase the epochs value, or both! The problem in this case will be the fact that the new algorithm will need probably more time and CPU / Memory resources to successfully perform its tasks.

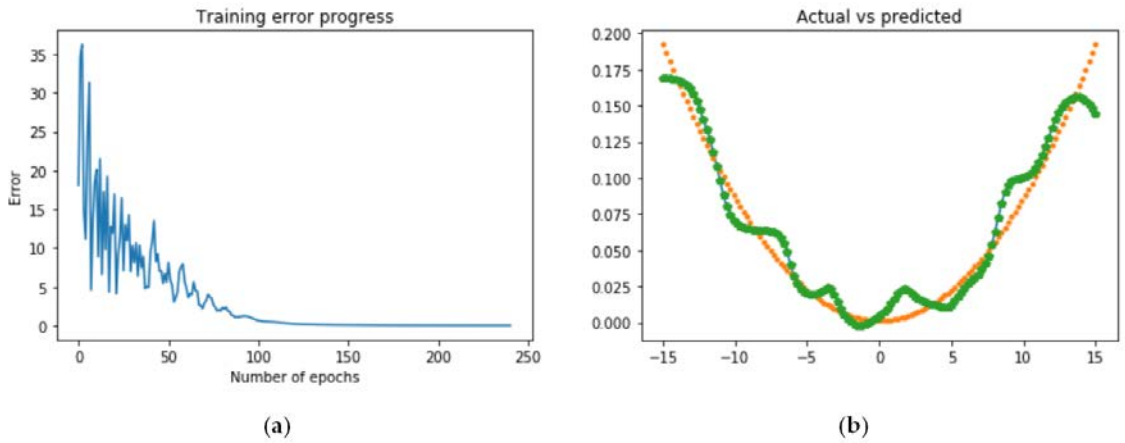


Figure 3.12: (a) Visualizing the training error progress of the Multi Layer Neural Network; (b) Demonstration of actual versus Neural Network predicted results for the same input data.

Deploying neural networks for user classification enables businesses to personalize experiences, target marketing campaigns more effectively, and improve customer retention. However, this approach also raises challenges around data privacy and fairness. Companies must ensure that their models do not reinforce harmful biases or discriminate unfairly. Additionally, the interpretability of neural networks can be limited, making it important to apply explainability techniques to understand and justify classification decisions. When implemented responsibly, Neural Network based user classification can deliver significant value while respecting user rights.

3.6 Identifying User through Neural Networking Applications

3.6.1 The K Nearest Neighbors (KNN) in Neural Networks

K Nearest Neighbors (KNN) is a classic non-parametric machine learning algorithm used for classification and regression. Though KNN itself is not a neural network, it is often discussed alongside them and can be combined with Neural Networks in hybrid systems. KNN operates on the principle of similarity. Given a new data point, it looks at the K closest examples (neighbors) in the training data and predicts the label (for classification) or value (for regression) based on them. For classification, this is typically done by majority voting among the K neighbors.

The main key steps in KNN are:

- Compute distances between the new point and all training points (e.g., Euclidean, Manhattan);
- Select K Nearest Neighbors with the smallest distances;
- Aggregate their labels/values to make a prediction.

The connection between K Nearest Neighbors and Neural Networks lies in their complementary strengths. While KNN is a simple, non-parametric method that relies on measuring similarity in feature space, Neural Networks excel at learning powerful representations of complex data. By using Neural Networks to extract meaningful features or embeddings, especially from high-dimensional or unstructured data like images or text, KNN can operate in a space where distances better reflect semantic similarity, improving its effectiveness. This synergy is evident in applications like few shot learning, where memory augmented networks and prototypical networks integrate KNN like neighbor search to generalize from limited examples. Overall, combining KNN with Neural Networks enables flexible, interpretable, and highly effective models that leverage both learned representations and instance based reasoning.

The advantages of K Nearest Neighbors come from its simplicity and intuitive approach. It makes predictions based on the idea that similar data points tend to have similar outcomes. One major strength is that it requires no training phase, making it easy to implement and adapt to new data without retraining a complex model. Its non-parametric nature allows it to model complex decision boundaries without assuming any specific functional form. Additionally, KNN can work well in cases where a meaningful distance metric is defined, enabling effective classification or regression even with small datasets. Also, because of the fact that predictions are based directly on stored examples, KNN also offers transparency. It is easy to see which neighbors influenced a prediction, making its decisions more interpretable than many black box models. These advantages make KNN a valuable baseline or component in hybrid systems, especially when combined with other methods to improve its scalability and robustness.

On the other hand, despite its simplicity, K Nearest Neighbors has several notable disadvantages that limit its scalability and effectiveness in certain scenarios. One of the primary issues is its computational cost at real time. Since KNN must compute the distance from a new point to every training example, prediction becomes slow and memory-intensive as the dataset grows. Additionally, KNN is highly sensitive to

irrelevant or redundant features, which can distort distance calculations and lead to poor performance. This issue is amplified in high dimensional spaces due to the curse of dimensionality, where data points tend to become equidistant, reducing the effectiveness of similarity based comparisons. KNN also requires careful tuning of hyper parameters like the value of K and the choice of distance metric, which can significantly affect its performance. Lastly, because it stores all training data and does not generalize in a traditional sense, KNN can be vulnerable to noise and outliers, potentially misclassifying points based on misleading neighbors. These drawbacks mean that while KNN is useful in specific contexts, it often requires preprocessing, feature selection, and optimization to perform competitively.

Combining K Nearest Neighbors with Neural Networks is especially valuable when the raw input space is too complex, noisy, or high-dimensional for direct distance based comparisons to be meaningful. Neural Networks excel at learning rich, task specific feature representations, for example, embeddings of images, text, or audio that capture underlying semantic similarities. Applying KNN in this learned feature space can dramatically improve its effectiveness, since distances now reflect more meaningful relationships. This approach is particularly useful in few shot learning, where Neural Networks extract features while KNN like mechanisms enable generalization from just a handful of labeled examples. It's also effective in retrieval tasks (like image search or recommendation systems), where the goal is to find similar items efficiently. Moreover, memory augmented and prototype based neural architectures explicitly integrate KNN like reasoning to support adaptable, interpretable predictions. In short, combining KNN with Neural Networks leverages the strengths of both. The representation power of Neural Networks and the instance based flexibility of KNN make them a powerful strategy for tackling complex, data-scarce or highly variable prediction problems.

3.6.2 Generic Implementation of the K Nearest Neighbors Neural Network

Recommender systems use the concept of nearest neighbors to find good recommendations. This is the process of finding the closest points to the input point from the data set and it is accomplished by sorting based on the proximity of the input data into various categories. To find the nearest neighbors of a given point in Python we will write the following code as shown in Figure 3.13.

```
import numpy as np
import matplotlib.pyplot as plt                                    (1)
from sklearn.neighbors import NearestNeighbors

# Input data
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9],
        [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1], [3.4, 1.9],
        [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])          (2)

# Plot input data
# Alternatively you may use the embeded function kneighbors_graph() by
# executing knn_model.kneighbors_graph()
# once your model is defined (i.e. after line 26)
plt.figure()
plt.title('Input data')                                            (3)
plt.scatter(X[:,0], X[:,1], marker='o', s=75, color='black')
# plt.show()

# Number of nearest neighbors
k = 5                                                              (4)

# Test datapoint
test_datapoint = [4.3, 2.7]

# Build K Nearest Neighbors model                                  (5)
knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors([test_datapoint])

# Print the 'k' nearest neighbors
print("\nK Nearest Neighbors:")                                    (6)
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])

# Visualize the nearest neighbors along with the test datapoint
plt.figure()
plt.title('Nearest neighbors')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='black')     (7)
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1],
        marker='o', s=250, color='b', facecolors='none')
plt.scatter(test_datapoint[0], test_datapoint[1],
        marker='x', s=75, color='r')

plt.show()
```

Figure 3.13: Python implementation of the K Nearest Neighbors Neural Network.

Analyzing the code above we can see that in section one we import the necessary packages. In this case we use the sklearn library. Scikit-learn is a Python library that offers a broad selection of advanced machine learning algorithms suitable for medium scale supervised and unsupervised tasks [173-176]. Designed to make machine learning accessible to non-experts through a high level general purpose language, it prioritizes ease of use, strong performance, thorough documentation, and consistent APIs. With minimal external dependencies and distribution under the permissive BSD license, it is well-suited for both academic and commercial applications. In section two, we define some two dimensional sample data points. In section 3, as we can see in Figure 3.14 (a), we visualize the input data using black circles. In section four, we specify the number of nearest neighbors to extract and a test data point. In section five, we create and train a KNN model using the input data and use it to extract the nearest neighbors to our test data point included in Python variable test_datapoint. In section six, print in command

line the nearest neighbors extracted from the model. Finally, in section seven, as we can see in Figure 3.14 (b), we visualize in a chart the given test point and the closest to the test point candidates.

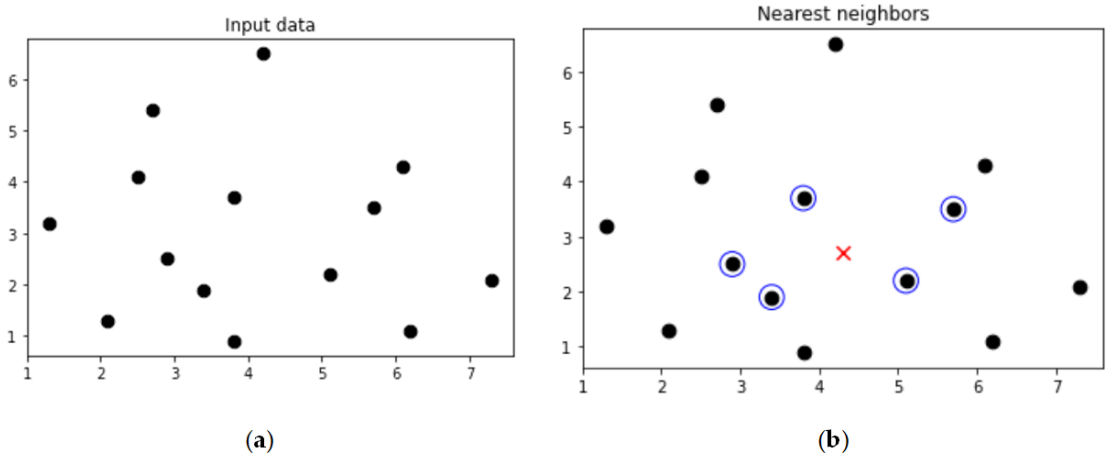


Figure 3.14: (a) Visualization of the KNN input data; (b) Visualization of the KNN results for k=5 and the given test point x.

3.6.3 Users' Identification using K Nearest Neighbors Neural Network

A K Nearest Neighbors classifier is a type of classification model that relies on the nearest neighbor approach to assign a class to a data point. It works by identifying the K closest points in the training dataset (i.e., known examples) and determining the class of the new point based on these neighbors. The model assigns the class that is most common among these K neighbors. The choice of K can vary depending on the specific problem or dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets

# Load input data
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')                    (1)
X, y = data[:, :-1], data[:, -1].astype(int)

# Plot input data
plt.figure()
plt.title('Input data')
marker_shapes = 'v^os'                                          (2)
mapper = [marker_shapes[i] for i in y]
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')

# Number of nearest neighbors
num_neighbors = 12                                              (3)

# Step size of the visualization grid
step_size = 0.01

# Create a K Nearest Neighbours classifier model
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='distance')

# Train the K Nearest Neighbours model                          (4)
classifier.fit(X, y)

# Create the mesh to plot the boundaries
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1             (5)
x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step_size),
        np.arange(y_min, y_max, step_size))

# Evaluate the classifier on all the points on the grid (6)
output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])

# Visualize the predicted output
output = output.reshape(x_values.shape)
plt.figure()                                                    (7)
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)
```

Figure 3.15: Training Neural Network for users' identification through class mapping.

As shown in Figure 3.15, in section one, we load the input data from the external csv file data.csv. The content of the data.csv file is shown in Appendix B.1. Concerning about the meaning of the content, in this example, the first two columns contains x, y coordinates for points in a two dimensional space and as a third column we use an integer number with values numbers from zero to three which represent four users' categories/classes. In this case our experiment tries to simulate a situation where for each user we know and we are able to measure two factors (the first two columns of the csv file) and our goal is to train the Neural Network with the third column as the expected result, which third column represent four users' categories. If the Neural Network is trained successfully we will be able to match any future new users' activity to the according user's type. Then, in section two, as shown in Figure 3.16 (a), we map the training points using four different shapes and the mapper variable to map each

shape to the different classes. At section 3, we define the number of nearest neighbors and the grid step size that will be used to visualize the model boundaries. In section four, we create the KNN model and train it with the training data. In section 5, we create the value grid that will be used to visualize the model. In section 6, we evaluate the classifier at all points in the grid to create a visualization of the boundaries. Finally, in section seven, as shown in Figure 3.16 (b), we create a color grid to visualize the pre-calculated model output for the entire two dimensional space.

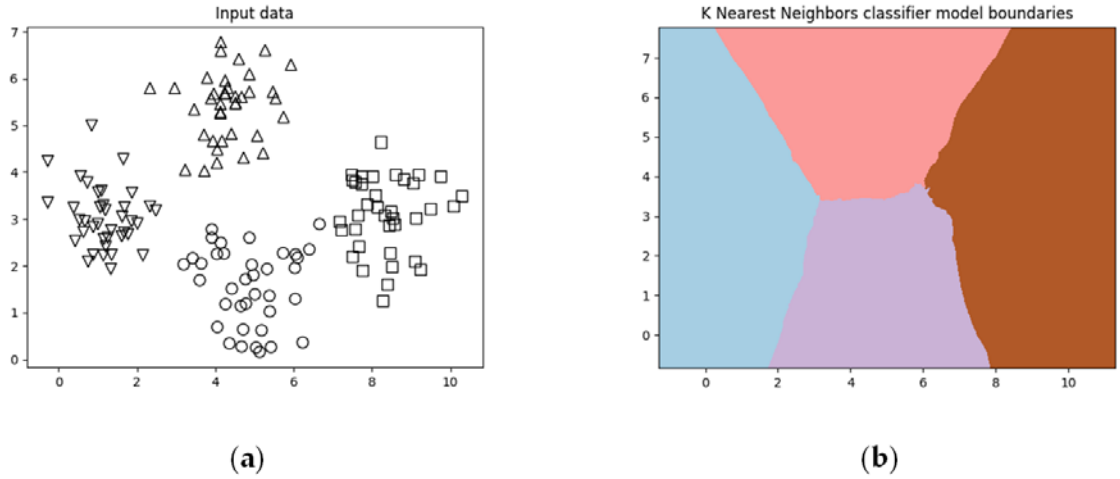


Figure 3.16: (a) Mapping of the training points using four different shapes; (b) Color grid to visualize the pre-calculated model output.

Moreover, by extending the functionality of the previous algorithm and as we can see in Figure 3.17, we can easily predict the users' account type and in such a case where we would have many input data we can even detect its unique identity.

```
# Overlay the training points on the map
for i in range(X.shape[0]):                                                    (1)
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=50, edgecolors='black', facecolors='none')
plt.xlim(x_values.min(), x_values.max())
plt.ylim(y_values.min(), y_values.max())                                       (2)
plt.title('K Nearest Neighbors classifier model boundaries')

# Test input datapoint
test_datapoint = [5.1, 3.6]
plt.figure()                                                                   (3)
plt.title('Test datapoint')
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i], s=75, edgecolors='black', facecolors='none')

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x', linewidth=6, s=200, facecolors='black')

# Extract the K nearest neighbors
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(int)[0]

# Plot k nearest neighbors
plt.figure()
plt.title('K Nearest Neighbors')                                               (4)

for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
            linewidth=3, s=100, facecolors='black')

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
        linewidth=6, s=200, facecolors='black')

for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')
print("Predicted output:", classifier.predict([test_datapoint])[0])           (5)
plt.show()
```

Figure 3.17: User' identification through Neural Network class mapping.

As we can see in Figure 3.17, in section one, we plot the training data over this color grid to visualize the data against the boundaries revealed by the KNN model. The result of this action is shown in Figure 3.18 (a) in which we can see the expected spatial partitioning. In section two, we set the X and Y bounds along with the title for the visualization. In section three, we define a test data point (representing the new user) to see how the classifier works. Also we create a new image with training data points and the test data point to see more clearly the topology of the two dimensional space. In section four, as shown in Figure 3.18 (b), we extract the K Nearest Neighbors at the test data point and plot them on a new chart. In section five we print the prediction (classification) of the test point, which in this case (with the given data) the result is "Predicted output: 1" which means that our test point x is classified as a type one user because eight out of the twelve neighbors are classified as type one.

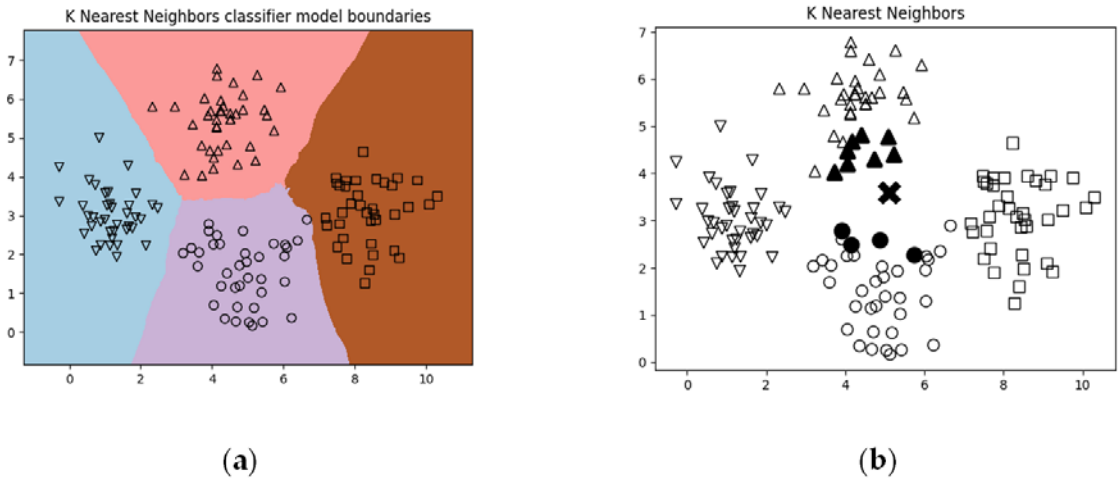


Figure 3.18: (a) KNN input training data over expected spatial partitioning; (b) Extraction of the 12 candidates of the K Nearest Neighbors at the test data point.

3.7 Account Preferences Predictions using Neural Networking AI Implementation

3.7.1 Introducing Account Preferences Prediction

In today's data-driven digital landscape, predicting users' account preferences has become central to enhancing personalization, improving engagement, and boosting customer satisfaction. Businesses from streaming services to e-commerce platforms and social media networks seek to anticipate what users want in order to tailor their offerings effectively. "User's account preferences predictions" refers to the use of analytical and machine learning methods to infer, forecast, and recommend settings, content, or features that align with an individual user's habits and interests. Understanding and implementing effective prediction systems is crucial not only for business growth but also for building trust and delivering value to users. As data privacy concerns, ethical and transparent handling of user preference prediction has also emerged as an essential component of this field.

3.7.2 Account Preferences Prediction Mechanisms

At its core, predicting a user's account preferences relies on data analysis. This involves gathering data such as past behavior (e.g., items purchased, pages visited), explicit

settings (e.g., language or notification preferences), demographic information, and even implicit signals like time of day usage patterns or click-through rates.

Machine learning models, particularly recommender systems, are widely used to make these predictions. Two main approaches dominate:

- Collaborative Filtering;
- Content-Based Filtering.

Collaborative Filtering analyzes patterns among users with similar behaviors. For example, if User A and User B have historically liked the same movies, a system might recommend to User A a movie that User B liked but User A hasn't seen yet. Collaborative filtering is widely used in recommendation engines such as those on Netflix or Amazon.

Content-Based Filtering relies on the attributes of items or settings the user has engaged with. If a user has chosen to receive sports notifications in the past, the system may predict that they will prefer other sports-related content or features. The model learns the features a user likes and finds similar options.

Advanced systems often use hybrid models that combine these approaches with deep learning techniques, natural language processing, and sequential modeling (e.g., using RNNs or Transformers) to capture complex patterns in user behavior over time.

The predictive modeling of user preferences has a broad range of applications. For example, content recommendations streaming platforms predict which movies or shows a user is likely to enjoy. Personalized playlists on music services are curated based on past listening habits. E-commerce personalization online retailers predict which products users might be interested in purchasing next. This includes recommending similar items, suggesting complementary products (cross-selling), or personalizing homepage layouts. In notification settings area, many apps allow prediction of which notifications a user wants. For example, a news app might prioritize topics the user frequently reads while minimizing unwanted alerts. Moreover, in privacy and security preferences section, some systems predict a user's likely security settings based on their past configurations. For example, suggesting two-factor authentication for users who prioritize security. Last but not least, in UI/UX personalization filed, apps can adjust

layouts, themes, and navigation flows based on predicted user preferences, creating a more personalized and satisfying experience.

3.7.3 Recommender Systems and Similarity Scores

To build a recommender system, it is important to understand how to compare different objects in our data set. The similarity score gives us an idea of how similar two objects are to each other.

There are two commonly used measures:

- Euclidean score, uses the Euclidean distance between two data points;
- Pearson score, a measure of correlation between two items.

The value of the Euclidean distance can take very large values and theoretically can even be unlimited. Therefore, we must convert it in the appropriate way so that the Euclidean score ranges from 0 to 1. Usually this process includes a normalization method where all the values that have been generated are modified appropriately to receive their corresponding value between 0 and 1. To do this, we must first find the minimum minA and maximum maxA value from the set of values that we already have. Then we define as new_minA the minimum possible value in the new set of values and as new_maxA the maximum possible value in the new set of values. The assignment of any value from the old set to the new set must be executed with a standard data modification process from the old set to the new one {minA, maxA} -> {new_minA, new_maxA}. In case we want the new set to have values between {0, 1}, the conversion method that we have to use is shown in Figure 3.19. For example, let's assume that our data belongs to the set {30-50} and let's say that we want to transform it so that it ranges from {0-1}. Deploying the function shown in the following figure, if we want to convert the number x=35 from the set {30-50} to its x_{scaled} counterpart in the set {0-1}, the operation we will need to perform is x_{scaled} = (35-30)/(50-30) = 5/20 = 0.25

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Figure 3.19: Min-Max Normalization function.

A large Euclidean value (distance) between two objects represents a small similarity. Correspondingly, a small Euclidean value (distance) represents a large similarity. When in particular the Euclidean value (distance) is equal to zero then it is an absolute match. Therefore, the Euclidean distance is inversely proportional to the Euclidean score.

On the other hand, The Pearson score uses the covariance between the two scores along with their individual standard deviations to calculate the score. The score ranges from -1 to +1:

- Pearson score between 0 and 1 (Positive correlation, which means that when one variable changes, the other variable changes in the same direction);
- Pearson score equal to 0 (No correlation, which means that there is no relationship between the variables);
- Pearson score between 0 and –1 (Negative correlation, which means that when one variable changes, the other variable changes in the opposite direction).

The experiment that we will use to help us run the basic scenario includes the followings. Let's assume that our dataset consists of people and their various preferences for movies based on a rating system that includes float numbers from zero to five. We will first try to find the degree of similarity between users by studying the ratings they have given. To compare two people with each other, we can use the file compute_scores.py which will include the corresponding Python code. We can store the input data for the users and their ratings in json format in the file ratings.json (see Appendix C.1).

In this Python file (compute_scores.py), we define the following functions:

- *build_arg_parser()*, the argument parser to process the input arguments (user1, user2 and similarity measure to be used as Euclidean or Pearson), see Appendix C.2;
- *euclidean_score(dataset, user1, user2)*, the function to calculate the Euclidean score between the input users in the dataset, see Appendix C.3;
- *pearson_score(dataset, user1, user2)*, the function to calculate the Pearson score be-tween the input users in the dataset, see Appendix C.4 ;

- the main function, which parses the input arguments, loads the user preferences from the ratings.json file into a dictionary and calculates and returns the similarity score, see Appendix C.5.

Finally, for example, to calculate the Euclidean score between David Smith and Bill Duffy we can execute in command line the following command:

compute_scores.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

Also, in order to calculate the Pearson score between Chris Duncan and Bill Duffy we can issue the following command:

compute_scores.py --user1 "Chris Duncan" --user2 "Bill Duffy" --score-type Pearson

3.7.4 Discovering similar users using collaborative filtering

Collaborative filtering is the process of recognizing patterns between objects in a data set in order to make a decision about a new object. In the context of recommendation engines, collaborative filtering can provide recommendations by examining similar users in the data set. Looking at the previous example, we can assume that people with similar preferences for a certain set of movies will have similar choices in a set of new unknown movies. We can use the same scoring techniques we have seen so far to find similar users in our data set. These methods can also be used in finance, e-commerce, marketing, customer studies, and so on.

A code example is provided in the collaborative_filtering.py file in which we define the following functions:

- *build_arg_parser()*, a function to parse the input argument user name, (see Appendix C.6);
- *find_similar_users(dataset, user, num_users)*, a function to find users in the dataset that are similar to the given user, (see Appendix C.7);
- The Pearson score function was imported from the previous file to calculate the Pearson score between the input user and all other users in the dataset, (see Appendix C.6);

- The main function parses the input arguments, loads the data from the movie preferences file ratings.json, and returns the top three users who are similar to the user specified by the input argument, (see Appendix C.8).

To run the algorithm and find out the users who have similar preferences to Bill Duffy for example, execute the following command:

collaborative_filtering.py --user "Bill Duffy"

3.7.5 Predicting Users' Preferences using Similarity Scores Application

At this stage, our goal is to create a movie recommendation system based on the similarity of movie choices made by different users. With all the building blocks in place (KNN, similarity scores, collective filtering), we can now create a movie recommendation system. The file ratings.json contains a set of people and their ratings for various movies. To find movie recommendations for a particular user, we need to find similar users in the dataset and make recommendations for that person based on the movies that the other similar users have seen and our user hasn't.

The file movie_recommender.py provides the Python code for such a system and has the following structure:

- *build_arg_parser()*, a function to parse input arguments (user name), (see Appendix C.9);
- *get_recommendations(dataset, input_user)*, a function to get movie recommendations for a given user, (see Appendix C.10);
- The main function main parses the input arguments, loads the movie ratings data, and outputs and prints the movie suggestions, (see Appendix C.11).

In more details, in *get_recommendations()* function we calculate a similarity score between the input user and all other users in the dataset. Then, we extract a list of movies that have been rated by each user but not rated by the input user. Also, for each item in the filtered list, we track the score weighted by the similarity score. We normalize the scores based on the weighted scores. Last, we sort the ratings and extract the movie recommendations.

We can execute the algorithm as follows (e.g. to find out movie suggestions for Chris Duncan):

movie_recommender.py --user "Chris Duncan"

3.7.6 Benefits and Ethical Considerations of Predicting Users' Preferences

Effective user preference prediction offers significant benefits for both users and service providers:

- Improved User Experience;
- Increased User Engagement;
- Higher Retention and Conversion Rates;
- Operational Efficiency;
- Competitive Advantage.

Despite these advantages, predicting user preferences comes with challenges:

- Privacy Concerns;
- Algorithmic Bias;
- Over-Personalization;
- Transparency and Control.

Last but not least, users' account preferences predictions are a powerful tool for delivering personalized, meaningful experiences in digital services. By leveraging data and advanced analytics, businesses can anticipate user needs, improve engagement, and drive growth. However, the field must balance innovation with ethical considerations, prioritizing privacy, fairness, and user agency. As technology evolves, the future of user preference prediction will depend on building systems that are not only intelligent but also transparent, respectful, and aligned with user values.

3.7.7 Conclusions

This study has comprehensively explored the design, implementation, and evaluation of Artificial Intelligence Neural Networks for the purpose of detecting and predicting user identities and preferences. Leveraging a user's historical profile data and behavioral patterns, we demonstrated that neural network based approaches, with both single layer

and multi layer architectures can provide substantial accuracy in recognizing user identity even without traditional authentication mechanisms like passwords or cookies. Furthermore, we validated models that can effectively predict future user preferences in real-time.

Throughout the research, we examined a variety of neural network types and ma-chine learning algorithms including supervised and unsupervised learning models, multi-layer Perceptrons, and K-nearest neighbors (KNN) techniques. These methods were applied in simulated and structured users' data with strong emphasis on behavior-based recognition. Our findings suggest that neural networks, particularly when they are well optimized and trained on clean, meaningful datasets, are powerful tools for creating adaptive systems capable of offering targeted content and security intelligence at scale.

From a technical standpoint, several key strengths emerged. First, the flexibility of deep learning models in recognizing nonlinear relationships allows them to outperform traditional rule based systems. Second, the use of user behavior as an identification factor introduces a promising additional layer in multi factor authentication strategies. By analyzing choices that users can make, such as navigation patterns, time spent on specific actions and clickstream data, we can extract unique behavioral fingerprints. These fingerprints, in turn, support both identification and personalization processes, making the user experience more fluid, predictive, and context-aware.

Equally important are the practical implications of these findings. In commercial environments, such as e-commerce platforms, streaming services, and mobile applications can have the ability to anticipate user preferences or recognize returning users even in anonymized scenarios, enabling more precise product recommendations, higher user engagement, and ultimately increased conversion rates. Furthermore, behavioral authentication has strong potential in cyber security applications by helping systems detect anomalous user activity, including account takeovers, without intrusive or burdensome authentication steps. This opens the door for frictionless yet secure user experiences.

However, the adoption of such systems must also be tempered with a deep under-standing of the ethical and legal frameworks surrounding data collection and profiling. Although AI powered identification does not always require personally identifiable information, its capacity to infer sensitive characteristics through pattern recognition raises concerns around transparency, consent, and data sovereignty. As machine

learning systems grow more autonomous and predictive, it becomes critical to ensure that users remain informed and empowered regarding how their behavioral data is used.

Privacy preserving AI techniques, such as federated learning, differential privacy, and on device processing, are recommended for future systems, allowing them to benefit from AI driven personalization without compromising user anonymity. In parallel, developers must address model explainability, especially in contexts like finance, healthcare, or justice, where opaque decisions could have serious consequences. Ensuring that predictions are not only accurate but also interpretable and fair will be essential for building trust in AI enabled user identification systems.

Looking forward, the intersection of neural networks with emerging fields like neuromorphic computing, zero-shot learning, and neuro-symbolic AI presents exciting avenues for research. These advancements could lead to models that require less data to make accurate predictions, are more energy-efficient, and possess reasoning capabilities closer to human cognition. Furthermore, the fusion of behavioral biometrics with traditional security layers could drive us in a new generation of adaptive, intelligent authentication systems capable of learning and evolving with user behavior over time.

In conclusion, this chapter presents a strong foundation for the use of neural networks in the detection and prediction of user identity and preferences. While the technical results are promising and the commercial potential is vast, the responsible and ethical deployment of these technologies must remain a core priority. With balanced design and continued research, such systems could significantly transform personalization, security, and user interaction in digital environments.

4 PLATFORM IMPLEMENTATION

4.1 RESTful APIs and MySQL

4.1.1 Introduction to RESTful APIs in Android Development

In modern Android app development, RESTful APIs (Representational State Transfer APIs) [177] play a central role in enabling communication between mobile clients and backend servers. A RESTful API is a web service that follows a set of architectural principles for interacting with resources (such as users, products, or posts) over HTTP. These APIs provide endpoints through which Android apps can perform operations like creating a new account, fetching user data, updating a profile, or deleting an item from a remote database.

RESTful APIs operate on standard HTTP methods like GET, POST, PUT, DELETE, PATCH, making them easy to use and compatible with virtually any platform. In REST, resources are typically represented in JSON or XML format, although JSON has become the standard due to its simplicity and efficiency for mobile applications.

For Android developers, RESTful APIs are essential for apps that require real-time or centralized data, multi-user environments, or server-side processing. Whether building a weather app, e-commerce platform [178], social network, or content management system integrating with a RESTful API allows the app to offload complex logic and

data storage to a remote server while focusing on providing a smooth and responsive user interface.

4.1.2 How Android Apps Communicate with RESTful APIs

Android apps communicate with RESTful APIs over the internet using HTTP protocols. The communication process typically involves sending a request to an API endpoint and receiving a response from the server. This interaction is usually done through libraries such as Retrofit, Volley, OkHttp, or even Java's built-in HttpURLConnection. These libraries handle networking tasks, parsing JSON responses, and error handling, greatly simplifying the integration process.

4.1.2.1 Overview of Retrofit for REST API Integration

Retrofit [179], developed by Square, is one of the most popular libraries for REST API communication in Android. It allows developers to define interfaces with annotations that specify the HTTP method, endpoint path, and parameters. Retrofit automatically converts JSON responses into Java/Kotlin objects using converters like Gson or Moshi, making the data easy to work with.

For instance, to fetch user details, a developer might define an interface like this (Figure 4.1):

GetUserMethod

```
@GET("users/{id}")
Call<User> getUser(@Path("id") int userId);
```

Figure 4.1: Example of Retrofit usage in order to achieve Android App - Web Server Restful Api communication

When called, this sends a GET request to the endpoint (e.g., https://api.example.com/users/1) and returns a User object parsed from the JSON response. Similar approaches apply to POST requests for submitting forms, PUT for updating records, and DELETE for removing entries.

4.1.2.2 Overview of Android Volley for REST API Integration

Android Volley [179] is an HTTP networking library introduced by Google to simplify and optimize network operations in Android applications. It provides a framework for managing network requests, handling responses, and performing asynchronous

operations without writing extensive standardize code. When building Android apps that communicate with RESTful APIs (a common architectural pattern for web services), Volley can play a central role by providing mechanisms to make efficient, structured, and error-tolerant HTTP calls. Given the RESTful model's and based on HTTP methods like GET, POST, PUT, DELETE, Volley's support for these operations can make it a valuable tool for Android development.

RESTful APIs are designed to exchange data over standard HTTP protocols, often using JSON as the data format. Volley simplifies communication with such APIs by offering request classes like StringRequest, JsonObjectRequest, and JsonArrayRequest. These allow developers to send HTTP requests to API endpoints and receive structured responses without managing low-level network details. For example, when an app needs to fetch a list of user profiles from a server, a GET request via JsonArrayRequest can retrieve the data asynchronously, and the response can be parsed directly within the callback.

Volley's RequestQueue system handles the queuing and execution of these network requests efficiently, supporting caching and threading out of the box. This means developers don't have to manage background threads manually. Additionally, Volley offers automatic scheduling of requests based on priority, as well as retry and timeout policies, which enhance the reliability of RESTful communication in environments with unstable network conditions.

4.1.2.3 Volley's Comparison, Strengths and Limitations

From an analytical standpoint, Volley excels in scenarios that involve rapid, small to medium sized data exchanges, such as syncing app content, retrieving JSON lists, or submitting form data [179]. Its lightweight architecture and ease of use makes it ideal for apps that need fast, responsive interactions with REST APIs but don't require complex data modeling or long lived connections.

However, Volley shows some limitations compared to more modern alternatives like Retrofit. One notable drawback is its lack of built in support for object mapping! JSON responses must be manually parsed or integrated with a library like Gson. This adds complexity when dealing with nested or large data structures. In contrast, Retrofit natively integrates with Gson or Moshi to automatically map JSON responses into Java or Kotlin models.

Moreover, Volley is not optimized for large file transfers or streaming media. Use cases involving file uploads, downloads, or long-lived HTTP connections are better handled by OkHttp or Retrofit, which offer more robust and scalable solutions. Despite this, Volley still performs well for REST operations that are lightweight and short-lived, especially when minimal dependencies are preferred.

4.1.2.4 When to Use Volley with REST APIs

Android Volley remains a relevant and capable option for integrating RESTful APIs into Android apps, particularly when the primary goals are simplicity, speed, and minimal dependencies. Its strong support for asynchronous operations, built-in caching, and retry mechanisms makes it well-suited for apps that frequently interact with remote servers but don't require advanced features like automatic serialization.

For developers, building apps that rely on clean, frequent API calls such as messaging apps, weather apps, or news feeds, Volley provides a reliable solution that balances ease of use with performance. However, for projects requiring scalable data models, advanced serialization, or heavy file handling, a combination of Volley with Gson, or a switch to Retrofit, may offer greater long term flexibility and maintainability.

4.1.3 Authentication and Security in RESTful API Integration

Security is a vital concern when interacting with RESTful APIs in Android apps. Because data is transmitted over the internet, it must be protected from interception, tampering, or misuse. The most fundamental practice is to use HTTPS (SSL/TLS encryption) for all communication between the app and the server. This ensures that data remains encrypted and secure during transmission.

Handling the server response and errors (like 404 or 500 codes) is also critical. Most libraries include callback mechanisms or coroutine support to manage asynchronous tasks. By integrating with RESTful APIs, Android apps gain access to live data and server logic, enabling powerful features like user authentication, file uploads, location-based services, and remote configuration.

Authentication is typically handled using API keys, OAuth tokens, or JWT (JSON Web Tokens). In mobile apps, once a user logs in with valid credentials, the server returns a token, which the app stores securely (usually in SharedPreferences or EncryptedSharedPreferences). This token is then sent with subsequent API requests in the HTTP header to identify and authorize the user.

Using this model, RESTful APIs remain stateless. Each request contains all the information needed to process it, which is crucial for scalability and simplicity. However, developers must ensure that tokens are stored securely and not exposed, especially in decompiled APKs. Tools like ProGuard and encrypted storage APIs help protect sensitive information.

Additionally, rate limiting, CORS policies, and access controls should be enforced server-side to protect against abuse. On the Android side, input validation, error handling, and session management help ensure the app responds securely and gracefully to API responses and failures.

4.1.4 Real-World Use Cases of RESTful APIs in Android Apps

RESTful APIs are widely used in real-world Android applications across various industries and functions. In e-commerce apps, REST APIs connect the frontend with product catalogs, payment systems, user accounts, and order histories. When a user browses products or places an order, the app communicates with the backend through REST endpoints to retrieve data or send transaction details.

In social media applications, APIs manage user profiles, posts, comments, likes, and messaging. For example, posting a photo involves sending a POST request with image data and metadata, while fetching a feed might use a GET request to retrieve a list of recent posts. RESTful APIs make it easy to scale these interactions across millions of users and platforms.

In weather apps, REST APIs fetch real-time weather forecasts, alerts, and climate data from external services like OpenWeatherMap or WeatherStack. The Android app parses and displays the information dynamically based on the user’s location or preferences.
Travel and booking apps rely on RESTful APIs to check availability, confirm bookings, show travel planned routes, and display maps or points of interest. These APIs are often connected to multiple backend systems, and the Android app acts as a unifying interface.

Educational, healthcare, finance, logistics, and entertainment apps also depend on RESTful APIs for tasks like managing content, syncing with cloud databases, handling secure communication, and integrating with third-party services. The modular and standardized nature of REST makes it ideal for almost any use case where mobile apps need to interact with remote resources.

4.1.5 Best Practices for Using RESTful APIs in Android

To build reliable, secure, and maintainable Android apps using RESTful APIs, developers should follow key best practices in both code design and communication architecture [180,181]. Always use asynchronous calls when interacting with APIs to avoid blocking the UI thread. Libraries support asynchronous requests using callbacks or Kotlin coroutines, which ensure smooth user experiences and prevent Application Not Responding (ANR) errors. Also, implement caching and offline strategies where appropriate. For example, when displaying news or products, the app can store the latest API response in local storage (using e.g. SQLite), allowing the user to access content even when offline. This also reduces server load and data usage. Moreover, handle errors and exceptions gracefully. Check for common HTTP status codes like 200 (OK), 401 (unauthorized), 404 (not found), and 500 (server error), and provide meaningful feedback to the user. Also, always validate and sanitize inputs before sending them to the server to avoid injection attacks.

Next, use a base URL and centralized network client to simplify code reuse and maintenance. Also, consider versioning your API. As apps evolve, changes to endpoints may break backward compatibility. By versioning endpoints (e.g. /api/v1/users), developers can support old and new versions simultaneously without disrupting users.

Finally, keep your API usage secure and minimal. Avoid over-fetching by requesting only the fields you need, paginate large data sets using parameters like limit or offset, and don’t forget to implement logging tools like Stetho or OkHttp Logging Interceptor during development to monitor API calls and debug efficiently.

In conclusion, RESTful APIs are a cornerstone of Android development, enabling apps to connect with powerful backend systems and deliver rich, data-driven experiences. By understanding how they work and applying thoughtful design principles, developers can create scalable, secure, and high-performance Android applications.

4.1.6 MySQL and Its Relevance to Android

MySQL is an open-source relational database management system (RDBMS) widely used for server side data storage in web and mobile applications. Known for its reliability, scalability, and robust query capabilities, MySQL is often the database of choice for backend systems, especially those built with PHP, Node.js, or Python. While Android devices typically use SQLite for local data storage, many apps need to interact with remote servers where MySQL is used as the central database.

In the context of Android, MySQL serves as a backend database, the place where structured data is stored, updated, and retrieved through a network connection. For example, in a mobile app for an online store, user information, product listings, order history, and transaction data are typically stored on a remote MySQL server. The Android app acts as the frontend client, communicating with this server to request or send data.

This communication doesn't happen directly between Android and MySQL. Instead, it occurs through a web API (Application Programming Interface), usually written in PHP, Python (Flask or Django), or Node.js. These APIs act as a bridge, accepting requests from the Android app (usually via HTTP/HTTPS), interacting with the MySQL database, and returning results in a format like JSON. This architecture ensures security, scalability, and maintainability, while also separating concerns between the frontend and backend.

4.1.7 Integrating MySQL with Android via Web Services

Direct access to a MySQL database from an Android app is not recommended due to security and architectural concerns. Instead, Android apps interact with MySQL through RESTful web services. These services are hosted on a web server and act as intermediaries between the mobile client and the database. This pattern ensures that the database remains protected behind a firewall and that the business logic is centralized and controlled.

To implement this setup, developers typically create a web API using a server-side language like PHP. Each endpoint of the API handles a specific task, such as inserting data, updating records, or fetching details. For instance, a getUserData.php script might receive a user ID via a POST request, query the MySQL database, and return the corresponding user information as a JSON object. The Android app sends an HTTP request to this URL and then parses the JSON response using libraries like Volley, Retrofit, or OkHttp.

Here is a simple example of how Android might fetch data from a MySQL database:

1. The user opens the app and logs in.
2. Android sends an HTTP POST request to a PHP endpoint (e.g., login.php) with username and password.
3. The PHP script validates the credentials against the MySQL database.

4. The server responds with success/failure in JSON format.
5. The Android app parses the JSON and proceeds accordingly.

Using web services provides several benefits. It decouples the frontend from the database logic, improves scalability, allows centralized data validation and protects sensitive information. REST APIs can also be reused across multiple platforms (e.g., Android, iOS, web), promoting consistency and reducing development time.

4.1.8 Android and MySQL Use Cases, Benefits, and Best Practices

The combination of MySQL and Android is ideal for applications that require centralized data storage and multi-user access. Examples include e-commerce apps, social media platforms, hotel booking systems, educational tools, and employee management apps. In all of these, MySQL acts as the central repository of information, while Android apps provide the interface for users to view, modify, or contribute to that data in real time.

One of the primary advantages of using MySQL with Android is data consistency. Since the database is hosted on a remote server, updates made by one user are instantly reflected for others. This is crucial in collaborative apps where data changes frequently, such as chat applications or inventory systems. Furthermore, MySQL's support for advanced queries, stored procedures, and indexing makes it efficient for handling complex operations like filtering, sorting, and data aggregation.

Another advantage is data backup and recovery. Centralized MySQL databases can be backed up regularly and restored in case of failure, reducing the risk of data loss. With local storage (e.g., SQLite), such protections require additional manual implementation.

However, developers must also consider some challenges. One is network dependency. Since data is stored remotely, users need internet access to use the app fully. To address this, developers often implement local caching or offline modes that temporarily store data on the device and sync with the server when a connection becomes available.

Security is another critical aspect. Exposing database logic or credentials to the client side is a major vulnerability. Developers should ensure that communication between Android and the server is encrypted (using HTTPS), that APIs are protected by authentication mechanisms (like tokens), and that inputs are validated to prevent SQL injection or other attacks.

To optimize performance and security when using MySQL with Android:

- Use prepared statements in server-side scripts.
- Limit data transfer by designing efficient endpoints (e.g., paginate results).
- Store only essential data locally to reduce memory usage.
- Implement proper error handling for network failures.
- Regularly test APIs and monitor server logs for anomalies.

In conclusion, MySQL remains a powerful tool for Android developers building apps that rely on centralized data management. By using web APIs as an intermediary, Android apps can access the full capabilities of MySQL while maintaining a secure and maintainable architecture. With thoughtful design and best practices, developers can create responsive, robust, and scalable mobile applications that seamlessly integrate with MySQL databases.

4.1.9 MySQL & RESTful API, Android Login and Registration with PHP, MySQL, and SQLite

Implementing login and registration in Android often involves combining local storage with a remote backend. A common stack for this is PHP + MySQL for the server side, and SQLite for local persistence on the Android device. This setup allows users to authenticate securely while also enabling offline access.

4.1.10 System Architecture

- **Android App (Client):**
  - Provides the user interface for login and registration.
  - Sends requests (via HTTP/JSON) to a PHP backend.
  - Uses **SQLite** to store user sessions or cached data locally for offline mode.
- **PHP Backend (Server):**
  - Receives HTTP requests from the Android app.
  - Processes login and registration logic.
  - Interacts with **MySQL** to store and verify user data.
- **MySQL Database (Remote):**
  - Stores registered user credentials (username, email, password hash).

- Supports authentication and data persistence.

The architecture for implementing login and registration in Android using PHP, MySQL, and SQLite is built on the interaction of three distinct but interconnected layers: the Android client, the PHP backend, and the Database Systems (MySQL and SQLite). Each component fulfills a specific role, and the system's strength lies in the way they collaborate to provide both online functionality and offline persistence. By separating concerns across these layers, the architecture achieves flexibility, scalability, and reliability.

At the client side, the Android application is the entry point for users. It provides the graphical user interface that allows individuals to input their credentials, initiate login, or complete the registration process. More importantly, the Android app is not just a passive front end. It takes an active role in managing communication with the server and in ensuring smooth user experience. For example, the application packages data into HTTP requests, typically in JSON format, before sending them to the backend. It also receives responses from the server, parses them, and presents meaningful results to the user. Beyond this transactional role, the Android app integrates with SQLite, which serves as a lightweight, embedded database. SQLite is critical for scenarios where internet connectivity is unstable or temporarily unavailable, because it allows the app to cache user sessions or store limited amounts of profile data locally. This design improves usability by supporting offline access and quick retrieval of data without constant server queries.

The PHP backend functions as the intermediary between the Android client and the remote MySQL database. It acts as a logical bridge, processing requests from the app, validating inputs, and applying the appropriate business rules. In the registration process, PHP scripts ensure that user inputs are sanitized to avoid malicious injections, check whether email addresses are already in use, and hash passwords for secure storage. During login, PHP compares the submitted credentials against hashed records in the MySQL database, ensuring both security and efficiency. By handling this logic at the server level, the architecture ensures that sensitive operations never occur on the client device, where security risks would be higher. PHP is also chosen for its simplicity and compatibility. Since it integrates seamlessly with MySQL and it is widely supported on affordable hosting platforms, it makes deployment accessible for developers.

The MySQL database is the backbone of the system's data storage. It holds the master records of all registered users, including hashed passwords, timestamps, and identifying details. Unlike SQLite, which is confined to the client device, MySQL is centralized, making it accessible across devices and ensuring data persistence beyond a single phone. This centralization is essential for scenarios where users need to log in from multiple devices or when the application needs to track usage data at scale. MySQL's relational structure ensures consistency and integrity through features such as primary keys, constraints, and indexing, which help maintain the reliability of the system as it scales.

The dual use of MySQL and SQLite in this architecture reflects a hybrid approach that balances global accessibility with local resilience. MySQL guarantees a central repository totally aware of every action of the users, where user data is synchronized and managed securely, while SQLite supports offline functionality, reducing dependence on constant internet connectivity. Together, these two databases allow the system to respond dynamically to different network conditions and user expectations. This hybrid approach is particularly beneficial in regions with intermittent connectivity, where maintaining user sessions locally can prevent frustration and encourage continued app usage.

Another significant aspect of this architecture is its modularity. Each component (Android, PHP, MySQL, and SQLite) can evolve independently. Developers may upgrade the Android client with new features without restructuring the server-side logic. Similarly, the MySQL schema can expand to support additional data types, while SQLite can be customized to manage different caching strategies. This separation also strengthens system security. Even if the local SQLite database is compromised, the sensitive operations and true records remain safeguarded in MySQL under PHP's protective logic.

The system architecture underpinning Android Login and Registration with PHP, MySQL, and SQLite demonstrates a well-balanced, layered design. It unites the strengths of centralized and local databases, leverages the simplicity and ubiquity of PHP, and provides a responsive Android client. The architecture not only ensures data security and consistency but also enhances the user experience through offline availability and fast performance. By combining these elements, the system achieves

both robustness and accessibility, making it a practical solution for real-world mobile applications.

4.1.10.1 System Workflow

To interact with MySQL database we need to build a REST API first. REST Api job is to get the request from client, interact with database and finally give the response back to client. So we'll create a simple PHP, MySQL API first as shown in Figure 4.2. Our API do's below jobs.

⇒ Accepts requests in GET/POST methods.

⇒ Interact with database by inserting / fetching data.

⇒ Finally, will give response back in JSON format.

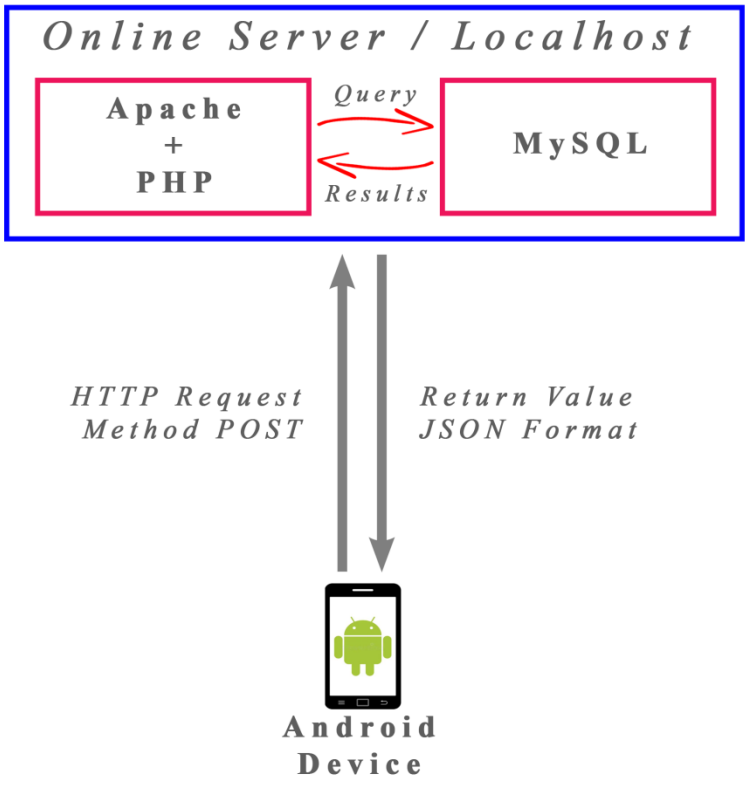


Figure 4.2: Android Device to MySQL Server Communication

Registration Flow

1. User enters details (username, email, password) in the Android app.
2. App sends data via **HTTP POST** to a PHP API endpoint (e.g., `register.php`).

3. PHP script validates inputs, checks for existing users, and inserts new records into MySQL.
4. PHP returns a **JSON response** indicating success or failure.
5. Android app displays a confirmation and optionally stores the user in **SQLite** for local access.

Login Flow

1. User inputs email and password in the app.
2. App sends credentials via **HTTP POST** to a PHP API endpoint (e.g., `login.php`).
3. PHP script queries MySQL to verify the user and password hash.
4. If valid, PHP returns user details as JSON.
5. Android app saves the user session/token in **SQLite** for persistent login.

The workflow of login and registration in an Android system supported by PHP, MySQL, and SQLite highlights how client-server interactions are organized into sequential steps to ensure both functionality and security. At its core, the workflow embodies two essential processes. The registration and login (design is shown in Figure 4.3), which are distinct yet interdependent. Together, they form the foundation of user authentication, enabling applications to identify users, safeguard data, and provide personalized access. Analyzing this workflow reveals not only the technical steps but also the underlying principles that make the system resilient, user-friendly, and adaptable.

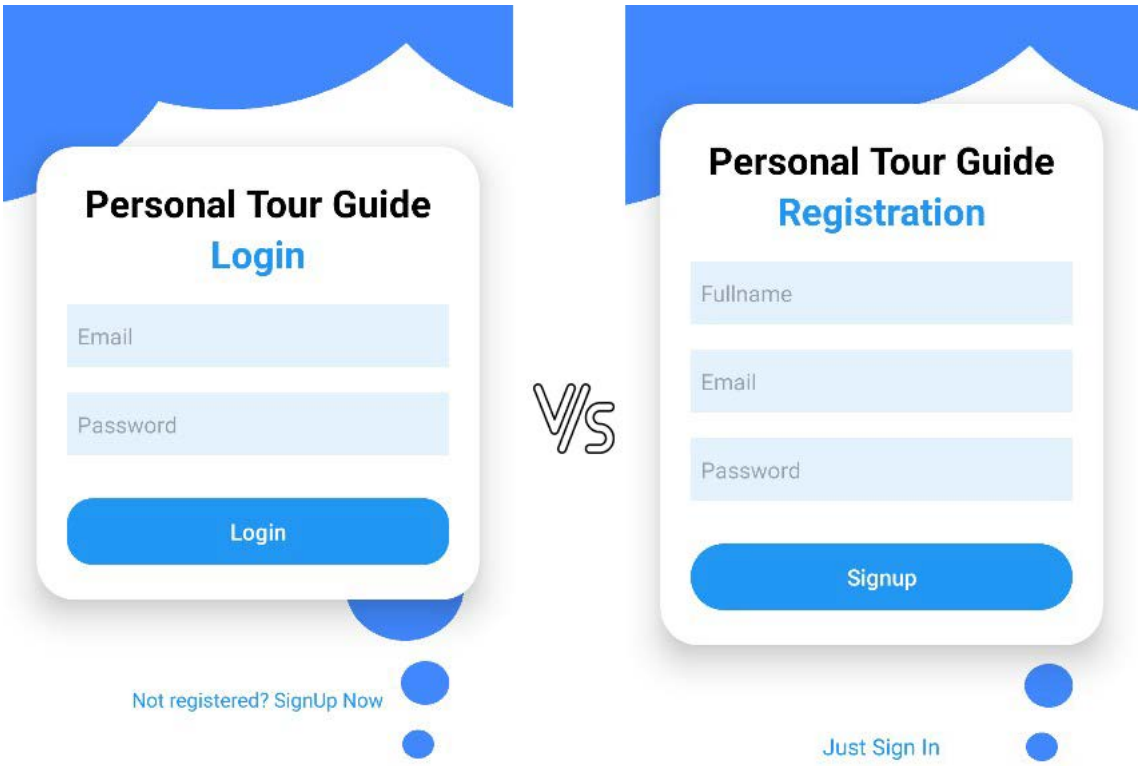


Figure 4.3: Application's Login and Registration Window

In the registration flow, the system's primary goal is to establish a new identity for the user within the database. This begins when the user enters their details into the Android application, usually consisting of a username, email address, and password. The Android client plays a significant role in packaging this data and transmitting it to the server, typically through an HTTP POST request (using Android Volley Library) directed to a PHP endpoint such as register.php. The transmission of this data represents the first critical point of vulnerability, making secure handling through methods like input validation and encryption crucial. Once received, the PHP backend validates the data, ensuring that essential fields are not empty, there is not an attempt for code injection and that the email address is not already associated with an existing account. If these checks are satisfied, PHP proceeds to hash the password using secured algorithms like bcrypt or Argon2, which prevent attackers from retrieving plain-text credentials even if the database is compromised. The PHP script then inserts the user's information into the MySQL database and responds with a JSON message indicating success or failure. On the Android side, the application interprets this response, informs the user of

the outcome, and optionally stores essential session data in SQLite. This structured workflow ensures that new users are added securely and seamlessly.

The login flow, while similar in its reliance on client-server interaction, serves a different purpose, verifying an existing user's identity. The user initiates this process by entering their email and password into the Android interface. As with registration, the application sends the credentials to the PHP backend via an HTTP POST request, often targeting an endpoint such as login.php. The PHP script then retrieves the corresponding record from the MySQL database and verifies the submitted password against the stored hash. If the verification succeeds, the backend generates a response containing user details, and in some cases, a session token to maintain authentication across activities or subsequent requests. This tokenized approach enhances security by preventing the need to transmit plain credentials repeatedly. Once the Android app receives the JSON response, it updates its local SQLite database with session or token information, allowing the user to remain logged in even when offline. In this way, the login workflow strikes a balance between server-side validation and client-side persistence, ensuring both security and convenience.

The interplay between these workflows highlights the system's capacity to integrate online and offline functionality. Registration necessarily requires a live connection to MySQL, as new user records must be inserted into the central database. Login, however, can extend beyond online verification. By storing user session data locally in SQLite, the application allows users to continue accessing the system even when disconnected from the internet. This design acknowledges real-world conditions, such as intermittent connectivity, and reflects a user-centered approach. It prevents frustration by ensuring that authentication is not exclusively dependent on constant network access. At the same time, the architecture ensures that SQLite's local role is supplementary rather than authoritative, with MySQL remaining the ultimate source of truth.

Another important aspect of the workflow is its reliance on JSON-based communication between client and server. JSON is lightweight, easy to parse, and universally supported, making it well-suited for mobile applications. Its inclusion in the workflow facilitates smooth data exchange while reducing overhead compared to more complex formats like XML. Additionally, the use of clear status messages such as "success" or "error" provides transparency to the client and ensures that errors are communicated

effectively. This feedback loop is essential not only for debugging and development but also for enhancing the user experience by offering clear guidance when login or registration attempts fail.

From a security perspective, the workflow incorporates multiple layers of protection. On the client side, sensitive data like passwords should ideally be transmitted over HTTPS to prevent interception. On the server side, PHP scripts apply sanitization to prevent SQL injection and rely on password hashing to guard against credential theft. The database enforces constraints to ensure uniqueness of user accounts, while session tokens mitigate risks associated with replaying credentials. Together, these measures demonstrate how workflow design intertwines with security requirements, making the authentication process robust against common threats.

In summary, the workflow of registration and login in Android applications using PHP, MySQL, and SQLite demonstrates a carefully structured process that balances usability, reliability, and security. Registration workflows focus on establishing user identities in a secure and centralized manner, while login workflows emphasize verification, session persistence, and offline accessibility. The Android client, PHP backend, and databases work in harmony, with each step carefully sequenced to maximize efficiency and minimize vulnerabilities. This workflow not only ensures that users can reliably create and access accounts but also reflects broader design principles of modern mobile development like user-centered design, hybrid online-offline support, and layered security. By analyzing the workflow in detail, it becomes clear that its structure is not arbitrary but deliberately engineered to align technical requirements with real-world usage scenarios.

4.1.10.2 MySQL Database Structure

The database structure in a login and registration system built with Android, PHP, MySQL, and SQLite represents the backbone of user authentication and data management. At its core, the design revolves around two complementary databases. MySQL on the server side and SQLite on the client side. Each serves a distinct purpose, and together they establish a hybrid model that ensures both centralized control and localized accessibility.

The MySQL database is the authoritative storage system for user accounts. Its relational design provides structure and integrity, with the users table typically including fields such as an auto-incrementing primary key (id), a unique email, a password hash, and

timestamps for record creation. This schema (shown in Figure 4.4), not only guarantees uniqueness and prevents duplicate accounts but also supports scalability as the application grows. By hashing passwords before storing them, MySQL ensures that sensitive information is never preserved in plain text, aligning the structure with modern security best practices. Furthermore, relational constraints help maintain consistency across records, ensuring that every registered user is represented accurately within the central repository.

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    password VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Figure 4.4: MySQL Table (users)

On the client side, SQLite plays a different but equally important role. Unlike MySQL, it is lightweight, file-based, and embedded directly within the Android device. The local_users table is designed to store essential session details such as the user's name, email, or session token, rather than serving as a complete mirror of the remote database. This local storage supports offline persistence, allowing users to remain logged in or access cached information even without an active internet connection. The design (shown in Figure 4.5) is intentionally minimalist, reducing the risk of data duplication or inconsistency while still providing functionality that improves user experience.

```
CREATE TABLE local_users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    email TEXT,
    session_token TEXT
);
```

Figure 4.5: SQLite Table (local_users)

The coexistence of these two databases highlights a thoughtful architectural decision. MySQL ensures that all accounts and credentials remain centralized and universally accessible, making it the single source of truth. SQLite complements this by delivering convenience and responsiveness, especially in environments with unreliable connectivity. The structural distinction between the two of them, one relational and global, the other lightweight and local creates a system that balances robustness with usability. In essence, the database structure is not just a technical implementation detail but a deliberate design strategy that reinforces the system's goals of security, accessibility, and efficiency.

4.1.10.3 The PHP API

The PHP APIs provided for registration and login are central to the interaction between the Android client and the MySQL database. These scripts function as the gateway through which data flows, transforming user input into secure database transactions and returning structured responses that the Android app can interpret. Their simplicity is intentional, making them easy to understand and deploy, but beneath this simplicity lies careful logic that ensures both security and reliability in handling sensitive operations. The hierarchy of the server side files is described in the Figure 4.6. The overall system was hosted under XAMPP software bundle version 3.2.4 for Windows 10 in localhost, including PHP Version 7.4.7 and database server 10.4.13-MariaDB. Nevertheless, the system is also fully tested in real/commercial online Apache web server, using shared web hosting environment.

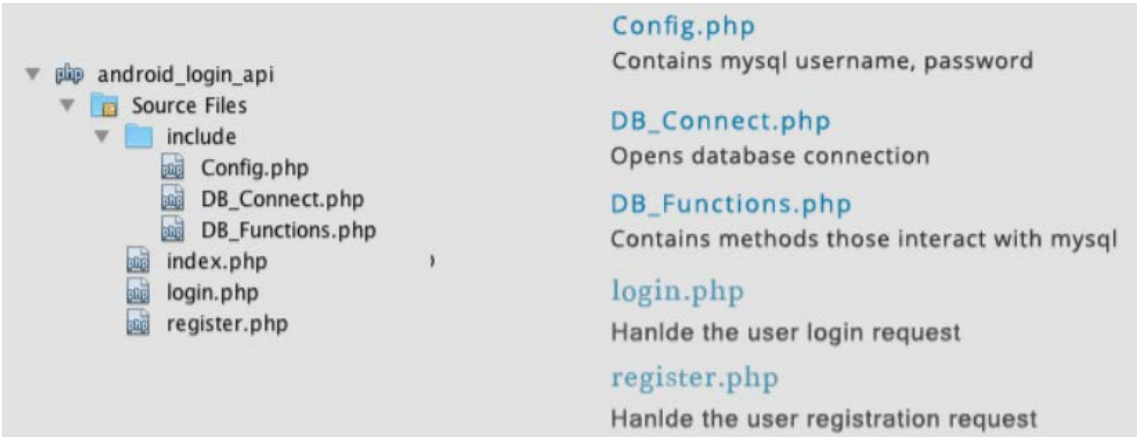


Figure 4.6: Web Platform Hierarchy

In the registration script, register.php shown in Figure 4.7, the workflow begins with capturing user input name, email, and password sent via POST from the Android application. The script then hashes the password using PHP's password_hash() function before inserting the record into the MySQL database. This step is critical because it ensures that raw passwords are never stored, protecting user data even if the database is compromised. By employing prepared statements ($stmt->bind_param), the script guards against SQL injection, which is one of the most common security vulnerabilities in web applications. The response is encoded in JSON, returning either a success message or an error, thereby standardizing communication with the Android app. This design demonstrates a strong adherence to modern web security practices while maintaining straightforward implementation.

```php
<?php
include 'db_connect.php'; // MySQL connection

$name = $_POST['name'];
$email = $_POST['email'];
$password = password_hash($_POST['password'], PASSWORD_DEFAULT);

$stmt = $conn->prepare("INSERT INTO users (name, email, password) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $name, $email, $password);

if ($stmt->execute()) {
    echo json_encode(["status" => "success", "message" => "User registered"]);
} else {
    echo json_encode(["status" => "error", "message" => "Registration failed"]);
}
?>
```

Figure 4.7: Content of the register.php file

The login script, login.php shown in Figure 4.8, plays an equally vital role but focuses on verification rather than storage. It retrieves the stored hash corresponding to the provided email and uses password_verify() to compare it with the submitted password. This ensures that validation is performed using cryptographic checks instead of simple string matching, which would be highly insecure. Upon successful verification, the script responds with a JSON object containing the user's details, allowing the Android client to establish a session. In cases of failure, a clear error message is returned,

enabling the app to provide immediate feedback to the user. This flow illustrates how the script balances user experience with backend responsibility, ensuring that authentication is both efficient and secure.

```php
<?php
include 'db_connect.php';

$email = $_POST['email'];
$password = $_POST['password'];

$stmt = $conn->prepare("SELECT * FROM users WHERE email=?");
$stmt->bind_param("s", $email);
$stmt->execute();
$result = $stmt->get_result();
$user = $result->fetch_assoc();

if ($user && password_verify($password, $user['password'])) {
    echo json_encode(["status" => "success", "user" => $user]);
} else {
    echo json_encode(["status" => "error", "message" => "Invalid login"]);
}
?>
```

Figure 4.8: Content of the login.php file

Overall, the PHP APIs highlight a minimal yet robust implementation of server-side authentication logic. They embody three key principles: security through hashing and prepared statements, clarity through JSON-based responses, and efficiency by delegating heavy operations to the server while keeping the client lightweight. While these scripts can be extended with advanced features like token-based authentication or error logging, even in their basic form they provide a solid foundation for integrating Android applications with a MySQL backend. Their design reflects a pragmatic balance between accessibility for developers and protection for end users.

4.1.10.4 Android Side

The Android side of the login and registration system represents the user-facing layer where functionality meets experience. Its role is not only to collect input and display results but also to mediate communication with the backend and manage local persistence. The architecture ensures that the Android client remains lightweight yet

capable of handling essential logic, particularly in parsing JSON responses and interacting with SQLite for session management. By leveraging libraries such as Volley, the Android app simplifies network communication, allowing developers to send HTTP requests and handle asynchronous responses without extensive boilerplate code. This abstraction is significant because it reduces complexity while improving reliability, ensuring that user data is transmitted securely and responses are processed efficiently. The overall Android side workflow contains the following:

- Use **Volley** to send requests to the PHP backend.
- Parse the JSON response.
- Store login state in **SQLite** for offline persistence.

Android application setup and configuration can be handled quickly and easily since we have developed a single file holding all the necessary information for the server's IP and all required secondary URLs for our application to be able to perform its functioning as shown in the Figure 4.9.

```
public class AppConfig {
    public static String SERVER_IP = "http://192.168.1.10/";
    // Server user login url
    public static String URL_LOGIN = SERVER_IP+"personaltourguide/login.php";
    // Server user register url
    public static String URL_REGISTER = SERVER_IP+"personaltourguide/register.php";
    public static String URL_IMAGE_HOST = SERVER_IP+"personaltourguide/images/";
}
```

Figure 4.9: Content of the AppConfig.class file

In order Android App to be able to handle HTTP requests using Volley external Library we must inform build.gradle file for our intentions as shown in the Figure 4.10 and we must update AndroidManifest.xml file for our intention to use Internet by adding the tag uses-permission with value "android.permission.INTERNET".

```
build.gradle
dependencies {
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:design:23.1.1'
    compile 'com.mcxiaoke.volley:library-aar:1.0.0'
}
```

Figure 4.10: Gradle Dependencies for HTTP Requests

An essential feature of the Android side is its ability to interpret and act upon the JSON data received from the PHP backend. Once a login request is validated and user details are returned, the app must store this information locally for persistent sessions. This is where SQLite becomes critical since it allows the app to cache credentials, session tokens, or user profiles, enabling offline access and improving load times by reducing repeated server calls. The local database is intentionally lightweight and tailored to the specific needs of the authentication system, ensuring that it supports responsiveness without overwhelming device resources.

Beyond data handling, the Android side also embodies usability and security considerations. For instance, error messages from failed login attempts are displayed to guide users, while session tokens retrieved from the backend are stored securely to prevent unauthorized access. By managing the state between online validation through PHP/MySQL and offline persistence through SQLite, the Android client ensures continuity of experience. This design reflects the hybrid nature of modern mobile applications, which must balance connectivity with resilience.

The Android side is not a passive consumer of backend services but an active participant in authentication workflows. Through libraries like Volley, local storage via SQLite, and thoughtful session management, bridges the gap between user interaction and server logic. Its effectiveness lies in its ability to combine usability, responsiveness, and security, making it an indispensable layer in the overall login and registration system.

4.1.10.5 Benefits of this Approach

The main benefits of this architecture are:

- **Scalability:** MySQL ensures data is centralized and accessible across devices.
- **Security:** Passwords are hashed before storage, preventing leaks.

- **Offline Support:** SQLite allows local storage of user data and sessions.
- **Flexibility:** PHP backend is lightweight and widely supported.

The combined use of Android, PHP, MySQL, and SQLite for login and registration provides a balanced approach that addresses both technical demands and user expectations. One of the most significant benefits is scalability. With MySQL as the centralized database, the system can grow to accommodate thousands of users while maintaining data integrity and performance. This makes the architecture suitable not only for small applications but also for projects that may expand over time. At the same time, the inclusion of SQLite on the client side ensures that the system remains responsive, reducing dependency on constant server queries and thereby easing the load on the backend.

Another major advantage is security. Password hashing in PHP ensures that sensitive credentials are never stored in plain text, minimizing risks in the event of a breach. Prepared statements protect against SQL injection, while session management using tokens prevents repeated exposure of user credentials. These measures demonstrate that even a relatively simple setup can achieve robust protection when implemented thoughtfully. In addition, the division of responsibility between client and server enhances security further as critical validation and authentication occur on the server side, while the Android client focuses on session handling and user experience.

The architecture also offers offline support, a critical feature for mobile applications operating in regions with unreliable internet connectivity. By leveraging SQLite, the app can store session tokens or limited user data locally, allowing users to remain logged in and access basic features without being online. This not only improves usability but also fosters trust, as users feel confident that their access will not be disrupted by temporary connectivity issues.

Finally, the approach provides flexibility and accessibility for developers. PHP is widely supported and easy to deploy, MySQL is a proven and robust relational database, and SQLite is natively supported in Android. Together, these technologies form a stack that is cost-effective, accessible to entry-level developers, and widely documented, which accelerates development and maintenance. In short, the benefits of

this approach lie in its ability to blend scalability, security, offline capability, and developer friendliness into a single cohesive system.

4.2 SQLite in Android

4.2.1 Introduction to SQLite in Android

SQLite [182-184] is a lightweight, embedded relational database management system that is integrated directly into Android. Unlike server-based databases like MySQL or PostgreSQL, SQLite does not require a separate server process. Instead, it reads and writes directly to ordinary disk files, making it extremely fast and efficient for use in mobile environments. Android includes a full-featured SQLite database engine that applications can use to store and manage structured data locally.

In Android development, SQLite is commonly used to store persistent data such as user settings, app configurations, offline content, logs, and more. The database is created and maintained within the internal storage of the app, making it secure and isolated from other applications. Since many mobile apps need to work offline or with intermittent connectivity, SQLite provides a reliable solution for local data storage and retrieval. Developers can use raw SQL queries or Android's built-in SQLite APIs to interact with the database.

SQLite is especially useful in cases where structured storage is preferred over flat files or shared preferences. For example, an app that manages a user's notes, tasks, or contacts can benefit from relational tables, indexing, and transactions. All these features are supported by SQLite. The ability to run complex queries, join multiple tables, and maintain data integrity makes SQLite a powerful tool for mobile developers.

4.2.2 Implementing SQLite in Android Apps

Working with SQLite in Android involves creating a database, defining tables, and performing CRUD operations (Create, Read, Update, and Delete). Android provides the *SQLiteOpenHelper* class to manage database creation and version control. This helper class simplifies the process of managing database lifecycles, including creating tables and handling upgrades when the schema changes.

To use SQLite in an Android project, a developer typically creates a class that extends *SQLiteOpenHelper* and overrides two key methods: *onCreate()* and *onUpgrade()*. The *onCreate()* method defines the SQL statements used to set up the initial schema, such as

creating tables. The *onUpgrade()* method handles changes between database versions, ensuring data migration or schema alterations are done smoothly without losing existing data.

Once the helper class is set up, a developer can open the database using *getReadableDatabase()* or *getWritableDatabase()*. These methods return a *SQLiteDatabase* object, which can be used to execute SQL statements or call convenience methods like *insert(), query(), update(), and delete()*.

An example of a common use case is a to-do list app. The app might have a tasks table with columns such as id, title, description, and timestamp. Using SQLite, the app can easily store tasks locally, retrieve them in order, and allow the user to modify or delete entries. All this can be done without requiring internet access or a remote server.

In addition to the standard SQLite API, developers can also use higher level abstractions like Room, a persistence library provided by Google as part of Android Jetpack. Room simplifies database access and reduces boilerplate code by using annotations and compile time checks. It also integrates easily with LiveData, Coroutines, and other modern Android components.

4.2.3 Advantages, Challenges, and Best Practices

One of the key advantages of SQLite in Android is its simplicity and minimal setup. Because it is embedded in the operating system, no additional configuration is required, and it does not add to the size of the apps' APKs in a significant way. SQLite is also efficient in terms of memory and CPU usage, which is critical for mobile applications. Its zero configuration nature and transactional support make it ideal for managing structured data on devices with limited resources.

Another major benefit is its reliability. SQLite supports atomic commit and rollback operations, ensuring that the database remains in a consistent state even if the device loses power or crashes during a transaction. For apps that need to store important user data or perform multiple updates at once, this level of reliability is essential.

However, SQLite is not without its limitations. It is not designed for handling very large scaled data or concurrent operations like those found in server databases. Although it supports multiple readers, it allows only one writer at a time, which can lead to performance bottlenecks in write-heavy scenarios. Also, while it supports many SQL features, it lacks advanced features like stored procedures or user-defined functions.

To make the most out of SQLite on Android, developers should follow best practices. One key practice is using parameterized queries or the built-in *ContentValues* class to prevent SQL injection attacks. Additionally, managing connections properly by opening and closing the database when needed can help avoid memory leaks or database locks. Another best practice is implementing database versioning thoughtfully. Whenever the schema changes (e.g., adding a new column or table), the *onUpgrade()* method must be updated to handle migrations without breaking existing data. Using the BEGIN TRANSACTION and END TRANSACTION statements during batch operations can also significantly improve performance and consistency. Last but not least, for apps with complex data models or requiring reactive data flows, developers may opt for the Room library, which provides compile-time verification of SQL queries, easier migration strategies, and seamless integration with the rest of the Android architecture components.

4.3 Geolocation System

4.3.1 Introducing GPS and its Evolution

The Global Positioning System (GPS) [185,186] is a satellite-based navigation system that enables users to determine their precise location anywhere on Earth. Originally developed by the United States Department of Defense for military purposes, GPS has evolved into a vital tool for civilian, commercial, and scientific applications. GPS was first conceptualized in the 1960s, inspired by the launch of Sputnik by the Soviet Union in 1957. Scientists discovered that by tracking the Doppler shift of the satellite’s radio signal, they could determine the satellite's orbit. This insight laid the groundwork for a satellite navigation system. The U.S. Department of Defense began work on the NAVSTAR GPS program in the 1970s. The first GPS satellite was launched in 1978, and by 1995, a full constellation of 24 satellites was operational, providing global coverage.

Initially, GPS signals were restricted for civilian use through a feature called Selective Availability, which intentionally degraded accuracy. This was discontinued in 2000, opening the door for widespread civilian and commercial use. Since then, GPS has become an indispensable technology used in smartphones, vehicles, agriculture, logistics, surveying, emergency response, and countless other fields.

4.3.2 Main Functionalities of GPS and Basic Components

GPS relies on a network of satellites, ground control stations, and GPS receivers. At any given time, at least 24 satellites orbit the Earth at an altitude of approximately 20,200 kilometers (12,550 miles), arranged so that at least four are visible from any point on the planet in any time. Each satellite continuously transmits a signal containing its location and the exact time the signal was sent. A GPS receiver on the ground picks up signals from multiple satellites and calculates how long each signal took to arrive. Because the signals travel at the speed of light, the receiver can use the time delay to calculate its distance from each satellite. By measuring distances from at least four satellites, the receiver uses a process called trilateration to determine its exact position in the three dimensional space (latitude, longitude, and altitude). The fourth satellite helps correct any timing errors in the receiver's internal clock. To function accurately, the system must maintain extremely precise timing. Each GPS satellite carries multiple atomic clocks that keep time within a few nanoseconds. Ground control stations monitor the satellites' health, position, and time accuracy, updating them regularly to ensure system reliability.

Summarizing, the GPS system is composed of three main segments:

- Space Segment, which includes the constellation of GPS satellites orbiting Earth. Although only 24 are needed for full coverage the extra satellites improve accuracy and reliability.
- Control Segment, which consists of a global network of ground stations that monitor and manage the satellites. The Master Control Station is located at Schriever Space Force Base in Colorado, with additional stations in Hawaii, Kwajalein Atoll, Diego Garcia, and Ascension Island. These facilities track the satellites, send commands, and perform system updates.
- User Segment, which includes all the GPS receivers and the software or devices that interpret satellite signals. Receivers range from simple devices in car navigation systems or smartphones to high-precision instruments used in surveying and scientific research.

Modern receivers often use data from multiple satellite constellations besides GPS, such as Russia's GLONASS, the EU's Galileo, and China's BeiDou, for improved accuracy and redundancy.

4.3.3 Applications of GPS

The growing integration of GPS into everyday technology has made location services a fundamental part of modern life. Some uses of GPS technology have expanded rapidly across various domains such us:

- Navigation, which is the most well-known use of GPS. It is generally used in cars, aircraft, ships, and personal devices for route planning, traffic updates, and location tracking.
- Mapping and Surveying, where GPS is used in creating detailed maps and performing land surveys with centimeter-level precision. Also, Geographic Information Systems (GIS) heavily depend on GPS data.
- Agriculture, where precision agriculture uses GPS for automated tractor guidance, soil sampling, field mapping, and crop monitoring. This increases efficiency and reduces waste.
- Emergency Response, where GPS allows emergency services to quickly locate incidents and navigate efficiently. It also plays a role in disaster management, search and rescue missions, and wildfire containment.
- Telecommunications, where GPS provides precise time synchronization necessary for mobile phone networks, financial systems, and data centers.
- Recreation and Fitness, where outdoor sports enthusiasts use GPS for hiking, biking, and running. In this case devices track distance, speed, elevation, and routes.
- Military and Defense, where GPS continues to be crucial for navigation, missile guidance, troop movement, and reconnaissance operations.

4.3.4 Limitations and Future Developments

The GPS system continues to evolve. The U.S. is currently deploying the GPS III satellites, which offer enhanced accuracy, improved resistance to jamming, and longer lifespans. Other countries are advancing their own navigation systems to reduce reliance on foreign systems. Emerging technologies such as quantum navigation, low-Earth orbit satellites, and multi-sensor integration (combining GPS with inertial navigation systems, Wi-Fi signals, and visual data) aim to provide greater accuracy and resilience, especially in environments where GPS struggles.

While GPS is powerful and widespread, it does have limitations such us:

- Signal Obstruction, where GPS signals can be blocked or weakened by tall buildings, dense forests, mountains, or indoor environments. Urban canyons can reflect signals, causing errors.
- Interference and Jamming, as GPS is vulnerable to interference from radio signals and intentional jamming. Military and civilian sectors continue to work on anti-jamming solutions even these days.
- Accuracy, as civilian GPS applications typically provide accuracy within 5 to 10 meters. Augmentation systems like WAAS (Wide Area Augmentation System) or RTK (Real-Time Kinematic) can enhance precision to sub-meter or even centimeter levels.
- Dependency, as many modern systems are heavily reliant on GPS. A large-scale failure could have widespread implications, making redundancy and backup systems essential.

In conclusion, GPS has transformed from a military navigation system to a global utility that underpins modern infrastructure, commerce, and daily life. As the technology continues to evolve, we can expect even greater precision, availability, and integration into the fabric of society.

4.3.5 GPS and Android Devices

The integration of GPS (Global Positioning System) technology into Android smartphones has revolutionized the way people interact with the world around them. Android, being the most widely used mobile operating system globally, provides a robust platform for GPS based services and applications. From basic navigation to complex location based services, the synergy between GPS and Android has transformed everyday tasks such as travel, communication, security, and commerce. The widespread availability of GPS enabled Android devices means that location aware services are now deeply embedded into daily life, often operating in the background without users even noticing.

At the heart of Android’s GPS capabilities is the device's built-in GNSS (Global Navigation Satellite System) chip. While GPS refers specifically to the U.S. satellite navigation system, modern Android phones typically support multiple systems such as

Russia’s GLONASS, Europe’s Galileo, and China’s BeiDou. This multi constellation support enhances the speed and accuracy of location fixes, especially in environments with limited sky visibility, like urban centers or dense forests. Android’s Location Services API leverages these satellite signals along with cellular networks, Wi-Fi networks, internal accelerometers and even Bluetooth beacons to provide accurate and fast positioning data even in the case that connection to satellites is not available. This combination, known as hybrid positioning, improves user experience by delivering reliable location information in real-time.

One of the most common and valuable uses of GPS on Android is for navigation and mapping. Apps like Google Maps, Waze, and HERE WeGo rely heavily on GPS to offer turn-by-turn directions, estimated travel times, real-time traffic updates, and alternative routes. These apps are integral to everyday commuting, road trips, and exploring new places. GPS also plays a key role in ride-sharing platforms such as Uber and Lyft, where drivers and passengers depend on precise location tracking for coordination and safety. Without GPS, these services would simply not function efficiently.

Beyond transportation, GPS on Android devices powers a wide array of applications in health, fitness, and recreation. Apps like Strava, Runkeeper, and Google Fit use GPS to track running or cycling routes, calculate speed, measure elevation gain, and log workout duration. For outdoor enthusiasts, GPS-enabled apps like AllTrails or Gaia GPS assist with trail navigation, topographic mapping, and offline location tracking. These capabilities empower users to explore with confidence, even in remote areas where traditional maps may not be available. GPS functionality has also enabled the rise of augmented reality games like Pokémon GO, where in-game experiences are tied directly to real-world locations.

Security and safety are other areas where GPS on Android devices offers critical benefits. Family locator apps allow parents to track the real-time location of their children. Emergency services can use GPS to locate a caller in distress, even if the person is unable to provide their exact address. Android’s Emergency Location Service (ELS) can automatically send a user’s location to first responders during a 911 call, increasing the chances of a quick and effective response. Additionally, lost or stolen phones can often be recovered using Google’s “Find My Device” feature, which uses GPS and network signals to help pinpoint the device’s last known location.

GPS also enhances Android’s functionality in commerce and marketing. Businesses use location data for geofencing, a feature that triggers specific actions when a user enters a defined geographic area. Retailers may send promotional notifications to nearby customers, while delivery apps can update customers on the real-time location of their packages or food orders. Furthermore, location based analytics allow businesses to understand customer behavior and improve services. Android developers can access powerful tools such as the Google Maps SDK and LocationManager class to create custom location-aware applications that fit a wide variety of use cases.

Despite its many advantages, GPS on Android is not without limitations. Location accuracy can be affected by physical obstructions like tall buildings or tunnels, atmospheric conditions, and the number of satellites in view. Battery consumption is another concern, as GPS is one of the more power-intensive features on a mobile device. Android addresses this by offering different modes such as "High Accuracy," "Battery Saving," and "Device Only" to allow users to balance location precision with power efficiency. In addition, Android 10 and later versions provide enhanced privacy controls, letting users manage which apps can access their location data and when. Looking ahead, the integration of GPS with Android is expected to become even more advanced. Android is already adopting support for dual-frequency GNSS, which significantly improves accuracy by correcting signal distortions caused by the atmosphere or reflected surfaces. This will benefit applications such as autonomous vehicles, drone navigation, and augmented reality. Continued improvements in AI and machine learning will also enhance how location data is interpreted and used, making GPS based services more intelligent, responsive, and personalized.

In summary, GPS technology on Android devices is a foundational feature that supports a wide range of applications across navigation, fitness, safety, business, and entertainment. Its seamless integration into the Android ecosystem allows developers to create innovative, location aware experiences while giving users powerful tools to navigate and interact with the world. As both GPS and Android continue to evolve, their combined potential will only grow, opening up new possibilities for connectivity, convenience, and exploration.

4.4 Interactive Maps

4.4.1 Introduction to OpenStreetMap

OpenStreetMap (OSM) [187,188] is a collaborative project that creates a free, editable map of the world. Often referred to as the "Wikipedia of maps" it is maintained by a global community of volunteers who contribute geographic data such as roads, trails, buildings, and points of interest. Launched in 2004 by Steve Coast in the UK, OSM was developed as a response to the restrictive licensing of commercial map data. Since then, it has grown into one of the largest open-source mapping platforms, used by individuals, companies, governments, and humanitarian organizations worldwide. The core idea behind OSM is openness, which means that anyone can use maps for free, contribute to them, and modify them under the Open Database License (ODbL). Unlike proprietary maps, which may not be updated frequently or may lack detail in less commercialized regions, OSM data is often highly detailed and frequently updated by local contributors. This makes it an attractive option for developers and organizations looking to build location based services without the licensing fees and limitations associated with platforms like Google Maps.

4.4.2 Integrating OpenStreetMap with Android

OpenStreetMap data is widely used in Android applications, especially those focused on mapping, navigation, tracking, and location services. Android developers can integrate OSM in various ways, using libraries and APIs that render OSM tiles or convert OSM data into usable formats. Some popular tools for integrating OSM with Android include OSMDroid, Mapsforge, Mapbox, and GraphHopper.

OSMDroid is an open-source Android library that allows developers to use OSM maps as an alternative to Google Maps. It supports features such as zoom controls, overlays, offline maps, and GPS tracking. Developers can use OSMDroid to create custom map applications without depending on Google Play Services. Mapbox is another popular platform that builds on OSM data, providing advanced tools for customizing maps, adding interactivity, and handling navigation and geolocation. Though Mapbox has its own licensing model, its reliance on OSM makes it an accessible alternative for developers needing more flexibility than what traditional mapping services offer. Additionally, developers can use GraphHopper or OpenRouteService for routing and navigation based on OSM data. These tools allow Android apps to calculate directions

for various travel modes like walking, cycling, and driving, including turn-by-turn navigation. These open-source tools enable developers to build powerful location-based services while keeping control over data and features.

4.4.3 Use Cases and Applications on Android

There are many Android apps that leverage OpenStreetMap data for different purposes. One of the most well-known is OsmAnd (OSM Automated Navigation Directions), a full-featured mapping and navigation app that uses offline OSM data. OsmAnd offers turn-by-turn voice guidance, hiking and cycling modes, and POI (point of interest) browsing. It is highly valued in regions with limited internet connectivity due to its robust offline capabilities. Maps.me is another popular OSM-based Android app. It provides downloadable maps for offline use, with fast rendering and routing features. This makes it an excellent companion for travelers who need reliable navigation without incurring data charges or depending on inconsistent mobile signals.

Humanitarian and disaster-response organizations also use Android apps built with OSM. For instance, MapSwipe and StreetComplete allow volunteers to contribute to OSM directly from their phones by validating or adding missing information. These apps are vital during natural disasters or humanitarian crises when accurate mapping of affected regions can support aid delivery and logistics.

In outdoor activities like hiking, cycling, and adventure sports, OSM-based apps often outperform commercial alternatives. Their ability to show trails, footpaths, topographic details, and user-contributed content makes them ideal for navigation in remote areas where Google Maps may lack detail. Moreover, Android GPS tracking apps that use OSM can help users record their routes, plan trips, and share location data with others.

4.4.4 Benefits of Using OSM on Android

There are several advantages to using OpenStreetMap data in Android development and applications. The most obvious is cost. Unlike commercial mapping platforms that charge usage fees for map views, API requests, or access to certain features, OSM data is free to use. This is particularly beneficial for startups, researchers, non-profit organizations, and independent developers who want to create location-aware services without incurring large expenses.

Flexibility is another significant benefit. Because OSM is open-source, developers can customize map styling, data layers, and interaction behaviors to suit their specific needs. Whether an app requires a minimal city map, a detailed topographic hiking map, or a route planner for delivery drivers, OSM can be tailored accordingly.

The community driven nature of OSM ensures that updates and corrections happen frequently, especially in regions where active contributors are present. In some cases, OSM maps are more accurate and up-to-date than their commercial counterparts. This is particularly true in rural areas, developing countries, or newly developed urban zones where commercial maps may lag in updates.

Offline functionality is another strong point. Many Android apps using OSM allow users to download entire regions or countries for offline use, something that is often limited or paid in commercial apps. This is vital for travellers, field workers, or people living in areas with poor internet connectivity.

Finally, using OSM aligns with values of digital sovereignty and openness. Developers and users are not locked into closed ecosystems or dependent on large corporations. This fosters innovation, experimentation, and global collaboration across diverse use cases and geographies.

4.4.5 Challenges and Future Potential

Despite its many strengths, there are challenges associated with using OSM in Android development. One issue is data consistency. Because OSM is user-contributed, the quality and completeness of data can vary by region. Some areas may have extremely detailed information, while others may lack basic features such as street names or building outlines. This can be problematic for applications requiring high levels of accuracy or uniform global coverage.

Another challenge is the technical complexity involved in integrating OSM data. Unlike proprietary platforms that offer plug-and-play solutions, working with raw OSM data may require handling large datasets, managing rendering engines, and setting up routing servers. For developers with limited resources or time, this can be a steep learning curve.

Legal and licensing concerns can also arise, especially when combining OSM data with proprietary data or services. Although the ODbL license is open, it requires that

derivative databases also be shared under the same terms, which may not align with every project's goals or commercial model.

Looking forward, the potential of OpenStreetMap on Android is huge. As mobile devices become more powerful and AI techniques evolve, we can expect smarter, more responsive location based applications. Real-time crowd sourced updates, dynamic mapping based on user behavior, and intelligent route planning are all on the horizon. The Android ecosystem, with its broad reach and developer community, will likely be at the forefront of these innovations.

Moreover, collaborations between OSM and public institutions or private companies could further enrich map data. As awareness of open data and privacy concerns grows, more developers and organizations may turn to OSM as a trustworthy alternative to closed mapping platforms. With continued community support and technological advancement, OpenStreetMap will remain a cornerstone of mobile mapping and navigation on Android for many years.

4.4.6 Implementing osmdroid in Android with Java and osmbonuspack library

One of the basic features of an application which is able to help tourists guide and find places of interest should be a service that could be able to make use of interactive digital maps and to support geo positioning and live routing. Google maps could be an option but its pricing policy makes it quite expensive. On the other hand osmdroid is free and makes use of the Open Street Map project.

OpenStreetMap (OSM) is a collaborative, open-source project that creates and maintains a free, editable map of the world. Instead of being managed by a single company, OSM is built by a global community of volunteers who collect and update geographic data such as roads, trails, buildings, rivers, and points of interest. Contributors use tools like GPS devices, aerial imagery, local knowledge, and mobile apps to record and upload map information. The main advantage of OpenStreetMap is that it’s open data. Anyone can use it, modify it, and share it, provided they follow the Open Database License (ODbL). This makes it very popular for developers, researchers, and organizations who need customizable map data without the restrictions or licensing costs that often come with commercial map providers. It powers many apps, navigation systems, disaster relief efforts, and even commercial services. In short, OSM is often

described as the “Wikipedia of maps,” because it’s constantly evolving, community-driven, and freely accessible.

On the other hand, osmdroid is an open-source Android library that provides a MapView component and related tools for displaying and interacting with maps based on OpenStreetMap (OSM) data. It serves as an alternative to Google Maps for Android, allowing developers to integrate customizable maps into their apps without needing an API key or being tied to Google’s licensing restrictions. With osmdroid, we can display online map tiles, cache them for offline use, and even load offline map files (like MBTiles). The library supports common mapping features such as zooming, panning, adding overlays (markers, polylines and polygons), showing the user’s location, and working with custom tile sources. It is designed to be modular and flexible, so developers can adapt it for applications like navigation, GIS, outdoor activities, and location-based services. Because osmdroid is community-driven and built on top of OpenStreetMap, it gives developers full control over how maps are displayed and what data sources are used.

In this section we will walk through the deployment of osmdroid library (an open-source Android map view) and osmbonuspack (an android library with useful overlays, geocoding, routing, KML, clustering, etc.) to a Java Android project. It covers setup, permissions, layout, core MapView usage, common overlays (markers, polylines), geocoding/routing, offline tiles, and useful tips/troubleshooting.

4.4.7 Android Project setup for OSM

In the Project setup stage, we must begin by adding the required dependencies for osmdroid (which provides the MapView and tile handling) and osmbonuspack (which adds extra features like overlays, clustering, routing, and geocoding) to our app’s build.gradle file. Since both libraries are hosted on Maven Central, we usually don’t need extra repositories, but we should always use the latest stable versions.

Next, we update the AndroidManifest.xml with essential permissions such as INTERNET, ACCESS_NETWORK_STATE, and location permissions (ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION), plus storage access if we plan to use offline tiles. On devices running Android 6.0 (Marshmallow) or higher, we also need to request user’s personal permissions (like location and storage) at runtime. Finally, we must configure the osmdroid user agent string in our code to

identify our app to tile servers, which is a key requirement to avoid being blocked when fetching map tiles. These mean the followings:

- Create a new Android project (Java) in Android Studio. Minimum SDK: API 21+ is recommended but osmdroid supports older versions.
- Add dependencies (Gradle, app/build.gradle), as shown in Figure 4.11:

```
implementation 'org.osmdroid:osmdroid-android:6.1.14'
implementation 'com.github.MKergall:osmbonuspack:6.9.0'
```

Figure 4.11: Dependencies for osmdroid and osmbonuspack

- Add repository if needed (most versions are available via Maven Central).
- Add AndroidManifest permissions and features, as shown in Figure 4.12:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Figure 4.12: Android Manifest permissions for osmdroid and osmbonuspack

Dealing with the files hierarchy we can include in a sample app the following:

- MainActivity.java — map setup, overlays, lifecycle
- activity_main.xml — MapView layout
- Utility classes for MapHelper.java to centralize tile sources, offline handling, and overlay creation
- Permission helper to request and handle runtime permissions

4.4.7.1 Implementing Layout and basic MapView

When setting up the layout and basic MapView, we first define the map in our activity's XML layout file. The MapView widget provided by osmdroid works like a regular Android view and can be placed inside any layout container, such as RelativeLayout or ConstraintLayout. By giving it an id, we can later reference it in our Java code. This

setup makes the map occupy the entire screen or a portion of it, depending on our design. Unlike Google Maps, osmdroid doesn't require an API key to start showing tiles, but we must ensure that the view has proper width, height, and positioning in the layout to display correctly. Adding match_parent for both width and height is a common practice when building a full-screen map activity. So, in res/layout/activity_main.xml we have to add a MapView element, as shown in Figure 4.13.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">


    <org.osmdroid.views.MapView
        android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />


</RelativeLayout>
```

Figure 4.13: MapView design implementation

In our activity (*MainActivity.java*), as shown in Figure 4.14, we initialize osmdroid and the MapView. In the Java activity class, we then configure it for use. The first critical step is to set the user agent value via *Configuration.getInstance().setUserAgentValue(getPackageName())*, which ensures that tile servers recognize your app and don't block your requests. After inflating the layout with *setContentView*, we retrieve the MapView using *findViewById*, enable multi-touch controls for pinch zooming, and set a starting location and zoom level through a MapController. The map can be centered on any GeoPoint (latitude, longitude), and from there, we can interact with it like a standard map using zooming, panning, and preparing to add overlays such as markers and paths. Additionally, calling *map.onResume()* and *map.onPause()* inside the activity's lifecycle methods ensures proper resource handling when the app is paused or resumed.

```java
public class MainActivity extends AppCompatActivity {
    5 usages
    private MapView map;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Important: set user agent for tile requests
        Configuration.getInstance().setUserAgentValue(getPackageName());
        setContentView(R.layout.activity_main);
        map = findViewById(R.id.map);
        map.setMultiTouchControls(true);

        MapController mapController = (MapController) map.getController();
        mapController.setZoom(15.0);
        GeoPoint startPoint = new GeoPoint( aLatitude: 37.98,  aLongitude: 23.72); // e.g., Athens
        mapController.setCenter(startPoint);
    }

    @Override
    public void onResume() {
        super.onResume();
        map.onResume();
    }

    @Override
    public void onPause() {
        super.onPause();
        map.onPause();
    }
}
```

Figure 4.14: MapView configuration example

4.4.8 Markers, InfoWindows, and basic overlays of osmbonuspack

To add a simple marker with a custom InfoWindow in osmdroid with osmbonuspack, we first create a Marker object, set its position using a GeoPoint (latitude and longitude), and configure its appearance (for example, adjusting the anchor point so the marker's bottom aligns with the map location). You can then set a title and snippet, which display in the default InfoWindow when the marker is tapped. If we want more control, osmbonuspack allows us to define a custom InfoWindow. This is essentially a popup view that can contain any layout, such as images, buttons, or styled text. By subclassing InfoWindow and overriding its onOpen and onClose methods, you can bind custom data and interactions to the marker. Finally, as shown in Figure 4.15, we add the marker to the map's overlay list and call map.invalidate() to refresh the display, resulting in an interactive marker that opens our custom InfoWindow when tapped. osmbonuspack provides Marker, Polyline, FolderOverlay, and helpers like POI and

ItemizedIconOverlay. We can subclass Marker or provide a custom InfoWindow to show richer layouts (images, buttons).

```
Marker marker = new Marker(map);
marker.setPosition(new GeoPoint( aLatitude: 37.98,  aLongitude: 23.72));
marker.setTitle("My Marker");
marker.setSnippet("This uses osmbonuspack InfoWindow by default");
marker.setAnchor(Marker.ANCHOR_CENTER, Marker.ANCHOR_BOTTOM);
map.getOverlays().add(marker);
map.invalidate();
```

Figure 4.15: Adding a simple marker with a custom InfoWindow

Drawing a polyline (path) in osmdroid involves creating a Polyline overlay and supplying it with a list of GeoPoint objects that represent the sequence of geographic coordinates we want to connect. Each GeoPoint contains a latitude and longitude, and when combined, the library draws straight lines between them on the map, forming a continuous path. We can customize the polyline's appearance by adjusting its width, color, and transparency, making it useful for showing routes, boundaries, or user tracks. Once configured, the polyline is added to the map's overlay collection with *map.getOverlays().add(polyline)*, and then the map is refreshed with *map.invalidate()* so the path becomes visible. This feature, shown in Figure 4.16, is especially helpful for visualizing navigation routes, hiking trails, or any connected set of geographic points.

```
Polyline line = new Polyline();
List<GeoPoint> pts = new ArrayList<>();
pts.add(new GeoPoint( aLatitude: 37.98,  aLongitude: 23.72));
pts.add(new GeoPoint( aLatitude: 37.99,  aLongitude: 23.73));
line.setPoints(pts);
line.setWidth(6f);
map.getOverlays().add(line);
```

Figure 4.16: Drawing a polyline (path)

4.4.8.1 Routing with osmbonuspack

The OSRMRoadManager in osmbonuspack provides a straightforward way to calculate and display routes between waypoints using the Open Source Routing Machine (OSRM) service, shown in Figure 4.17. To use it, we first create an instance of OSRMRoadManager, then supply it with an ArrayList of GeoPoint objects representing our start point, intermediate waypoints (if any), and destination. The manager sends

these coordinates to the OSRM service (which is an online URI path), which returns a Road object containing route information such as geometry, length, and estimated duration. We can then use *RoadManager.buildRoadOverlay(road)* to generate a polyline overlay that visually represents the route on the map, and add it to the map's overlay list. Finally, refreshing the map makes the route appear. This approach is useful for navigation or travel apps, though it's important to note that public OSRM servers may have usage limits, so for production apps we might consider hosting our own routing server or switching to another supported provider like GraphHopper. We must point out that mobile device must have internet to be able to calculate routing paths or else the osmbonuspack will only show a straight line from user position (start point) to destination.

```
RoadManager roadManager = new OSRMRoadManager();
ArrayList<GeoPoint> waypoints = new ArrayList<>();
waypoints.add(new GeoPoint( aLatitude: 37.98,  aLongitude: 23.72));
waypoints.add(new GeoPoint( aLatitude: 37.99,  aLongitude: 23.73));


Road road = roadManager.getRoad(waypoints);
org.osmdroid.views.overlay.Polyline roadOverlay = RoadManager.buildRoadOverlay(road);
map.getOverlays().add(roadOverlay);
map.invalidate();
```

Figure 4.17: Routing example with OSRMRoadManager

4.4.8.2 Location, permissions, and lifecycle

To be able to get user's location we must request ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION at runtime. We can use *FusedLocationProviderClient*, shown in Figure 4.18, to get device location and then show a custom *MyLocationNewOverlay* from osmdroid to display user location.

```
MyLocationNewOverlay mLocationOverlay = new MyLocationNewOverlay(new GpsMyLocationProvider( context: this), map);
mLocationOverlay.enableMyLocation();
map.getOverlays().add(mLocationOverlay);
```

Figure 4.18: Requesting access to user's location

Because of the fact that the procedure of getting user's location is an asynchronous non-blocking task we must assign this task in a new independent thread and all procedures connected to this service must wait for it to complete its task. This also means that

routing service (which is an asynchronous non-blocking remote task itself) must wait firstly for our device to be able to lock user's position and then request routing from the online Open Source Routing Machine (OSRM) provider.

```
////////////////////////////// Navigate From User's Location to Destination ////////////////////////////////
ArrayList<GeoPoint> waypoints = new ArrayList<GeoPoint>();  // add current user's location and destination to this list
GeoPoint endpoint = point;      // this is destination
waypoints.add(endpoint);

// Get Current User's Position
mLocationOverlay.runOnFirstFix(new Runnable() {
    public void run() {
        waypoints.add(new GeoPoint(mLocationOverlay.getLastFix().getLatitude(),mLocationOverlay.getLastFix().getLongitude()));
        new UpdateRoadTask().execute(waypoints);    // when user's location is found find route
    }
});
```

Figure 4.19: Navigating from User's position to Destination

As shown in Figure 4.19, in a new thread, when we manage to lock user's location then we ask using another thread called *UpdateRoadTask* the online service of osmbonuspack to find a route between user's location and destination

4.4.8.3 Performance, and troubleshooting

When working with osmdroid, setting up a proper user agent is one of the most critical tips. Tile servers like OpenStreetMap require apps to identify themselves, so you should call *Configuration.getInstance().setUserAgentValue(getPackageName())* early in your app. This prevents your requests from being blocked and ensures compliance with usage policies. Another best practice is to handle permissions carefully. For example, location and storage permissions are considered dangerous on Android, so they must be requested at runtime. If users deny these permissions, your app should fail gracefully by disabling certain features while still allowing access to the map view. Respecting user privacy and permissions will also help our app meet Play Store requirements.

In terms of performance, osmdroid offers several optimizations. Tile caching is enabled by default, so maps load faster after the first use, but we may want to manage cache size using *Configuration.getInstance().setTileFileSystemCacheMaxBytes(...)* to avoid filling up storage. When displaying large numbers of markers, using the *RadiusMarkerClusterer* overlay from osmbonuspack is recommended, as it groups nearby markers into clusters and prevents performance issues caused by rendering hundreds of markers at once. Additionally, heavy operations such as geocoding or routing should never run on the UI thread. They should be executed asynchronously to avoid blocking the map's responsiveness. This ensures smooth interactions like panning and zooming, even while background tasks are running.

For troubleshooting, we must always call *map.invalidate()* after adding or updating overlays to ensure changes are rendered. If tiles aren't loading, we must check that our user agent is set and that the device has network access. When testing offline tiles, we must confirm they are stored in the correct directory (by default /sdcard/osmdroid/) and in a supported format like MBTiles or zip archives. If location tracking fails, we must verify that GPS or network location providers are enabled on the device and that the app has runtime permissions. Finally, we must keep an eye on logcat for error messages from osmdroid, as they often give precise clues like missing permissions, invalid tile sources, or network timeouts that help us quickly diagnose issues. With careful setup and attention to these details, we can create reliable and performing mapping features in our Android app. Summarizing, we must pay attention to the following points and keep them in mind as a recipe of good practices:

- User agent, (always set a meaningful user agent since tile servers may block anonymous clients).
- Avoid blocking network calls on UI thread, (geocoding and routing requests should run off the UI thread).
- Clustering and tile zoom limits, (use clustering for >100 markers and control tile cache size with Configuration).
- Permissions, (handle denied permissions gracefully and explain to users why location/storage is needed).
- Map tile providers, (use responsible providers and check tile provider terms of service).
- Debugging overlays, (call map.invalidate() after changing overlays).

4.5 QR Code

4.5.1 Introduction to QR Code Technology

Quick Response (QR) codes are two dimensional barcodes that store data in a square grid of black and white modules [189,190]. Developed in 1994 by the Japanese company Denso Wave, QR codes were initially used in the automotive industry for tracking parts. Over time, their use has expanded into consumer facing applications

such as marketing, payments, and logistics due to their capacity to hold more data than traditional barcodes and their ability to be read quickly from any orientation.

A QR code can encode various types of data, including URLs, contact information, Wi-Fi credentials, plain text, calendar events, or payment information. Thanks to error correction techniques, QR codes can still be read even if partially damaged or dirty. This robustness, combined with their flexibility, makes them an ideal medium for touchless interactions, a quality that became particularly valuable during the COVID-19 pandemic.

On Android devices, QR codes are typically scanned using the device's camera, and the embedded data is then parsed and acted upon by apps. While some Android phones come with native QR scanning capabilities built into the default camera app, many third-party apps offer enhanced features such as multi-code scanning, offline reading, or advanced formatting. The popularity of QR codes and the ubiquity of smartphones have turned Android devices into powerful tools for reading and interacting with these codes.

4.5.2 Implementing QR Code Readers in Android

Android offers several methods for implementing QR code readers, ranging from low-level camera APIs to high-level libraries that abstract much of the complexity. One of the most widely used tools is Google's ML Kit, which provides a Barcode Scanning API capable of reading various barcode formats, including QR codes. ML Kit simplifies the development process by handling image processing, camera input, and data extraction, all with minimal code.

Another popular option is ZXing (Zebra Crossing), an open-source barcode image processing library that supports QR codes and many other formats. ZXing provides Android developers with tools to integrate scanning capabilities directly into their apps. The library can be used through an embedded scanner view or by invoking an external intent to a pre-installed scanning app. It is lightweight, fast, and widely supported, making it a good choice for many use cases.

To implement a QR code scanner, developers typically start by integrating the chosen library into their Android project and setting up permissions to access the device's camera. A scanner view is then added to the app's UI, allowing the user to point the camera at a QR code. Once the camera detects a code, the data is extracted and passed

to a handler function where the developer can define what happens next, such as opening a URL, saving contact info, or connecting to a Wi-Fi network.

Newer Android devices and versions have started integrating native QR code recognition into the operating system. For example, Android 10 and later can scan QR codes through the system camera app or via Google Lens. However, for apps that require customized functionality or QR interaction as a core feature, dedicated in-app scanning remains essential.

4.5.3 Use Cases, Benefits, and Best Practices

QR code scanning has unlocked a wide array of use cases across various industries. In retail and marketing, QR codes are used to link customers directly to promotional websites, product information, or social media pages. Restaurants often use them to display menus, reducing the need for physical contact. In transportation and events, QR codes serve as digital tickets or boarding passes that can be validated quickly using mobile devices.

Another rapidly growing application is in mobile payments. Many payment systems, including Google Pay, use QR codes for peer-to-peer transfers and merchant transactions. Users simply scan a QR code to pay, bypassing the need for cash or physical cards. In logistics, QR codes help track packages through each stage of the delivery process, with Android-based handheld devices scanning codes to update real-time status.

Education and healthcare have also adopted QR codes to streamline services. Educational institutions use them for attendance tracking, content sharing, and examination access. In healthcare, QR codes can store patient data, appointment details, or vaccination records, all easily accessible through a quick scan.

Despite their versatility, developers should follow best practices to ensure a smooth and secure user experience. First, always validate the scanned data before acting on it. Malicious actors can create QR codes that link to harmful websites or trigger unwanted actions, so apps should verify content (especially URLs) before proceeding.

Second, consider offline capability. Not all users will have a stable internet connection, so apps should handle QR data locally when appropriate, or queue actions until the device is back online. Also, developers should design apps to handle various lighting

conditions and camera quality to ensure reliable scanning performance across different devices.

Finally, user privacy and data security must be considered. If the scanned data involves personal information, ensure that the app adheres to data protection regulations and uses secure storage methods. Inform users when scanning sensitive information, and give them control over what data is saved or transmitted.

In conclusion, QR code readers on Android devices offer a highly efficient and scalable solution for interacting with the physical world. Whether for business, utility, or user convenience, QR code integration can greatly enhance the functionality and user experience of Android apps. With the right implementation and attention to security and usability, developers can leverage this technology to build innovative, responsive, and engaging mobile applications.

4.5.4 Implementing a QR Code Reader in Android with ZXing Library

QR codes (Quick Response codes) have become a widely used method of storing information such as URLs, contact details, and product information. In Android applications, QR code scanning is often needed in e-commerce apps, payment gateways, ticket verification systems, and more. One of the most popular open-source libraries for handling QR codes is ZXing (Zebra Crossing). This library supports both generating and scanning barcodes/QR codes, making it a convenient choice for developers.

We will focus on implementing a QR code reader in Android using Java and ZXing library. We will cover project setup, permissions, integration, and code implementation.

4.5.4.1 Setting Up the Project

1. **Create an Android Project with** a minimum SDK version that supports our target devices (API level 21 or above is recommended)
2. **Add ZXing Dependency**. ZXing is available via Gradle. Add the following dependency inside `app/build.gradle` file:
    a. implementation 'com.journeyapps:zxing-android-embedded:4.3.0'
    b. The library (`zxing-android-embedded`) also provides camera integration for scanning.
3. **Sync the Project** to download and include the libraries.

4.5.4.2 Adding Required Permissions

QR code scanning requires access to the device's camera. Add the following permission inside AndroidManifest.xml:

<uses-permission android:name="android.permission.CAMERA"/>

Also, add the camera feature declaration to inform the Play Store that our app uses the camera:

<uses-feature android:name="android.hardware.camera.any"/>

4.5.5 Creating the Scanner Activity

The zxing-android-embedded library provides a simple way to open a scanning interface using the following *barLauncher*. As soon as the QR Code reader detects a QR Code figure it translates its content and the result is forwarded to an object of *RecordLoader* Class which is able to communicate with our Firebase Realtime Database and load the accord *TourGuide* reference. In each QR code image we are able to store the unique "key" value of a *TourGuide*. Using this techniques we are able to distinguish the records hosted in our online database. Here's how to implement it in the Android activity in Figure 4.20.

```
public class QRCodeReader extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Reader reader = new Reader(barLauncher);
    }

    1 usage
    ActivityResultLauncher<ScanOptions> barLauncher = registerForActivityResult(new ScanContract(), result->{
        if(result.getContents() != null){   // Found QRCode from camera
            Log.d( tag: "QRCodeReader",result.getContents() );

            RecordLoader recordLoaderByKey = new RecordLoader( ctx: this);
            recordLoaderByKey.readKey(result.getContents());
        }
        finish();
    });
}
```

Figure 4.20: Activation of QR code reader

The following code, in Figure 4.21, automatically opens the camera, scans the QR code, and returns the decoded result in the barLauncher variable. The result can be displayed

in a Toast, stored in a database, or used to perform an action such as opening a webpage.

```
public class Reader{
    1 usage
    public Reader(ActivityResultLauncher<ScanOptions> barLauncher) {
        ScanOptions options = new ScanOptions();
        options.setPrompt("Volume up to flash on");
        options.setBeepEnabled(true);
        options.setOrientationLocked(true);
        options.setCaptureActivity(CaptureActivity.class);
        barLauncher.launch(options);
    }
}
```

Figure 4.21: Implementation of QR code reader

To test and deploy the algorithms above we have to do the followings.

- Run the app on a physical device (emulators generally lack camera support).
- Point the camera at a QR code.
- Verify that the scanned result matches the QR code's content.

Implementing a QR code reader in Android using Java and the ZXing library is straightforward and efficient. With just a few lines of code, developers can integrate QR scanning capabilities into their applications. ZXing offers flexibility by providing both intent-based scanning and custom in-app scanning views. This feature is useful in real-world scenarios such as payments, authentication, product identification, and ticketing systems. By following the steps above, developers can quickly create reliable and user-friendly QR code readers in their Android applications.

4.6 Biometric Authentication

4.6.1 Introduction to Biometric Authentication

Biometric authentication has become a cornerstone of modern security systems, offering a method of verifying identity through physical or behavioral characteristics unique to individuals [191,192]. Unlike traditional security methods such as passwords or PINs, biometrics are inherently tied to a person and are much harder to forge or steal. This makes them particularly effective in high-security environments, from smartphones to border control systems. The most commonly used biometric identifiers include fingerprints, facial recognition, iris scans, voice patterns, and even behavioral traits such as walk or typing rhythm. As technology advances, the precision and reliability of

biometric systems continue to improve, making them more accessible and widely adopted.

One of the key advantages of biometric authentication is its convenience. Users no longer need to remember complex passwords or carry physical tokens like ID cards. A quick scan of a finger or face is often all that’s needed to unlock a device or access secure data. This ease of use is particularly important in consumer electronics, where user experience is critical. For example, smartphones today commonly feature fingerprint sensors or facial recognition systems for unlocking devices and authorizing mobile payments. These features not only enhance user experience but also encourage the adoption of more secure practices among users who might otherwise rely on weak or reused passwords.

4.6.2 Privacy Risks and Data Security

Despite its advantages, biometric authentication is not without challenges. Privacy concerns are at the forefront of the debate. Since biometric data is immutable, unlike passwords, which can be changed, it poses a significant risk if compromised. If a database containing fingerprint or facial recognition data is breached, users cannot simply update their biometric traits to protect themselves. This makes secure storage and handling of biometric data critical. Many modern systems mitigate this risk by storing biometric data locally on the device in a secure enclave rather than transmitting it to a central server. This approach reduces the potential attack surface and protects user privacy to a greater extent.

4.6.3 Accuracy, Inclusivity, and System Limitations

Accuracy and inclusivity are also concerns with biometric systems. Environmental factors, aging, injuries, or physical differences can impact the reliability of biometric readings. For instance, facial recognition systems may struggle with identical twins or varying lighting conditions, and fingerprint sensors might fail for individuals with worn fingerprints due to manual labor. Moreover, certain demographics such as older adults or people with disabilities may face challenges with biometric systems that are not designed inclusively. These limitations highlight the need for multi-modal biometric systems that combine two or more types of biometric data for more robust authentication. Such systems can improve reliability while offering alternatives for users with difficulty using a single biometric method.

4.6.4 How Biometric Systems Work

From a technical standpoint, biometric systems operate through a process involving enrollment, storage, and matching. During enrollment, the system captures and stores a user's biometric template, which is a digital representation of their unique traits. Later, during authentication, a new sample is captured and compared against the stored template using pattern matching algorithms. The system then makes a decision based on the similarity score. Modern biometric systems use advanced machine learning techniques to enhance the accuracy of this process, especially in dynamic environments. However, the robustness of these algorithms must be continuously assessed to prevent false positives or false negatives, which can compromise either security or usability.

4.6.5 Public Security and Ethical Concerns

Biometric authentication is also seeing growing application in public security and surveillance. Law enforcement agencies increasingly use facial recognition to identify suspects in crowds or match images from security footage. Border control agencies utilize iris and fingerprint scans to verify traveler identities efficiently and reduce processing times. While these implementations enhance operational efficiency and public safety, they also raise ethical and legal concerns about surveillance and potential abuse. In some jurisdictions, regulations have been introduced to ensure transparency, accountability, and the protection of civil liberties in biometric deployments. The balance between security and privacy remains a central issue in the broader adoption of biometric technologies.

4.6.6 Adoption in the Financial Sector

Financial institutions are another major sector adopting biometrics to enhance customer security and reduce fraud. Banks are integrating biometric login options into their mobile apps, while some ATMs now support biometric authentication to replace or augment card and PIN access. This dual-factor approach combining something the user knows with something they are, adds an extra layer of protection. However, the integration of biometrics in financial systems also necessitates strict regulatory compliance, especially concerning data protection laws such as the General Data Protection Regulation (GDPR) in Europe or the California Consumer Privacy Act (CCPA) in the United States. These regulations mandate informed consent, minimal data collection, and secure processing practices, making legal compliance an essential part of biometric system implementation.

4.6.7 Digital Identity and Centralized Systems

Biometric authentication is also central to the development of digital identity frameworks, which aim to provide secure, portable, and verifiable digital credentials. Governments and private entities are collaborating to create unified systems where biometric data can be used for seamless access across services ranging from healthcare to education and to e-commerce. In such systems, a user's biometric traits serve as a digital key to verify their identity across multiple platforms, reducing the need for multiple logins and redundant documentation. However, centralizing identity verification through biometrics also creates significant risks if the infrastructure is not resilient to attacks or misuse. The challenge lies in building systems that are not only efficient and scalable but also ethical and trustworthy.

4.6.8 Emerging Technologies and Innovations

Advancements in artificial intelligence and sensor technology are pushing the boundaries of biometric authentication even further. Technologies like 3D facial recognition, thermal imaging, and vein pattern recognition offer more sophisticated ways to differentiate individuals with greater accuracy and resistance to spoofing. Behavioral biometrics such as the way a person types, swipes or walks, are also gaining traction because they can provide continuous authentication rather than just a one-time check. These methods add an additional dimension of security, especially in high risk environments like cyber security defense or sensitive data handling. Yet, these innovations also demand rigorous testing and oversight to ensure they are not discriminatory or biased in their operations.

4.6.9 The Future of Biometric Authentication

Looking ahead, the future of biometric authentication will likely involve a layered approach that integrates biometric data with other forms of identity verification. The goal is to create a seamless, secure user experience while addressing the inherent limitations of any single authentication method. For example, combining biometrics with cryptographic keys or decentralized identity frameworks could offer more resilient and privacy preserving solutions. Additionally, greater transparency in how biometric systems work, combined with user education, will be key to building public trust. As society becomes increasingly digitized, biometric authentication stands as both a

promising solution and a profound responsibility, shaping the way identity and security will be managed for years to come.

4.6.10 Android Fingerprint Technology, Mobile Security and User Experience

In the ever evolving landscape of mobile technology, biometric authentication has become a cornerstone of security and convenience. Among the most widely adopted biometric systems is fingerprint recognition. Android fingerprint technology, in particular, has revolutionized the way users interact with their smartphones and secure their personal data. Introduced widely in Android 6.0 Marshmallow, the fingerprint API provided a standardized way for devices and developers to harness the power of biometric authentication. Since then, it has grown increasingly sophisticated, integrating advanced hardware and software innovations to deliver a seamless, secure, and user friendly experience.

Fingerprint authentication in Android devices operates through a combination of hardware sensors and software algorithms. When a user registers their fingerprint, the sensor captures the unique patterns of ridges and valleys on the fingertip. These patterns are then converted into a digital template, which is stored securely in a trusted execution environment (TEE) such as a secured file on the device or a dedicated piece of hardware. The actual image of the fingerprint is NOT stored. Only a mathematical representation of the fingerprint data is saved, ensuring that even if the device is breached, the raw biometric information remains inaccessible. During authentication, the fingerprint presented is compared against the stored template, and if a match is found, access is granted.

The sensors used in Android devices vary by manufacturer and device model. They typically fall into three categories such as optical, capacitive, and ultrasonic. Capacitive sensors, which are the most common, use electrical currents to map the fingerprint and are highly accurate due to their ability to detect the fine details of the ridges. Optical sensors, which use light to capture a 2D image of the fingerprint, are generally less secure and have fallen out of favor for high-security applications. Ultrasonic sensors, introduced in more recent flagship devices, use high-frequency sound waves to create a 3D map of the fingerprint, offering superior accuracy and the ability to function through dirt, oil, and moisture. These advances in sensor technology have contributed significantly to the robustness and reliability of Android fingerprint authentication.

From a software standpoint, Android provides developers with tools through its Biometric Prompt API, which supports fingerprint scanning along with other biometric methods like facial and iris recognition. This API allows for a consistent and secure user interface across different apps and devices. By delegating authentication to the system, developers can ensure that biometric data is handled appropriately without direct access, reducing the risk of security flaws. The API also supports fallback mechanisms, such as PINs or patterns, for users who prefer alternative methods or when biometric authentication fails. Additionally, Android has incorporated machine learning into its biometric systems, allowing for continuous improvement in recognition accuracy and fraud detection.

Security is a paramount concern in fingerprint authentication. Android's fingerprint technology is designed with several layers of protection to guard against spoofing and unauthorized access. Liveness detection, for example, prevents attackers from using artificial replicas of fingerprints, such as silicone molds. The secured enclave, typically a physically isolated part of the processor, ensures that fingerprint data cannot be accessed by the operating system or applications. Android also mandates certification processes for biometric hardware, ensuring that devices meet rigorous standards before enabling fingerprint based authentication for sensitive tasks like mobile payments or banking. As of Android 10, Google introduced the Biometric Security Levels classification ranging from strong to weak and convenience, to help users and developers understand the reliability of different biometric implementations.

In more details Biometric security levels are categorized into three classes: Class 3 (formerly Strong), Class 2 (formerly Weak), and Class 1 (formerly Convenience).

These classes indicate the strength of the biometric implementation, with Class 3 offering the highest level of security and Class 1 the lowest:

- Class 3 (Strong), is considered the strongest and is suitable for protecting sensitive data and applications, including those that handle financial transactions or store cryptographic keys. Class 3 biometrics have a low Spoof Acceptance Rate (SAR) and False Acceptance Rate (FAR), meaning they are difficult to bypass with fake biometric data or by incorrectly identifying an authorized user.
- Class 2 (Weak), is still offering a reasonable level of security. Class 2 biometrics are not strong enough to protect sensitive data like cryptographic keys. They

may have a higher SAR and FAR than Class 3, making them more susceptible to spoofing or false acceptance.

- Class 1 (Convenience), focuses on user convenience rather than security. Class 1 biometrics are not suitable for securing sensitive data or applications and are primarily used for unlocking devices or basic authentication.

It's important to note that the classification of biometrics can vary depending on the specific implementation and platform. For example, facial recognition can be classified as either Class 3 or Class 2 depending on the implementation and fingerprint recognition is often categorized as Class 2.

The adoption of fingerprint technology in Android has significantly enhanced user convenience. Unlocking a phone, authorizing app logins, approving purchases, and verifying identity in secure apps can all be done with a single touch. This frictionless experience not only saves time but also encourages users to adopt stronger security practices. Before biometrics, many users avoided using screen locks due to the hassle of entering PINs or passwords. With fingerprint authentication, however, enabling device security became effortless, boosting overall device protection. Furthermore, the integration of fingerprint sensors under the display in modern Android devices reflects a shift toward minimalism and user-centric design, making security practically invisible and non-intrusive.

Despite its advantages, Android fingerprint technology is not without challenges. One major concern is the variability in implementation across manufacturers. Unlike Apple’s tightly controlled hardware-software ecosystem, Android spans a diverse range of devices with differing sensor quality and software optimization. This fragmentation can lead to inconsistent performance and user experience. Another concern is user privacy. Although Android’s architecture is designed to safeguard biometric data, public trust can be undermined by security breaches or unclear data handling policies by manufacturers and app developers. As a result, continued transparency, rigorous compliance with privacy laws, and regular security updates are essential to maintaining user confidence.

Looking forward, the future of Android fingerprint technology promises exciting developments. Innovations like multi-fingerprint authentication, where multiple fingers must be presented to unlock a device, are being explored to enhance security. In-display fingerprint sensors will continue to improve, becoming faster and more accurate.

Additionally, hybrid biometric systems that combine fingerprint recognition with facial or voice authentication could offer multi-modal security tailored to various use cases. Advancements in artificial intelligence will also play a role in detecting spoofing attempts, analyzing behavioral patterns, and adapting to subtle changes in a user's fingerprint due to aging or environmental factors.

In conclusion, Android fingerprint technology has become an integral part of the mobile security ecosystem, providing a secure and convenient mean of authentication for millions of users worldwide. Its evolution from basic capacitive sensors to sophisticated ultrasonic readers under the screen reflects the rapid pace of innovation in this space. Coupled with Android's robust biometric APIs and security frameworks, fingerprint authentication offers a compelling balance of usability and protection. As the technology continues to mature, and as security threats evolve, Android's fingerprint systems will undoubtedly remain at the forefront of biometric security, shaping the way users engage with their digital lives.

4.6.11 Implementing Biometrics in Android with Java

In this section we will briefly show how to implement biometric authentication (fingerprint, face, or other device-supported biometrics) in an Android app using Java. It covers setup, code examples using the AndroidX BiometricPrompt API, UX and security considerations. The examples target Android API levels where BiometricPrompt is available although AndroidX supports back to older API levels through compatibility libraries.

First we need a device or emulator with biometric support (in emulator configure virtual sensors in AVD Manager). Then we must check the minimum SDK required since BiometricPrompt via AndroidX works on many older API levels, but some biometric features require newer Android versions. And next we need to add the AndroidX Biometric dependency in gradle as shown in the Figure 4.22.

```
dependencies {
    implementation "androidx.biometric:biometric:1.2.0" // check latest version
}
```

Figure 4.22: Biometrics Dependency

Before developing source code we must explore some important concepts:

- **BiometricPrompt**: the recommended high-level API to show biometric UI and handle results.
- **BiometricManager**: check whether biometric authentication is available and enrolled.
- **BiometricPrompt.AuthenticationCallback**: receives authentication results (success, failure, errors).

4.6.11.1 Basic Implementation

Check availability

Before showing biometric prompt, we have to confirm hardware and enroll biometrics as shown in Figure 4.23. We can use BIOMETRIC_STRONG for modern biometrics (Authentication allows fingerprint sensor only), BIOMETRIC_WEAK (Authentication allows all biometrics i.e. fingerprint, face, and iris) or DEVICE_CREDENTIAL which allows fallback to device PIN/pattern.

```
BiometricManager biometricManager = BiometricManager.from(context);

int canAuthenticate = biometricManager.canAuthenticate(BiometricManager.Authenticators.BIOMETRIC_STRONG |
        BiometricManager.Authenticators.DEVICE_CREDENTIAL);

if (canAuthenticate == BiometricManager.BIOMETRIC_SUCCESS) {
    // good to go
} else {
    // fallback: ask for PIN/password or show message
}
```

Figure 4.23: Declaring the types of Biometric Authentication

Build the Biometric Prompt

To build a biometric prompt we have to create an Executor as shown Figure 4.24, a *BiometricPrompt.AuthenticationCallback*, and show the prompt. If we prefer device PIN/fallback but not a negative button, include *setAllowedAuthenticators(BiometricManager.Authenticators.BIOMETRIC_STRONG | BiometricManager.Authenticators.DEVICE_CREDENTIAL)* in the builder. The AndroidX Biometrics Library provides a method called *setNegativeButtonText()* that provides an option for using an account/app credential if the user doesn't want to use biometrics.

```
Executor executor = ContextCompat.getMainExecutor(this);

BiometricPrompt.AuthenticationCallback callback = new BiometricPrompt.AuthenticationCallback() {
    @Override
    public void onAuthenticationSucceeded(@NonNull BiometricPrompt.AuthenticationResult result) {
        super.onAuthenticationSucceeded(result);
        // Successful auth — proceed to protected action
    }


    @Override
    public void onAuthenticationFailed() {
        super.onAuthenticationFailed();
        // biometric matched but not recognized — keep UX friendly
    }


    @Override
    public void onAuthenticationError(int errorCode, @NonNull CharSequence errString) {
        super.onAuthenticationError(errorCode, errString);
        // handle permanent errors (e.g., lockout)
    }
};


BiometricPrompt biometricPrompt = new BiometricPrompt(this, executor, callback);


BiometricPrompt.PromptInfo promptInfo = new BiometricPrompt.PromptInfo.Builder()
.setTitle("Sign in")
.setSubtitle("Use your fingerprint to sign in")
.setNegativeButtonText("Use account password") // required if not using DEVICE_CREDENTIAL
.build();


// To show the prompt:
biometricPrompt.authenticate(promptInfo);
```

Figure 4.24: Setup of Biometric Prompt in Android

Handle results

In *onAuthenticationSucceeded* we have to unlock the protected content or continue the operation. We have to keep UI feedback simple (toast, navigate, unlock key material). Moreover, some security best practices techniques include the followings:

- Always provide a fallback (PIN/password) for users without biometrics enrolled.
- Don’t store raw biometric data. Android never exposes raw biometric templates if we rely on platform APIs and Keystore.
- Keep prompt text clear and explain why biometric is requested.
- Handle lockouts and error codes gracefully (e.g., too many failed attempts triggers *BiometricConstants.ERROR_LOCKOUT*).
- Avoid long-running operations in the authentication callback, do minimal work and then move to the background if necessary.
- When using *setUserAuthenticationValidityDurationSeconds*, understand that short windows require re-authentication for each operation.

4.7 Text-to-Speech

4.7.1 Introduction to Text-to-Speech in Android

Text-to-Speech (TTS) technology allows devices to convert written text into spoken voice output [193,194]. In Android, the TTS engine provides developers with a way to enhance accessibility and improve user interaction by giving applications the ability to "speak" text aloud. This is particularly valuable for users with visual impairments, literacy challenges, or situations where hands-free interaction is preferred.

Android's TTS functionality is built into the operating system, making it accessible to all applications without requiring additional third party software. The core class used to implement this feature is TextToSpeech, a part of the Android SDK. It interfaces with the system TTS engine and enables developers to configure the language, pitch, and speech rate of the voice output. Most Android devices come with Google’s TTS engine pre-installed, which supports multiple languages and regional dialects. However, users can install and select other engines from the Play Store based on their preferences.

The value of TTS in Android goes beyond accessibility. It plays a vital role in e-learning apps, navigation, language translation, audiobooks, virtual assistants, and more. By giving apps a "voice", developers can provide users with an auditory experience that complements visual or text-based interfaces. As devices become more interactive and user centric, integrating TTS becomes an increasingly important feature in modern Android applications.

4.7.2 Implementing Text-to-Speech in Android Applications

Implementing Text-to-Speech in Android is straight forward thanks to the *TextToSpeech* class. Developers begin by initializing a *TextToSpeech* object, typically in an activity or service. This initialization requires a context and a listener to handle the success or failure of the TTS engine startup. Once initialized, developers can call the *speak()* method to read out any string of text.

Here’s a basic outline of how to implement TTS, shown in Figure 4.25.

```
TextToSpeech tts = new TextToSpeech(context, new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS) {
            tts.setLanguage(Locale.US);
            tts.speak("Hello, welcome to the app!", TextToSpeech.QUEUE_FLUSH, null, null);
        }
    }
});
```

Figure 4.25: TTS Engine Setup Example

This snippet sets up the TTS engine, specifies the language (e.g., US English), and speaks a simple greeting. The QUEUE_FLUSH parameter tells the engine to clear any existing queue before speaking the new text. Developers can also adjust parameters like pitch (*setPitch(float)*) and speech rate (*setSpeechRate(float)*) to better match the app's tone or target audience.

TTS can be used in various components of an Android app. For instance, an educational app can pronounce vocabulary words aloud when clicked, or a news app can read articles to the user. Navigation apps often use TTS to deliver turn-by-turn directions. It is also common in chatbots or voice assistants to read back responses to user queries. Moreover, Android supports output to either the device's speaker or a connected Bluetooth audio device. Last, TTS can be used offline, as long as the necessary language files are installed on the device. This makes it suitable for use cases where network access is limited or unavailable.

4.7.3 Use Cases, Challenges, and Best Practices

Text-to-Speech adds a valuable auditory layer to Android apps, and its use cases are vast. Accessibility is one of the primary motivations for TTS. Screen readers for blind or visually impaired users rely on TTS to read UI elements, notifications, and other content. Android's native accessibility service, TalkBack, uses TTS for this purpose. Developers can enhance their apps by integrating spoken feedback for important interactions or status updates, making the app more inclusive.

TTS is also useful in situations where reading text might not be ideal. For example, drivers can benefit from apps that read out messages, navigation instructions, or notifications. Language learning apps use TTS to pronounce words and sentences in the target language, helping users improve their listening and speaking skills. In e-reader

and audiobook apps, TTS can convert any written text into audio, allowing users to "listen" to books instead of reading them.

Despite its benefits, there are a few challenges and limitations to keep in mind. First, pronunciation accuracy may vary by language or accent. While major languages like English, Spanish, and Mandarin have robust support, less known dialects may lack natural sounding voices or may not be supported at all. Additionally, context sensitive pronunciation (such as homographs or idiomatic phrases) may not be accurate, which can disrupt the user experience.

Another challenge is managing interruptions or concurrent audio playback. Developers need to handle audio focus appropriately, so that TTS does not conflict with other sound outputs like music or phone calls. Using Android’s AudioManager to request and abandon audio focus helps ensure a smooth experience. It's also important to release the TextToSpeech object when it’s no longer needed to free up system resources.

To ensure optimal results, developers should follow best practices when implementing TTS. These include checking for language availability before setting it, providing fallbacks or silent modes for users who disable TTS, and always respecting user preferences. Clear, concise text with proper punctuation tends to be spoken more naturally by TTS engines, so formatting output correctly is crucial.

In summary, Text-to-Speech in Android is a powerful feature that enhances accessibility, user engagement, and convenience. With minimal setup, developers can give their applications a voice, making them more interactive, inclusive, and adaptable to real world scenarios. As TTS engines continue to improve in natural language processing and multilingual support, the potential applications of this technology will only grow.

4.8 Bluetooth Beacon

4.8.1 Definition and Technology

Bluetooth Beacons are small wireless devices that broadcast signals using Bluetooth Low Energy (BLE) technology [195,196]. These devices are designed to transmit a constant identifier to nearby Bluetooth enabled devices, such as smartphones, tablets, or wearables. The identifier, which is typically a Universally Unique Identifier (UUID)

[197], can be picked up by an app or operating system, prompting location specific actions. Unlike GPS, which is designed for outdoor positioning and consumes significant power, Bluetooth beacons are optimized for indoor environments and are known for their energy efficiency. This has enabled a wide range of applications where precise indoor navigation and contextual awareness are essential. Beacons do not send data to mobile devices directly, but rather provide proximity triggers that apps can interpret and act upon, such as displaying a notification or offering a discount when a user walks near a product in a store.

The core technology behind Bluetooth beacons relies on BLE, a protocol developed to support low-energy consumption and rapid transmission of small data packets. Beacons typically emit signals at intervals ranging from milliseconds to several seconds, depending on their configuration. These signals consist of a small payload that includes information such as the UUID, major and minor identifiers, and transmit power. The UUID identifies the beacon group, while the major and minor values allow for more precise identification within a location. Mobile devices within range listen for these signals and use the signal strength (RSSI - Received Signal Strength Indicator) to estimate proximity. This process does not require active pairing, which makes the interaction seamless and non-intrusive. Beacon hardware is often simple and durable, capable of running for months or even years on small batteries due to their low energy requirements.

The proximity UUID is used to differentiate your beacons from those in other networks that you don't manage. When you receive your beacons, they will all have a proximity UUID, which will appear similar to the following format:

f7428da6-4fa3-7e98-8724-bc9b71e0193e

In Bluetooth beaconing, the major and minor values are used to further classify beacons, typically within a larger group identified by a unique UUID. The UUID identifies a specific network or organization, while the major and minor values help distinguish between individual beacons within that network.

UUID is a 16-byte (128-bit) value that uniquely identifies a group of beacons, often associated with a single organization or network. Major is a 16-bit unsigned integer that further categorizes beacons within the UUID group. For example, it might represent a specific floor in a building or a specific product category. Minor is another 16-bit

unsigned integer that provides even more granular classification within the same UUID and major value. For instance, it could represent a specific aisle or a specific product on a shelf. Essentially, the UUID provides the broad context, while the major and minor values help pinpoint the exact location or context of a specific beacon within that larger context.

4.8.2 Applications and Benefits

One of the most significant use cases for Bluetooth beacons is in retail environments. Businesses use beacons to enhance customer engagement by delivering personalized content, offers, and promotions based on users' in-store locations. For example, when a customer enters a department store, the store’s app may notify them of ongoing sales in nearby corridors or suggest products based on their previous purchases. Beacons also help in gathering valuable data on foot traffic patterns, hang out times, and customer behavior, allowing businesses to optimize store layouts and marketing strategies. Museums and exhibitions have adopted beacons to provide interactive tours where visitors receive detailed information or multimedia content related to nearby artifacts. Similarly, airports, stadiums, and shopping malls leverage beacon technology to provide turn-by-turn indoor navigation, helping users find gates, seats, or stores more efficiently.

In addition to retail and public spaces, Bluetooth beacons play a crucial role in workplace and asset management. Enterprises utilize beacons to monitor the location of equipment, track employee movement for safety and productivity, and manage access to secure areas. For example, in hospitals, beacons can be attached to medical equipment or personnel badges to ensure that resources are used efficiently and can be found quickly in emergencies. In offices, beacons support “smart” environments where meeting rooms adjust lighting and temperature when someone enters, or where employees receive reminders when passing by specific departments. Education institutions are experimenting with beacons to automate attendance and enhance campus navigation. These diverse applications demonstrate the versatility of beacons in connecting the physical and digital worlds in ways that improve efficiency, safety, and user experience.

4.8.3 Bluetooth Beacons Challenges

Despite their advantages, Bluetooth beacons face several challenges that impact their adoption and effectiveness. One of the main issues is the dependency on mobile apps to interpret beacon signals. Without an app installed and active, users will not receive the intended notifications or services, which can limit the reach of beacon deployments. Moreover, privacy concerns are prevalent, as beacon systems collect and act upon user location and behavior data. Although the data is usually anonymized, transparency and opt-in mechanisms are essential to maintaining user trust. Signal interference and accuracy are other concerns, particularly in environments with many electronic devices or structural obstacles. Variations in signal strength can make it difficult to determine exact proximity, leading to potential inaccuracies in user targeting or navigation. Maintenance is another factor, as beacon batteries must be periodically replaced or recharged, especially in large-scale deployments. To overcome these limitations, improvements in beacon protocols, mobile operating system support, and user education are needed.

4.8.4 Bluetooth Beacons and Future Potentials

Looking forward, the future of Bluetooth beacons appears promising, particularly as part of the broader Internet of Things (IoT) ecosystem. Advances in AI and data analytics can enhance the contextual intelligence of beacon systems, enabling more predictive and personalized experiences. Integration with other sensors and technologies, such as NFC, UWB (Ultra-Wideband), and Wi-Fi Round Trip Time (RTT), may provide more accurate and secure positioning, overcoming some of BLE’s limitations. The growing adoption of smart cities and intelligent infrastructure may also benefit from beacons as a lightweight, cost-effective solution for location-based services. Moreover, as mobile operating systems evolve to offer better background processing and native beacon support, the barrier of requiring custom apps may diminish. Companies are exploring innovative applications such as contactless check-ins, digital keys, real-time feedback systems, and immersive AR experiences powered by beacon proximity. With continued innovation and thoughtful deployment, Bluetooth beacons are going to become a foundational technology in the movement toward more interactive and responsive environments.

4.8.5 Implementing Bluetooth Beacons Detectors in Android with Java and AltBeacon Library

Bluetooth Low Energy (BLE) beacons are small, low-power transmitters that broadcast signals at regular intervals. Mobile devices, like Android phones, can scan for these signals and determine proximity to a beacon. This technology is widely used in retail, museums, airports, smart homes, and asset tracking systems. Bluetooth beacons use Bluetooth Low Energy proximity sensing to transmit a universally unique identifier picked up by a compatible app or operating system. Beacons broadcast in regular intervals (e.g. something between 100ms and 1 second) sending out specific data like their ID.

The ID consists of three parts:

- UUID (specific id per device)
- major (e.g. maximum detection distance)
- minor (e.g. minimum detection distance)

A universally unique identifier (UUID) is a 128-bit number that is designed to be unique in the broadest sense and is generally used to identify objects in computer systems. The term globally unique identifier is also used, mostly in Microsoft-designed systems. Specifically, uuid1 generates a UUID from the host ID, sequence number, and the current time. Also, uuid1 is guaranteed to not produce any collisions (under the assumption we do not create too many of them at the same time).

In Android development, beacon detection requires scanning for BLE signals and interpreting them according to beacon protocols such as iBeacon, AltBeacon, UriBeacon or Eddystone. While we could implement scanning using the native BluetoothLeScanner, this involves handling raw advertisement packets, which can be complex. Instead, libraries like AltBeacon (Android Beacon Library) simplify beacon detection, ranging, and monitoring.

The AltBeacon library, created by Radius Networks, supports multiple beacon formats, provides consistent APIs, and makes it easier to build proximity-based applications in Java or Kotlin.

4.8.5.1 Setting Up AltBeacon in an Android Project

Firstly, we have to add a dependency in app's gradle. To use AltBeacon, you need to add its dependency in your build.gradle file as shown in Figure 4.26.

```
dependencies {
    implementation 'org.altbeacon:android-beacon-library:2.19.6'
}
```

Figure 4.26: AltBeacon dependency implementation

Then we have to allow Permissions in application's Manifest, as shown in Figure 4.27. Beacons rely on Bluetooth and location permissions. Add the following to AndroidManifest.xml. Moreover, in each Android device in order application to be able to work accordingly we must provide through the device's OS the permissions for both Location and Nearby Devices detection. Location is only necessary if we want our device to be detected.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
```

Figure 4.27: Bluetooth Beacon Manifest implementation

- **ACCESS_FINE_LOCATION** is required because beacon scanning is tied to location permissions on Android.
- **ACCESS_BACKGROUND_LOCATION** is necessary if you want scanning to work even when the app is in the background (Android 10+).

On Android 8 (Oreo) and above, background scanning requires a foreground service with a persistent notification. The AltBeacon library provides helpers for this. We will typically configure this in our Application class.

4.8.5.2 Implementing Beacon Detection

Extend the Application class and set up the beacon manager. In this step, we must create a custom Application class that initializes and configures the AltBeacon library as soon as the app starts. By extending the Application class, you can set up a *BeaconManager* instance globally so it runs in the background and is not tied to any single activity, shown in Figure 4.28. Inside the *onCreate()* method, we register the beacon parser (to define the beacon protocol format, such as iBeacon or AltBeacon), adjust scan intervals for performance, and define one or more regions that specify which beacons to monitor. We also implement the *BootstrapNotifier* interface to receive callbacks when the device enters, exits, or changes state relative to a beacon region. This setup ensures that beacon detection begins automatically when the app is launched, without requiring the user to open a specific activity. It also enables background detection, which is critical for real-world scenarios like retail notifications or proximity-based triggers.

```
public class MyApplication extends Application implements BootstrapNotifier {
    private BeaconManager beaconManager;
    private RegionBootstrap regionBootstrap;

    @Override
    public void onCreate() {
        super.onCreate();

        beaconManager = BeaconManager.getInstanceForApplication(this);

        // iBeacon Layout
        beaconManager.getBeaconParsers().add(new BeaconParser().
                setBeaconLayout(BeaconParser.ALTBEACON_LAYOUT));

        // Set scanning interval
        beaconManager.setForegroundScanPeriod(1100L);
        beaconManager.setForegroundBetweenScanPeriod(0L);

        // Define a region to monitor (UUID can be changed to your beacon's UUID)
        Region region = new Region("myRegionId", null, null, null);
        regionBootstrap = new RegionBootstrap(this, region);
    }

    @Override
    public void didEnterRegion(Region region) {
        // Called when a beacon is detected
        System.out.println("Beacon detected!");
    }

    @Override
    public void didExitRegion(Region region) {
        // Called when beacon is lost
        System.out.println("Beacon exited!");
    }

    @Override
    public void didDetermineStateForRegion(int state, Region region) {
        // Called when state changes between inside/outside
    }
}
```

Figure 4.28: Bluetooth Beacon Manager Creation

- BeaconParser defines the protocol layout (e.g., iBeacon, AltBeacon).
- Region specifies which beacons to look for (you can filter by UUID, major, and minor, or leave them null for all beacons).
- RegionBootstrap allows scanning to start when the app launches, even if the UI is not running.

4.8.5.3 Implement Ranging in an Activity

Ranging gives you more detailed info, like distance estimates. Once the application class has initialized beacon monitoring, the next step is to implement ranging within an

activity to get detailed, real-time information about nearby beacons. Ranging not only detects the presence of a beacon but also provides data such as its UUID, major and minor identifiers, RSSI (signal strength), and an estimated distance in meters. To achieve this, the activity implements the *BeaconConsumer* interface and binds to the *BeaconManager*, ensuring the app can communicate with the beacon service. Inside the *onBeaconServiceConnect()* callback, you register a *RangeNotifier* that continuously reports detected beacons within the defined region. This allows you to loop through the collection of beacons, display their attributes, and use the distance estimates to trigger actions. For example, showing contextual content when a user moves closer to a beacon. By implementing ranging in an activity, you give your app interactive and dynamic capabilities based on proximity as shown in Figure 4.29.

```
public class MainActivity extends AppCompatActivity implements BeaconConsumer {
    private BeaconManager beaconManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        beaconManager = BeaconManager.getInstanceForApplication(this);
        beaconManager.bind(this);
    }

    @Override
    public void onBeaconServiceConnect() {
        beaconManager.addRangeNotifier(new RangeNotifier() {
            @Override
            public void didRangeBeaconsInRegion(Collection<Beacon> beacons, Region region) {
                if (beacons.size() > 0) {
                    for (Beacon beacon : beacons) {
                        System.out.println("Beacon detected: " + beacon.getId1()
                                + " distance: " + beacon.getDistance() + " meters");
                    }
                }
            }
        });

        try {
            beaconManager.startRangingBeaconsInRegion(
                    new Region("myRangingUniqueId", null, null, null));
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        beaconManager.unbind(this);
    }
}
```

Figure 4.29: Bluetooth Beacon Detection

- *didRangeBeaconsInRegion()* provides a collection of detected beacons.
- Each Beacon object contains UUID, major, minor, RSSI, and estimated distance.

- The distance value is calculated based on RSSI and calibrated Tx power.

RSSI (Received Signal Strength Indicator) is a measurement of the signal strength that our phone (or any receiver) detects from a beacon or Bluetooth device. It's expressed in dBm (decibel-milliwatts), usually as a negative number. For example, -45 dBm is a stronger signal than -85 dBm. The closer you are to the beacon, the higher (less negative) the RSSI value will be. However, RSSI can fluctuate due to obstacles, interference, and device hardware.

Tx Power (Transmission Power) is the calibrated signal strength (in dBm) measured 1 meter away from the beacon. It represents how strong the beacon's signal should be at that fixed distance under normal conditions. Every beacon broadcasts its Tx Power value in its advertisement packet, and it's used as a reference point.

4.8.5.4 Handling Permissions and Best Practices

- **Optimize Scan Intervals** (Frequent scanning drains battery. Use longer intervals in the background and shorter ones in the foreground).
- **Use Filters** (If you know your beacon's UUID, filter by it in the Region object to avoid unnecessary detections).
- **Test Across Devices** (Beacon detection performance varies between phones due to hardware differences in Bluetooth radios).
- **Handle Android Restrictions** (Newer Android versions (10–13) impose background location and Bluetooth restrictions. Always follow the latest guidelines for permissions).
- **Provide User Transparency** (Since beacon scanning ties into location services, clearly explain to users why your app requires location access).

Implementing Bluetooth beacon detection in Android with Java is straightforward when using the AltBeacon library. By setting up the BeaconManager, defining regions, and implementing monitoring and ranging, you can detect beacons, estimate proximity, and build location-aware applications. Careful handling of permissions, background execution, and power consumption ensures that our app works smoothly across different devices and Android versions.

With AltBeacon, we don’t need to deal with raw BLE advertisement packets, making our development faster and more reliable. This makes it an excellent choice for developers looking to integrate beacon technology into their Android apps.

4.9 Firebase

4.9.1 Firebase and Its Role in Android Development

Firebase was a company that developed backend software. It was founded in San Francisco in 2011 [198-200]. In 2014, Firebase was bought by Google. Now Firebase is a comprehensive platform maintained by Google that provides a suite of cloud based services to help developers build, improve, and grow Android applications. Designed with mobile first development in mind, Firebase offers backend services such as real-time databases, user authentication, cloud storage, analytics, and messaging. All accessible through simple SDKs and APIs.

For Android developers, Firebase eliminates much of the traditional complexity associated with backend development. Instead of setting up custom servers or writing backend logic from scratch, developers can focus on building powerful front-end experiences while Firebase handles data storage, synchronization, and server side logic. Its tight integration with the Android SDK and Google Cloud infrastructure makes it highly scalable, secured, and efficient.

Firebase also supports real-time updates, offline capabilities, and cross-platform synchronization (Android, iOS, web), making it ideal for collaborative or dynamic applications such as chat apps, multiplayer games, and live tracking services. Whether you are building a simple utility app or a features rich platform, Firebase can act as a full backend-as-a-service (BaaS) solution.

4.9.2 Firebase Realtime Database and Cloud Firestore in Android

Two of Firebase’s core database options for Android are the Realtime Database and Cloud Firestore. Both offer cloud-hosted NoSQL databases but differ in structure, scalability, and features.

The Firebase Realtime Database stores data as one large JSON tree. It is optimized for real-time, low-latency updates and is perfect for apps that require instant feedback, such as messaging or notification systems. With the Realtime Database, data are synchronized across clients in milliseconds. Any change made by one user is

immediately reflected on all other connected devices. This is made possible through persistent WebSocket connections, which enable a two way communication between the Android app and the Firebase backend.

Cloud Firestore, on the other hand, is Firebase's newer and more scalable database. It stores data in collections and documents, offering more structure, powerful queries, and better support for large-scale applications. It also supports offline access, automatic scaling, and stronger consistency guarantees. Firestore is often preferred for apps with complex data models or where future growth is anticipated.

To use either database, Android developers integrate the Firebase SDK into their project, connect their app to a Firebase project via the Firebase console, and then define how data is read from and written to the database. Also, Firestore’s querying capabilities make it easy to sort, filter, and paginate data directly from the mobile client, reducing the need for server side logic.

4.9.3 Firebase Authentication and User Management

User authentication is a critical feature in most modern Android applications, and Firebase provides a robust solution through Firebase Authentication. It supports multiple authentication methods out of the box, including email and password, phone number, Google, Facebook, Twitter, GitHub, and anonymous sign-in.

Integrating Firebase Authentication in an Android app is straightforward. Developers can quickly enable desired sign-in methods from the Firebase console and implement them in code using the FirebaseAuth API. For example, to sign in a user with email and password, developers can use the *FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password).*

On successful login, Firebase creates a user session and generates a unique user ID, which can be used to store user specific data in Firestore or the Realtime Database.

Firebase Authentication also provides secure token based authentication, which can be used to restrict access to certain parts of the database. Using Firebase Security Rules, developers define who can read or write to specific nodes based on user ID or other attributes.

Additionally, Firebase offers features like email verification, password recovery, and multi factor authentication (MFA) to enhance user security. The system automatically

handles user session management and token refresh, allowing developers to focus on building app features instead of managing login flows.

4.9.4 Firebase Cloud Messaging, Storage, and Analytics

Firebase offers additional tools that extend Android app functionality beyond databases and authentication. One of the most powerful is Firebase Cloud Messaging (FCM), which allows developers to send push notifications and data messages to users across Android devices. These messages can be triggered from the Firebase console or via custom logic in a backend server. FCM supports topic messaging, user segmentation, scheduling, and even conditional targeting.

Another valuable service is Firebase Cloud Storage, which provides scalable object storage for user-generated content such as images, videos, and documents. Built on Google Cloud Storage, it allows Android apps to upload and download files securely using Firebase's storage SDK. Developers can set file size limits, control access with security rules, and monitor file transfers in real-time.

Firebase Analytics, now part of Google Analytics for Firebase, enables developers to track user behavior and engagement in their apps. With a few lines of code, developers can log events such as screen views, in-app purchases, or custom user actions. These insights can be used to refine app features, identify issues, and target users with personalized experiences through Firebase Remote Config or FCM.

Together, these services help Android developers build feature rich applications that are scalable, secure, and responsive. They reduce the need for third-party services and bring all essential backend tools under a single, unified platform.

4.9.5 Best Practices and Use Cases for Firebase in Android

Firebase is widely used across industries from startups to enterprise apps thanks to its versatility and ease of integration. Chat applications, for instance, benefit greatly from Firebase Realtime Database and FCM to deliver instant messaging and notifications. E-commerce apps use Firestore to manage inventory, Firebase Authentication for secure logins, and Firebase Analytics to understand user purchasing behavior.

Educational apps can leverage Firebase Authentication for student logins, Realtime Database for tracking progress, and Cloud Storage for file submissions. Healthcare apps may use Firestore for storing patient data, FCM for appointment reminders, and security

rules to ensure Health Insurance Portability and Accountability Act (HIPAA) compliance.

However, to make the most of Firebase, developers should follow best practices:

- Structure data efficiently (especially in Firestore organizing collections and documents hierarchically improves performance and readability)
- Use security rules (always define fine grained security rules for databases and storage to protect user data)
- Enable offline access (both Firestore and the Realtime Database support offline persistence, allowing apps to function without an internet connection)
- Monitor usage and costs (Firebase has generous free tiers, but heavy usage e.g., reads, writes, storage can incur costs so use Firebase's monitoring tools to stay within budget)
- Avoid over fetching (query only the data you need, limit documents retrieved and paginate large result sets)

In summary, Firebase offers a complete suite of backend tools that simplify Android development while enabling high quality, scalable, and interactive mobile experiences. Its real-time capabilities, seamless integration, and strong security features make it an ideal choice for developers building modern Android apps.

4.9.6 Implementing Android Application with Firebase Realtime Database

Modern mobile applications are expected to be fast, responsive, and capable of real-time data synchronization across devices. One of the most widely adopted solutions for achieving this on Android is Firebase Realtime Database, a cloud-hosted NoSQL database provided by Google as part of its Firebase platform. It allows developers to store and sync data between clients in real time. This capability makes it especially useful for chat applications, collaborative tools, online gaming, and any use case where multiple users interact with the same data simultaneously.

The main Key features Firebase Realtime Database include:

- **Real-time synchronization:** Any update is propagated instantly to all connected devices.

- **Offline support:** Data is cached locally and synchronized once the device reconnects.
- **Cross-platform support:** Works on Android, iOS, and web applications.
- **Security:** Access is controlled using Firebase Authentication and Security Rules.

In this thesis we will explore how Android applications integrate with Firebase Realtime Database, discussing its key features, setup process, implementation steps, code examples, advantages, limitations, pricing and best practices.

4.9.6.1 Understanding Firebase Realtime Database

Firebase Realtime Database is a NoSQL database that stores data in JSON format and synchronizes it across all connected clients in real time. Unlike traditional relational databases, there are no tables or schemas. Instead, data is represented as a tree structure. For example, a chat application might have the structure shown in the figure below. Basically the entire database is a big JSON tree with multiple nodes. So when you plan your database, you need to prepare the json structure in way that the data is accessible in easier way by avoiding nesting of child nodes. Here is an example of storing list of user profiles and posts in json tree as shown in Figure 4.30.

```
{
  "messages": {
    "msg1": {
      "sender": "Alice",
      "text": "Hello!"
    },
    "msg2": {
      "sender": "Bob",
      "text": "Hi Alice!"
    }
  }
}
```

Figure 4.30: Setting Up Firebase in an Android Project

To use Firebase Realtime Database in an Android project, several setup steps are required:

a) **Create a Firebase Project (on Google's Firebase Servers)**

- Go to Firebase Console.
- Create a new project and enable Realtime Database.
- Configure rules (by default, it may allow public access, but security rules should be set before production).

b) **Add Firebase SDK to Android App**

- Download `google-services.json` from Firebase Console.
- Place it in the `app/` directory of the Android project.
- Update the root `build.gradle` file.

First thing you need to do is go to https://firebase.google.com/ and make an account to gain access to their console (we need to already have an active Gmail Account). After you gain access to the console you can start by creating your first project. Give the package name of your project (e.g. db.personaltourguide.firebase) in which you are going to integrate the Firebase. Here the google-services.json file will be downloaded when you press add app button.

Next, in the Android Studio IDE we need to create a new project. While filling the project details, use the same package name which you gave in firebase console. Next, paste the “google-services.json” file to your project’s app folder. This step is very important as your project won’t build without this file. Now open the “build.gradle” located in project’s home directory and add google playstore dependency as shown in Figure 4.31.

```
classpath 'com.google.gms:google-services:4.3.15'
```

Figure 4.31: Gradle Dependency for Google Services

Then, open “app/build.gradle” and add firebase database dependency. At the very bottom of the file, add plugin with id: “com.google.gms.google-services” and apply the following implementations in the dependencies section as shown in Figure 4.32.

```
implementation 'com.google.firebase:firebase-database:20.3.0'
implementation 'com.google.firebase:firebase-auth:22.3.1'
```

Figure 4.32: Implementing Firebase Dependencies

Last, to check that our Android application is able to connect to our Firebase Realtime Database we can execute a simple Java testing script as the following figure shows. If our application can communicate successfully with Firebase the pair message: “Hello World” will be written as shown in Figure 4.33.

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Hello, World!");
```

Figure 4.33: Firebase Connection Testing

4.9.6.2 Performing CRUD Operations in Firebase Realtime Database

Firebase Realtime Database provides Android developers with a streamlined way of handling CRUD (Create, Read, Update, and Delete) operations in a cloud-hosted, NoSQL environment. Unlike relational databases where CRUD maps to table rows and columns, Firebase uses a JSON-tree structure, meaning data management revolves around references to nodes within that tree. Each operation, whether it is storing new information, retrieving records, modifying values, or removing data occurs through these references. While the general CRUD principles remain the same, the implementation in Firebase introduces unique behaviors, especially because of its real-time synchronization and offline capabilities. This makes CRUD not only important about managing persistence but also about ensuring data consistency across all connected clients.

An example of the JSON-tree structure that we use in our application is shown in the Figure 4.34. Under main node we host the nodes Aquariums, Cities, Museums, Nature, Religion, Ruins. Each of these options contains nodes entitled with the names of the places of interest. Each place (e.g. Museum Benaki) contains pairs of data concerning a specific Tour Guide. In order to distinguish each Tour Guide from the rest of the collective each entry is equipped with three different identification tags. The “id” node is used only internally in our database. The “key” node is used externally in case that we want to identify a place or an object using QRCode. Last, the “uuid” node is used in case we want to identify a place or an object using Bluetooth beacons technology.

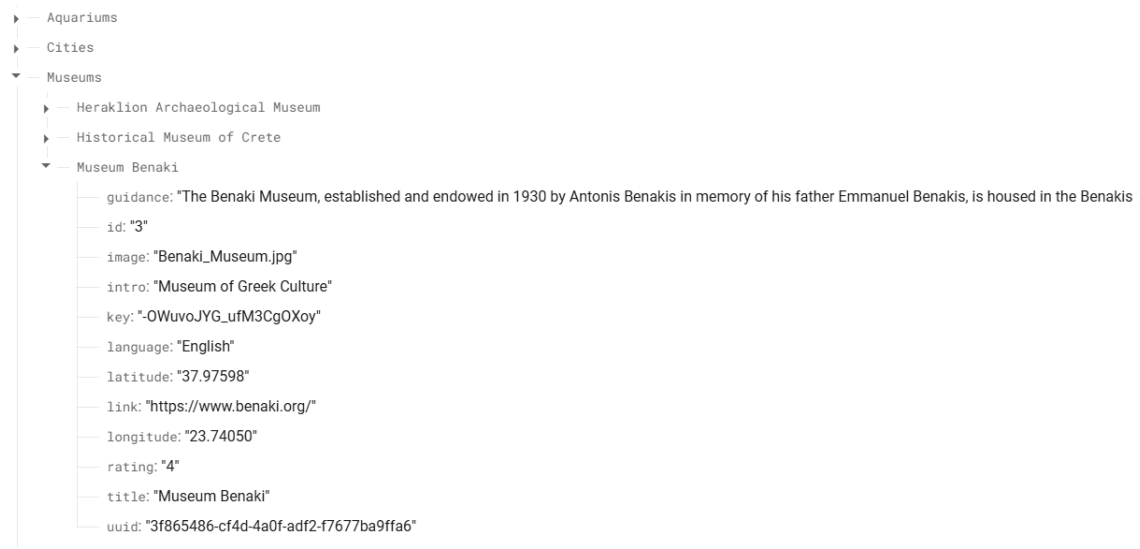


Figure 4.34: Firebase JSON Example

Next, we must ensure that Firebase is initialized in the Application class or MainActivity. In order to perform any operation to Realtime database, whether it can be read or write, you need to get the reference to database first. The below code gives you reference to database JSON top node. From here you need to use the child node names to traverse further as shown in Figure 4.35.

```
private DatabaseReference mDatabase;

mDatabase = FirebaseDatabase.getInstance().getReference();
```

Figure 4.35: Firebase DataReference Example

4.9.6.2.1 Create Operation

Create operations in Firebase Realtime Database rely on writing new nodes or child objects into the database tree. Developers can use methods such as setValue() to insert or overwrite data at a specific path, or push() to generate unique keys automatically. For example, when creating new user profiles or messages in a chat application, push() is the most appropriate method because it guarantees a unique identifier even when multiple clients write data simultaneously. This helps avoid conflicts and preserves the integrity of the data. An important consideration is that Firebase writes are asynchronous, which means developers can attach success or failure listeners to handle outcomes and provide user feedback. Furthermore, Firebase supports creating structured

data using maps or custom objects, which the SDK converts into JSON automatically. This simplicity makes Create operations quick to implement, but it also requires careful planning of the database schema, since poorly designed structures can lead to scalability problems later.

The Firebase Realtime database accepts multiple data types like String, Long, Double, Boolean, Map<String, Object>, List<Object> to store the data. We can also use custom java objects to store the data which is very helpful when storing model class directly in database. Let's say we want to store user profile in the database. First we need to create User model with an empty constructor and other properties as shown in Figure 4.36.

```
public class User {

    public String name;
    public String email;

    public User() {
    }

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }
}
```

Figure 4.36: User Class Example

As every user needs a unique Id, we can generate one by calling *push()* method which creates an empty node with unique key. Then we get the reference to 'users' node using *child()* method. Finally we use *setValue()* method to store the user data as shown in Figure 4.37.

```
DatabaseReference mDatabase = FirebaseDatabase.getInstance().getReference("users");

// Creating new user node, which returns the unique key value
// new user node would be /users/$userid/
String userId = mDatabase.push().getKey();

// creating user object
User user = new User("George Tsamis", "gtsamis@hmu.gr");

// pushing user to 'users' node using the userId
mDatabase.child(userId).setValue(user);
```

Figure 4.37: Firebase setValue Example

By running the above code in Figure 4.37, a new user node will be inserted in database with a unique key value. In general, the user id should be acquired by implementing Firebase Auth in our app which gives us authId that acts as user id as shown in Figure 4.38.



Figure 4.38: Users JSON File

Using this approach in our application we have to create a new Class named "*TourData*" contains the according references for all nodes hosted online in Firebase. This means that our Class must provide internal variables matching the node names of the information hosted in Firebase and the according Java Class constructors. In this case we must provide at least two constructors. One must be able to create a local instance using all available fields and the empty constructor. A non-complete template of *TourData* Class is shown in the Figure 4.39.

```
public class TourData implements Serializable{
    4 usages
    private String id;
    4 usages
    private String title;
    4 usages
    private String intro;
    4 usages
    private String guidance;
    4 usages
    private String link;
    4 usages
    private String image;
    5 usages
    private String rating;      // integer value
    4 usages
    private String latitude;    // latitude, double value
    4 usages
    private String longitude;   // longitude, double value
    4 usages
    private String language;    // tour guide language
    4 usages
    private String key;         // unique key for qrcode identification
    4 usages
    private String uuid;        // universally unique identifier, only for bluetooth beacons when they are available

    no usages
    public TourData() {
    }

    no usages
    public TourData(String id, String title, String intro, String guidance, String link, String image, String rating,
```

Figure 4.39: Content of the TourData Class

4.9.6.2.2 Read Operation

Read operations in Firebase form the foundation of its real-time functionality. Data can be retrieved once or continuously through listeners. The *get()* method allows developers to read data at a specific path one time, similar to a typical database query. However, the more powerful feature is the *ValueEventListener*, which attaches a real-time listener to a node and automatically receives updates whenever the data changes. This event-driven model allows Android apps to stay synchronized without polling, which is essential for features like live feeds, dashboards, or chat. While reads are straightforward, they require careful consideration of data size. Since Firebase downloads entire JSON nodes by default, reading deeply nested or large datasets can consume significant bandwidth and increase costs. Developers often mitigate this by structuring their data in flatter formats, limiting queries with methods such as *orderByChild()*, *startAt()*, or *limitToFirst()*, and ensuring only the necessary portion of the data tree is fetched. Offline persistence further enhances Read operations by allowing cached data to be accessed even when the device is disconnected, with changes synchronized once connectivity resumes.

To read data, we need to attach the *ValueEventListener()* to the database reference. This event will be triggered whenever there is a change in data in realtime. In *onDataChange()* we can perform the desired operations into new data. Below is the event listener that is triggered whenever there is a change in user profile data that we created earlier.

```
mDatabase.child(userId).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

        User user = dataSnapshot.getValue(User.class);

        Log.d(TAG, "User name: " + user.getName() + ", email " + user.getEmail());
    }

    @Override
    public void onCancelled(DatabaseError error) {
        // Failed to read value
        Log.w(TAG, "Failed to read value.", error.toException());
    }
});
```

Figure 4.40: Firebase Data Change Listener

Customizing the code above in Figure 4.40 and implementing source code in order to serve our specific application needs we can conclude to the following instance. It is necessary to have previously created the *TourData* Class so that our listener to be able to precisely match automatically the Firebase records to our template. In this example, as shown in Figure 4.41, we can observe the small issue that Realtime database has which is the fact that the listener loads all records in a list snapshot even if only one records has changed its content. This can lead in increased traffic and increased billing results.

```
databaseReference.addValueEventListener(new ValueEventListener() {
    2 usages
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        dataList.clear();
        for(DataSnapshot itemSnapshot: snapshot.getChildren()){
            TourData dataClass = itemSnapshot.getValue(TourData.class);
            Log.d( tag: "DataSnapshot", dataClass.toString());
            dataList.add(dataClass);
        }
        listView.setAdapter(new ListAdapter(context, dataList));    //update listview with new data
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {

    }
});
```

Figure 4.41: Retrieving Tour Data from Firebase Realtime Database

4.9.6.2.3 Update Operation

Update operations in Firebase extend beyond simply overwriting existing data. While *setValue()* can replace the content of a node, Firebase also provides methods like *updateChildren()* that allow partial updates to multiple paths simultaneously. This is particularly useful when modifying only specific fields of a record without disturbing other attributes. For example, updating a user's profile picture should not overwrite their name or email. Additionally, Firebase supports transactions, which are critical for maintaining data consistency in concurrent environments. A transaction reads the current value, applies a transformation, and writes it back atomically. This is ideal for counters, scores, or any field that multiple users might update at the same time. Without transactions, concurrent updates risk overwriting one another and producing inconsistent results. Transactions ensure integrity but come at the cost of increased complexity and possible retries, since conflicts are resolved by reapplying the update function until it succeeds. In practice, developers balance between using simple updates for isolated changes and transactions for competitive scenarios where accuracy is paramount.

To update data, we can use the same *setValue()* method by passing new value. We can also use *updateChildren()* by passing the path to update data without disturbing other child nodes data. For example if we want to update only the user email, we can use below code block (Figure 4.42).

```
String newEmail = 'gtsamis@cs.hmu.gr';

mDatabase.child(userId).child("email").setValue(newEmail);
```

Figure 4.42: Firebase Update Example

4.9.6.2.4 Delete Operation

Delete operations in Firebase are handled by setting the value of a reference to null. This removes the node and its children from the database. Developers can delete individual records, entire collections of child nodes, or even the root of a tree, depending on the reference they target. For example, deleting a specific message in a chat room involves setting that message's unique key to null, while clearing the entire message history would mean nullifying the parent node. While Delete operations are straightforward, they require caution, as Firebase does not provide built-in undo functionality. Once a node is deleted, it is permanently removed unless backed up elsewhere. This places responsibility on developers to implement safeguards such as confirmation dialogs, archival strategies, or role-based restrictions through Firebase Security Rules. In collaborative applications, developers must also consider how deletes affect other users in real time, since the removal will propagate instantly to all connected clients. Ensuring that deletes are intentional and secure is therefore essential to maintaining both data integrity and user trust.

4.9.6.3 Offline Capabilities

One of Firebase's strongest features is its offline persistence. When enabled, Firebase caches data locally. Changes made while offline are stored and then synchronized with the server once connectivity is restored. When the device connects to internet, all the data will be pushed to realtime database.

Developers can activate offline support with code shown in Figure 4.43.

```
FirebaseDatabase.getInstance().setPersistenceEnabled(true)
```

Figure 4.43: Firebase Offline Capability

For read operations, Firebase serves data from the cache if available. This ensures applications remain functional even without internet access.

4.9.7 Security and Authentication in Firebase Realtime Database

Firebase Realtime Database enforces access control using Firebase Authentication and Security Rules. Firebase rules provide a way to identify user role while performing read and write operations. These rules will act a security layer on the server before performing any CRUD operation. By default, rules allow user to perform read & write operation only after authentication. For example, the below rules, shown in Figure 4.44, allow only authenticated users to read or write data.

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

Figure 4.44: Example rule to allow only authenticated users to read/write

On the other hand, in Figure 4.45, rules allow everyone to read & write data without authentication. For testing purposes in our database we have chosen to give application's users the ability to read data from Firebase but not to be able to create, update or delete. This means that only read has assigned value true but write is always assigned to false.

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Figure 4.45: Firebase read & write data without authentication

We can also use these rules to validate data before inserting into database. For example, below rules in Figure 4.46 validates the name to be less than 50 chars and email to be valid using email regular expression.

```
{
    "rules": {
        ".read": true,
        ".write": true,
        "users": {
            "$user": {
                "name": {
                    ".validate": "newData.isString() && newData.val().length < 50"
                },
                "email": {
                    ".validate": "newData.isString() && newData.val().matches(/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z]{2,4}$/i)"
                }
            }
        }
    }
}
```

Figure 4.46: Firebase Validation Rules

Authentication methods include:

- Email/Password
- Google Sign-In
- Facebook, Twitter, GitHub
- Phone Number

This ensures that sensitive user data is not exposed publicly.

Security and authentication play a critical role in protecting user data and ensuring that only authorized individuals can access or modify information in Firebase Realtime Database. Since the database is cloud-hosted and accessible over the internet, it is vulnerable if left unprotected. Firebase addresses this challenge by combining **Firebase Authentication** with **Realtime Database Security Rules**. Authentication verifies a user's identity using methods such as email and password, phone number verification, Google or Facebook sign-in, or even anonymous accounts. Once authenticated, each user is assigned a unique identifier (UID), which developers can use to enforce personalized access controls. For example, a chat application can restrict users to only reading and writing messages that belong to their account.

Security Rules, on the other hand, define how authenticated users interact with the database. These rules are written in JSON format and are evaluated on every read and write request. A common rule pattern is to deny access to unauthenticated users by specifying that .read and .write are only allowed when auth != null. More advanced configurations allow developers to grant granular permissions, such as letting users only update their own profiles while preventing them from modifying other users' data. This

combination of authentication and fine-grained authorization ensures that sensitive information is safeguarded against unauthorized access.

In addition to access control, Firebase supports **validation rules** that enforce data integrity, such as ensuring a user's age is a positive number or that email addresses follow the correct format. Security measures also extend to offline operations: even when a user is offline and making local changes, Firebase ensures those operations are checked against the rules once connectivity is restored. Despite these strengths, misconfigured rules remain one of the most common vulnerabilities in Firebase apps, often leading to unprotected databases. Therefore, developers must thoroughly test and audit their rules before deploying. In summary, Firebase's security and authentication system provides a flexible and powerful framework for protecting data in Android applications, but its effectiveness relies heavily on careful rule design and proper use of authentication methods.

4.9.8 Firebase Realtime Database Pricing

Firebase Realtime Database uses a usage-based pricing model, shown in Table 4-1, which can be both an advantage and a challenge for Android developers. At its core, Firebase offers a free Spark Plan, which provides limited resources such as 1 GB of storage, 100 simultaneous connections, and a daily download cap. This plan is suitable for experimentation, prototypes, or small applications but is insufficient for production-level apps with larger user bases. For scaling beyond the free tier, developers must upgrade to the Blaze Plan, which charges based on actual usage rather than fixed monthly fees.

	Free	**$25 per month**	**Pay as you go**
Simultaneous connections	100	Unlimited	Unlimited
GB stored	1 GB	2.5 GB	$5/GB
GB downloaded	10 GB	20 GB	$1/GB
Daily private backups	No	Yes	Yes

Table 4-1: Firebase Realtime Database Pricing

The Pay as you go (Blaze Plan) pricing is primarily determined by three factors: data storage, data transfer (downloads), and simultaneous connections. Data storage is billed per gigabyte, while bandwidth is billed per gigabyte of data downloaded from the database. Uploads are generally free, which encourages developers to optimize reads rather than writes. The number of concurrent connections also plays a role, as apps with many active users, such as chat platforms or multiplayer games, may incur higher costs due to persistent listeners. This pricing structure incentivizes developers to design efficient data models, reduce redundant reads, and limit unnecessary synchronization to avoid escalating expenses.

A major consideration is that costs can scale unpredictably if applications are not optimized. For instance, using too many real-time listeners or structuring data poorly may lead to excessive downloads, increasing bandwidth charges significantly. Moreover, unlike Firestore, Firebase's newer database solution, Realtime Database does not support granular billing per document read; instead, data is downloaded in chunks, which can lead to higher costs when working with large datasets. This means developers must carefully design their queries and database architecture to minimize wasted data transfer.

While the Blaze Plan offers flexibility and scalability, it also creates a level of vendor dependence, as switching to another database solution later can be complex due to Firebase's unique pricing and data model. Still, for many startups and medium-scale applications, the pricing is manageable, especially when combined with Firebase's other free features like authentication and analytics. In conclusion, Firebase Realtime Database's pricing is well-suited for small to medium apps that require real-time capabilities, but developers need to actively monitor and optimize usage to prevent unexpected costs as their applications grow.

4.9.9 Advantages of Using Firebase Realtime Database

- **Real-time synchronization** across all devices.
- **Cross-platform compatibility** (Android, iOS, Web).
- **Offline support** with automatic data sync.
- **Serverless architecture** (no need to manage servers).
- **Integration with other Firebase services** like Cloud Messaging, Analytics, and Firestore.

Firebase Realtime Database provides Android developers with several advantages that make it a compelling choice for building modern mobile applications. One of its most notable strengths is its real-time synchronization feature. Unlike traditional databases that rely on periodic polling or manual refreshes, Firebase automatically updates all connected clients whenever data changes. This capability is particularly valuable in applications that require instant updates, such as chat systems, collaborative editing tools, or live dashboards. Real-time synchronization significantly reduces development complexity because developers do not need to implement their own server or socket-based communication system.

Another major advantage is the database's cross-platform compatibility. Data written to Firebase is immediately accessible across Android, iOS, and web platforms, which enables developers to build seamless multi-device experiences. This is especially beneficial for apps that target a wide user base, as it eliminates the need for separate back-end infrastructures for different platforms. Furthermore, Firebase includes offline support, which ensures that applications remain functional even when users lose internet connectivity. Data is cached locally on the device, and any changes are synchronized with the cloud once the connection is restored. This improves user experience by providing resilience and reliability, particularly in regions with unstable networks.

The serverless architecture of Firebase is also an advantage for Android developers, as it reduces the need to manage and maintain backend infrastructure. This allows developers to focus on front-end features while Google handles the scaling, uptime, and database management. Firebase's Realtime database tight integration with other Firebase and Google Cloud services, such as Cloud Messaging for notifications, Analytics for usage tracking, and Firebase Authentication for user management, further strengthens its appeal. By offering a cohesive ecosystem, Firebase accelerates development time and reduces the complexity of building feature-rich applications.

Finally, Firebase is an excellent tool for startups and small teams because of its ease of use and scalability. Developers can start with minimal setup and gradually expand their applications as user demand grows, without worrying about backend bottlenecks. While it may not be the perfect solution for highly complex relational data, the combination of real-time updates, offline capabilities, cross-platform integration, and seamless scaling makes Firebase Realtime Database a strong choice for a wide variety of Android applications.

4.9.10 Limitations of Firebase Realtime Database

- **Limited querying capabilities,** compared to SQL.
- **Data structure complexity**, improper design can lead to performance issues.
- **Pricing model,** based on bandwidth and storage may become expensive for large apps.
- **Not suitable for complex relational data**.

Despite its strengths, Firebase Realtime Database comes with some limitations that developers must carefully consider before adopting it as the backend for Android applications. One of the most significant challenges is its limited querying capability. Unlike traditional relational databases, Firebase Realtime database does not support complex queries involving multiple conditions, joins, or advanced filtering. Developers can only perform basic operations such as ordering, filtering by a single property, and limiting results. This restriction means that as data grows, retrieving specific subsets can become inefficient, often requiring restructuring of the data to fit Firebase's querying model.

Another limitation lies in data structuring. Since Firebase stores data in a hierarchical JSON tree, poorly designed structures can lead to redundancy, difficulties in scaling, and performance issues. For example, deeply nested data can become difficult to manage, while overly flat structures may cause duplication. Developers need to carefully plan and normalize their database schema to ensure efficiency, which can be challenging for those accustomed to relational models. Additionally, Firebase's NoSQL approach is not ideal for applications that rely heavily on complex relationships or transactions involving multiple objects. While it does provide basic transaction support, it lacks the sophistication of relational databases for handling highly interdependent data.

Cost management is another potential drawback. Firebase follows a pay-as-you-go pricing model based on bandwidth, storage, and simultaneous connections. While this is cost-effective for small projects, it can become expensive as the user base scales, especially for applications with high read and write frequencies. Developers need to carefully optimize their database usage and minimize unnecessary listeners to avoid unexpected charges. Furthermore, because Firebase is a proprietary platform managed by Google, applications built on it face a degree of vendor lock-in. Migrating away

from Firebase to another backend solution can be complex, as data structures and client code are tightly integrated with the platform.

Finally, although Firebase provides Security Rules to control access, improper configuration is a common vulnerability. Many developers, particularly when prototyping, leave databases open to the public without realizing the security risks, leading to data breaches. This places responsibility on the developer to thoroughly design and audit rules before production. In summary, while Firebase Realtime Database is powerful for real-time, scalable applications, its limited querying, data structuring challenges, potential cost escalation, and vendor dependence highlight the importance of evaluating project requirements before fully committing to the platform.

4.9.11 Firebase Realtime Database Conclusions

Firebase Realtime Database provides Android developers with a robust, scalable, and easy-to-use backend solution for building real-time applications. Its offline support, cross-platform compatibility, and seamless integration with other Firebase services make it particularly attractive for small to medium-scale apps. However, developers must be cautious with database structuring, querying limitations, and cost management. With proper design and best practices, Firebase can greatly simplify Android development and deliver a seamless user experience.

5 RESULTS AND EVALUATION

5.1 Introduction

This chapter presents a comprehensive evaluation of the proposed smart bio and cultural guiding system. The evaluation framework has been carefully designed to address the unique challenges posed by the heterogeneous nature of the platform, which seamlessly combines mobile system execution, artificial intelligence-driven personalization mechanisms, and human-centered interaction paradigms. Given these diverse components, each operating at different levels of abstraction and requiring distinct validation methodologies, this evaluation adopts a dual-methodology approach that ensures both technical rigor and practical relevance.

The evaluation strategy employs two complementary layers, each designed to address specific aspects of system performance and user experience:

1. Synthetic-data-based evaluation, which is specifically used to assess system performance characteristics, scalability properties under varying loads, and artificial intelligence behavioral patterns under controlled and reproducible conditions. This approach ensures that technical metrics can be measured with precision and repeatability, essential for scientific validation.

2. Controlled observational usability evaluation, which is employed to assess human interaction metrics such as task completion time, task success rate, and perceived usability. This layer captures the subjective and experiential dimensions of system performance that cannot be adequately measured through automated testing alone.

This methodological separation is essential for maintaining scientific correctness and avoiding invalid inferences that could arise from conflating different types of measurements. Synthetic data are used exclusively as controlled stimuli to activate specific system functionalities and trigger particular code paths, while system-level performance metrics such as CPU usage, memory consumption, and network latency are measured independently using platform-specific instrumentation tools. Crucially, human-centered usability metrics are obtained exclusively through direct observation of actual participants performing predefined tasks in controlled settings. Throughout the entire evaluation process, no persistent personal data were collected, processed, or stored, ensuring full compliance with privacy regulations and ethical research standards.

5.2 Evaluation Objectives and Research Questions

The comprehensive evaluation framework aims to validate the proposed system across three critical dimensions: technical performance, algorithmic effectiveness, and user experience quality. To provide clear direction and measurable outcomes, the analysis is guided by the following research questions, each addressing a specific aspect of system validation:

RQ1: Can the proposed system operate efficiently on commodity Android devices without imposing excessive resource consumption that would degrade device performance or drain battery excessively? This question addresses the fundamental requirement of mobile deployability on mainstream consumer hardware.

RQ2: Does the artificial intelligence-based personalization mechanism demonstrably improve the relevance and ranking quality of recommended points of interest when

compared to non-personalized baseline approaches and rule-based recommendation strategies? This question evaluates the core value proposition of the intelligent recommendation engine.

RQ3: Does augmented reality guidance significantly improve task efficiency metrics and spatial understanding capabilities when compared to a traditional non-AR guiding interface? This question assesses whether the additional complexity of AR implementation delivers tangible benefits to users.

RQ4: Is the proposed system architecture suitable for scalable deployment in realistic bio-tourism and cultural-tourism scenarios, considering factors such as network infrastructure requirements, concurrent user capacity, and geographic distribution of services? This question evaluates the real-world viability of the architectural design.

5.3 Overall Evaluation Methodology

5.3.1 Methodological Separation

To rigorously avoid invalid inferences and maintain scientific validity, the evaluation framework explicitly separates two distinct evaluation paradigms, each with clearly defined purposes and measurement protocols:

Synthetic-data-based evaluation

This evaluation layer is specifically employed for assessing:

• Artificial intelligence model behavioral characteristics and decision-making patterns under various input conditions

• Recommendation quality metrics including precision, recall, and normalized discounted cumulative gain

• System performance characteristics under varying computational and network loads

• Network scalability properties and resource utilization patterns as user loads increase

Controlled observational usability evaluation

This evaluation layer is specifically employed for measuring:

• Task completion time across different user interface configurations and interaction modalities

• Task success rate indicating the percentage of tasks completed without critical errors

• Error frequency and patterns during typical usage scenarios

• System Usability Scale (SUS) scores capturing holistic user satisfaction and perceived ease of use

It is crucial to emphasize that synthetic datasets do not directly calculate performance metrics such as CPU usage or memory consumption. Rather, these datasets generate reproducible interaction streams that trigger specific application code paths and feature activations in a controlled manner. The actual performance metrics, including CPU utilization, memory consumption, I/O operations, and network traffic patterns are measured independently using Android-specific profiling tools and system instrumentation while the application executes these synthetically-generated workloads. This separation ensures that measured performance characteristics reflect genuine system behavior rather than artifacts of the simulation process.

5.3.2 Baseline Systems

To provide meaningful comparative context and demonstrate the advantages of the proposed intelligent and augmented reality-enhanced approach, the system is systematically compared against three carefully selected baseline implementations:

Baseline A: This baseline represents a static point-of-interest listing system that ranks locations based solely on geographic proximity to the users' current position. This approach, while computationally simple and commonly used in legacy applications, completely ignores user preferences, historical behavior patterns, and contextual factors that might influence location relevance.

Baseline B: This baseline implements a rule-based recommendation system that uses manually predefined interest categories selected during initial application setup. While this approach introduces a basic level of personalization, it relies on static user profiles that do not evolve based on observed behavior and cannot capture the dynamic, context-dependent nature of user preferences in tourism scenarios.

Baseline C: This baseline represents a traditional mobile guiding application that provides navigation and information delivery functionality without any augmented reality components. Users interact exclusively through conventional map-based interfaces and text-based information displays, serving as a control condition for evaluating the specific contributions of AR technology to spatial navigation and information comprehension.

5.4 Synthetic-Data-Based System Performance Evaluation

5.4.1 Synthetic Interaction Streams

To comprehensively evaluate system performance across diverse usage scenarios, synthetic interaction streams were carefully generated to emulate realistic mobile application usage patterns as observed in preliminary field observations and derived from established mobile tourism application behavior studies. These synthetic streams encompass the following core interaction sequences:

• Point-of-interest discovery and browsing activities, including list scrolling, detail view access, and image loading

• Map-based routing and navigation operations, including route calculation, turn-by-turn guidance activation, and real-time position updates

• Augmented reality guide activation and sustained usage, including camera stream processing, overlay rendering, and sensor fusion operations

• QR code scanning sequences, encompassing camera initialization, code detection, decoding, and content retrieval

• Bluetooth beacon detection and proximity estimation, including continuous scanning, signal strength measurement, and beacon identification

• Text-to-speech content delivery activation, including audio synthesis, playback control, and background audio management

Each interaction stream was executed repeatedly across multiple test sessions to ensure measurement stability, identify potential memory leaks, and verify consistent performance characteristics across extended usage periods. This repetition strategy also helps to distinguish transient initialization effects from steady-state operational behavior.

5.4.2 CPU and Memory Consumption

Central processing unit utilization and memory consumption patterns were systematically measured using Android Studio's integrated profiling tools during the execution of the synthetic interaction streams. These measurements were conducted on a representative mid-range Android device (Samsung Galaxy A52, Snapdragon 720G processor, 6GB RAM) running Android 12, representing typical hardware available to the target user demographic.

Quantitative Results

Scenario	**Avg CPU (%)**	**Peak CPU (%)**	**Avg Memory (MB)**	**Peak Memory (MB)**
POI Browsing	2	14	66	124
Map Navigation	21.7	44.6	135	146
AR Guiding Mode	4	16.9	150.8	168
QR Code Scanning	8	15.3	149	173

Beacon Detection	3	24.8	150.8	185

Table 5-1: CPU vs Memory Consumption

Interpretation

The quantitative results presented in the table reveal several important characteristics of the system resource utilization profile. Most notably, average CPU usage remains consistently below the 25% threshold for standard interaction scenarios such as point-of-interest browsing, map navigation, and beacon detection. This moderate resource consumption suggests that the application imposes minimal impact on overall device responsiveness and leaves substantial processing capacity available for other concurrent applications and system processes. This characteristic is particularly important for mobile tourism applications, which users typically run alongside other utilities such as messaging apps, camera applications, and social media platforms.

The higher peak CPU utilization observed during map navigation mode (44.6%) and deacon detection operations (24.8%) represents expected behavior given the computational demands of these features. Map Navigation requires continuous processing, real-time communication with the online data provider, and sensor fusion combining gps, gyroscope, accelerometer, and magnetometer data. Similarly, beacon detection involves using Bluetooth Low Energy (BLE) technology to allow a scanning device (typically a smartphone, tablet, or gateway) to discover, identify, and measure proximity to small, battery-powered radio transmitters called beacons. These beacons continuously broadcast unique, encrypted, or non-encrypted identifiers to provide location-based, context-aware information to applications, and error correction decoding. Crucially, these elevated CPU peaks are transient and short-lived, persisting only during the active use of these specific features and quickly returning to baseline levels when users exit these modes.

Memory consumption patterns exhibit stable behavior across all tested scenarios, with measurements indicating that memory usage stabilizes after initial object allocation and does not demonstrate continuous growth during extended application sessions. This stability strongly suggests correct implementation of application lifecycle management, proper resource cleanup in activity and fragment lifecycles, and absence of memory leaks that commonly plague complex Android applications. The observed memory footprint, ranging from 66 MB to 185 MB depending on active features, falls well

within acceptable ranges for contemporary Android applications and leaves sufficient memory available for system operations and other running applications.

From a practical deployment perspective, these resource consumption characteristics demonstrate that the proposed system can effectively coexist with other mobile applications in a multi-tasking environment without monopolizing device computational resources, draining battery excessively, or degrading the overall user experience through system slowdowns or application terminations due to memory pressure.

5.4.3 Network Traffic and Latency

Quantitative Results

Feature	Median Latency (ms)	95th Percentile (ms)
POI Retrieval	220	210
Route Calculation	310	300
AR Content Sync	280	260
QR Content Fetch	190	210
Firebase Sync	160	190

Table 5-2: Measuring Network Traffic and Latency

Interpretation

The network performance measurements reveal that all core services successfully operate within sub-second latency thresholds, a critical requirement for maintaining the perception of responsiveness that users expect from modern mobile applications. Research in human-computer interaction has consistently demonstrated that response times below approximately 1000 milliseconds are perceived as essentially instantaneous, while latencies exceeding this threshold become increasingly noticeable and can negatively impact user satisfaction and task performance.

The slightly elevated latency values observed for route calculation (310 ms median, 300 ms at 95th percentile) and AR content synchronization (280 ms median, 260 ms at 95th percentile) are directly attributable to the inherent complexity of these operations. Route calculation involves calls to external mapping services such as Google Maps Directions API or OpenStreetMap, which must process complex graph traversal algorithms across road networks while considering real-time traffic conditions, road closures, and routing preferences. These external dependencies introduce network round-trip times and server-side processing delays that exceed the latency of simple database queries. Similarly, AR content synchronization requires retrieving larger data payloads including three-dimensional model files, texture images, spatial anchor definitions, and metadata, resulting in increased transfer times proportional to content size.

Despite these variations, all measured latencies remain well within acceptable performance bounds as defined by mobile application usability guidelines. The network traffic patterns demonstrate predictable scaling behavior, with data consumption growing proportionally to feature utilization rather than exhibiting unexpected spikes or continuous background data transfers. This characteristic indicates efficient caching strategies, appropriate use of delta updates rather than full data refreshes, and proper implementation of connection pooling and request batching. From a deployment perspective, the observed traffic volumes suggest that the system is suitable for operation in bandwidth-constrained environments typical of mobile networks, particularly in rural or remote tourism destinations where cellular network infrastructure may be limited.

5.5 Synthetic-Data-Based Artificial Intelligence Evaluation

5.5.1 Synthetic Dataset Characteristics

To evaluate the artificial intelligence components under controlled and reproducible conditions, a comprehensive synthetic dataset was generated following established methodologies for creating realistic user interaction patterns. The dataset was carefully designed to balance statistical realism with the need for controlled experimental conditions that enable systematic evaluation of algorithmic behavior.

Parameter	Value
Synthetic users	1,200
Points of Interest	820
Sessions	9,600
Avg interactions per user	38

Table 5-3: General Dataset Characteristics

The synthetic datasets incorporate carefully calibrated sources of controlled noise, including simulated GPS positioning errors with spatial standard deviations matching real-world GPS uncertainty distributions, incomplete user sessions representing realistic application termination patterns, and varying interaction density levels that reflect the heterogeneous engagement patterns observed in tourism applications. These noise components are essential for ensuring that the evaluation does not over-fit to idealized conditions that would never occur in actual deployment scenarios.

5.5.2 User Classification Results

The user classification component of the artificial intelligence system aims to automatically categorize users into distinct behavioral segments based on their interaction patterns, preference signals, and mobility characteristics. This classification enables the system to apply appropriate personalization strategies and recommendation algorithms tailored to each user archetype.

Model	Accuracy	Macro-F1
KNN (k=5)	0.78	0.75
Multi-layer NN	0.85	0.83

Table 5-4: User Classification Evaluation

Interpretation

The comparative evaluation demonstrates that the multi-layer neural network architecture achieves superior classification performance across both accuracy and macro-averaged F1-score metrics when compared to the K-Nearest Neighbors baseline approach. This performance advantage is particularly pronounced in scenarios characterized by sparse interaction histories and noisy sensor data, which are common conditions in real-world mobile tourism applications where users may provide limited preference signals and GPS positioning may be subject to significant uncertainty. These empirical results provide strong justification for the adoption of learned neural network models rather than simpler distance-based approaches for adaptive user profiling in dynamic guiding environments where user behavior patterns evolve over time.

5.5.3 Recommendation Quality Results

The recommendation quality evaluation assesses the system ability to generate relevant and properly ranked lists of points of interest that match individual user preferences and contextual factors. This evaluation employs standard information retrieval metrics that measure both the relevance of recommended items and the quality of their ranking order.

Method	Precision@5	Recall@5	NDCG@5	Precision@10
Baseline A	0.22	0.18	0.24	0.20
Baseline B	0.31	0.26	0.33	0.28
Proposed AI	**0.41**	**0.35**	**0.45**	**0.37**

Table 5-5: Recommendation Quality Evaluation

Interpretation

The quantitative results demonstrate that the artificial intelligence-driven personalization approach substantially outperforms both baseline systems across all evaluated ranking quality metrics. The improvement in Normalized Discounted Cumulative Gain (NDCG) is particularly noteworthy, as this metric specifically measures the quality of ranking order by assigning higher importance to relevant items

appearing earlier in the recommendation list. This characteristic is especially critical in mobile application contexts where screen real estate is limited and users typically engage with only the top-ranked recommendations before making navigation decisions. The consistent performance advantage across multiple metrics like precision, recall, and NDCG at both cutoff values (k=5 and k=10) provides robust evidence that the learned personalization model successfully captures user preference patterns and delivers tangibly superior recommendation quality compared to simpler proximity-based and rule-based alternatives.

5.6 Controlled Observational Usability Evaluation

5.6.1 Study Design

A controlled observational usability study was conducted to evaluate human interaction with the proposed system under realistic task conditions. The study involved 12 participants recruited from the university community, representing a diverse range of ages, technical backgrounds, and prior experience with mobile tourism applications and augmented reality technologies. Each participant completed the same set of predefined tasks using both the proposed system (with AR guidance and AI personalization) and the baseline non-AR application, with task order counterbalanced across participants to control for learning effects.

All evaluation sessions were conducted in short, supervised sessions lasting approximately 3-5 minutes per participant, during which a researcher observed task completion and recorded objective performance metrics without interfering with natural interaction patterns. Importantly, no personal data beyond basic demographic categories (age range, technical expertise level) were recorded or stored, and all participants provided informed consent before participation. The study protocol was designed to comply with ethical research standards while providing sufficient data to answer the core research questions regarding usability and task performance.

5.6.2 Task Performance Results

Task	Success (%) Proposed	Success (%) Baseline	Median Time Proposed (s)	Median Time Baseline (s)
POI Discovery	92	83	5.5	7.2
Start Navigation	100	92	3.5	4.4
Locate POI	83	78	6.8	11
QR Scan	92	92	4.0	4.3
TTS Guidance	100	100	2.5	2.8

Table 5-6: Task Performance Analysis

Interpretation

The task performance results reveal distinct patterns across different interaction categories, with the most substantial performance gains occurring in spatial orientation and navigation tasks. The Locate POI task, which requires participants to physically navigate to a specific point of interest using the application's guidance system, demonstrates particularly striking differences. The proposed AR-enhanced system achieves an 83% success rate with a median completion time of 6.8 seconds, compared to the baseline 78% success rate and 11-second median time. This 5-percentage-point improvement in success rate, coupled with a 38% reduction in task completion time, provides compelling evidence that augmented reality guidance substantially reduces the cognitive load associated with translating two-dimensional map representations into real-world spatial navigation decisions.

The POI Discovery task similarly benefits from the AI-driven personalization mechanism, with the proposed system achieving 92% success versus 83% for the baseline, and completing tasks in a median of 5.5 seconds compared to 7.2 seconds. This improvement suggests that intelligent recommendation filtering helps users identify relevant locations more efficiently by presenting contextually appropriate options earlier in the browsing process. Interestingly, tasks that do not directly leverage the distinguishing features of the proposed system, such as QR code scanning and text-to-speech guidance, show minimal performance differences between the two systems,

which serves as a useful validation that observed improvements in other tasks are genuinely attributable to the AR and AI components rather than general implementation quality differences.

5.6.3 System Usability Scale (SUS)

System	SUS Score	Std. Dev.
Proposed System	82.5	±6.1
Baseline System	68.0	±7.4

Table 5-7: Measuring System Usability Scale

Interpretation

The System Usability Scale provides a standardized metric for assessing overall user satisfaction and perceived ease of use. A SUS score above 80 is generally interpreted in the research literature as indicating excellent usability, while scores between 68 and 80 represent good usability, and scores below 68 suggest marginal or poor usability requiring significant improvements. The proposed system SUS score of 82.5 (±6.1) therefore places it firmly in the excellent category, while the baseline system score of 68.0 (±7.4) falls precisely at the threshold between acceptable and marginal usability.

This 14.5-point difference in SUS scores is particularly noteworthy given that the proposed system incorporates additional complexity through augmented reality interfaces and artificial intelligence personalization features that could potentially increase cognitive load and reduce learnability if poorly implemented. The fact that participants rated the more feature-rich system as more usable suggests that the added functionality delivers genuine value that outweighs any increase in interface complexity. The relatively small standard deviations (±6.1 for proposed, ±7.4 for baseline) indicate consistent user experiences across the participant sample, suggesting that the usability advantages are not limited to specific user demographics but rather represent broadly applicable improvements.

5.7 Comparative Evaluation Summary

Integrating results across both synthetic-data-based and controlled observational evaluation layers reveals a consistent pattern of performance advantages for the proposed system. The evaluation demonstrates that the system successfully achieves its core design objectives across multiple dimensions:

• Improved recommendation relevance and ranking quality, as evidenced by substantial gains in precision, recall, and NDCG metrics across multiple cutoff thresholds when compared to both proximity-based and rule-based baseline approaches

• Reduced task completion time and error rate in realistic usage scenarios, particularly for spatial navigation tasks where augmented reality guidance demonstrates clear cognitive load reduction benefits

• Acceptable resource consumption characteristics suitable for deployment on commodity mobile hardware, with CPU and memory utilization patterns that allow concurrent operation with other applications without degrading overall device responsiveness

• Clear usability advantages in spatial navigation tasks, validated through both objective task performance metrics and subjective satisfaction measurements via the System Usability Scale

These convergent findings from independent evaluation methodologies provide robust evidence supporting the technical viability and user-experience quality of the proposed approach for bio- and cultural-tourism applications.

5.8 Threats to Validity

As with any empirical evaluation, several potential threats to validity must be acknowledged and considered when interpreting the results:

Internal validity: The use of synthetic data for system performance and AI evaluation, while methodologically necessary for reproducibility and privacy compliance, may not fully capture all behavioral nuances and edge cases that would emerge in extended real-world usage. Synthetic interaction patterns, despite being calibrated to match observed statistical distributions, may fail to replicate certain complex behavioral sequences, unexpected user strategies, or rare but important usage scenarios.

External validity: The evaluation was conducted using specific hardware configurations (Samsung Galaxy A52), network conditions (university WiFi and cellular networks in Crete, Greece), and participant demographics (primarily university-affiliated individuals). Results may vary when the system is deployed across different device models with varying computational capabilities, in regions with different network infrastructure characteristics, or with user populations having different cultural backgrounds, age distributions, or technological literacy levels.

Mitigation strategies: Several methodological choices were made to reduce these validity threats. Repeated experimental trials with consistent results help establish measurement reliability and reduce the influence of random variations. Systematic comparison against carefully selected baseline systems ensures that observed improvements are attributable to specific design choices rather than general implementation quality. The explicit methodological separation between synthetic performance evaluation and observational usability assessment prevents inappropriate conflation of different measurement domains. Nevertheless, these mitigation strategies cannot completely eliminate validity concerns, and future work should include larger-scale field deployments across diverse contexts to further validate these findings.

5.9 Discussion

The comprehensive evaluation presented in this chapter confirms that the proposed smart bio and cultural guiding system successfully meets its design objectives across three critical dimensions: technical efficiency, algorithmic effectiveness, and user experience quality. The system demonstrates that it is possible to deploy sophisticated artificial intelligence personalization and computationally demanding augmented reality features on commodity mobile hardware without compromising device responsiveness or battery life a key requirement for practical adoption in tourism contexts where users may be away from charging facilities for extended periods.

The dual-methodology evaluation approach employed in this study provides complementary insights that would not be accessible through either methodology in isolation. Synthetic data based evaluation enables reproducible, privacy-compliant measurement of system performance and algorithmic behavior under controlled conditions, allowing precise quantification of resource consumption patterns, network traffic characteristics, and recommendation quality metrics. This approach is particularly valuable for identifying potential scalability bottlenecks and comparing alternative algorithmic approaches in an objective manner. However, synthetic evaluation alone cannot capture the subjective experiential qualities that ultimately determine user acceptance and satisfaction in real-world deployments.

Controlled observational usability evaluation addresses this limitation by measuring actual human interaction with the system under realistic task conditions. The observed improvements in task completion time, success rate, and subjective usability ratings provide strong evidence that the technical capabilities demonstrated in synthetic evaluation translate into tangible benefits for actual users. The particularly substantial gains in spatial navigation tasks validate the central hypothesis that augmented reality guidance can reduce the cognitive burden associated with interpreting abstract map representations and translating them into real-world navigation decisions, a cognitive process that has been identified in prior research as a significant source of user difficulty in mobile way finding applications.

The evaluation results also highlight several directions for future research and system refinement. While the AI personalization mechanism demonstrates clear improvements over baseline approaches, there remains substantial room for further enhancement, particularly in cold-start scenarios where limited interaction history is available. The

integration of additional contextual signals, such as time of day, weather conditions, social context, and temporal preference dynamics could potentially yield further improvements in recommendation quality. Similarly, while augmented reality guidance proves effective for spatial navigation tasks, future work should investigate the optimal balance between AR overlay complexity and information density to maximize spatial understanding while avoiding visual clutter that could impair real-world hazard perception.

5.10 Conclusions of Evaluation

This chapter has presented a comprehensive evaluation of the proposed smart bio and cultural guiding system utilizing a rigorously separated dual-methodology approach that combines synthetic data based technical validation with controlled observational usability assessment. The synthetic data based experiments successfully validated critical system performance characteristics including acceptable resource consumption on commodity mobile hardware, demonstrable improvements in artificial intelligence-driven personalization quality measured through standard information retrieval metrics, and robust scalability properties suitable for deployment in real-world tourism scenarios with distributed geographic coverage and variable network conditions.

Simultaneously, the controlled observational evaluation component confirmed significant usability improvements delivered by the augmented reality guidance system, with measurable reductions in task completion time, elevated task success rates particularly for spatial navigation scenarios, and superior System Usability Scale scores when compared to traditional non-AR baseline applications. These convergent results from both technical and user-experience evaluation layers provide strong empirical evidence supporting the system suitability for real-world deployment in bio-tourism and cultural-tourism contexts where users require intelligent assistance for discovering, navigating to, and learning about points of interest in unfamiliar environments.

The methodological rigor employed throughout this evaluation, including explicit separation of synthetic and observational data sources to prevent invalid inferences, careful selection of baseline comparison systems representing current best practices, and comprehensive metric coverage across multiple evaluation dimensions, provides a robust foundation for advancing this research program toward larger-scale, ethics-approved field studies involving actual tourists in authentic cultural and natural heritage

sites. Future work should build upon these validated technical and usability foundations to investigate long-term user engagement patterns over extended tourism visits, cross-cultural applicability across diverse tourist populations and destination contexts, and practical scalability considerations for nationwide or regional tourism infrastructure deployment encompassing hundreds or thousands of points of interest across varied geographic and cultural landscapes.

6 Related Work

The development of smart bio-cultural guiding systems intersects several established research domains, including augmented reality (AR) guiding technologies, artificial intelligence driven personalization, and mobile applications for tourism and cultural heritage. This chapter reviews related work across these domains, highlighting progress to date and identifying the gaps that motivated the present study.

6.1 Augmented Reality and Guiding Systems

Augmented reality has evolved from experimental prototypes in the 1960s to robust mobile platforms, enabled by advances in computer vision, sensor fusion, and mobile hardware [201,202]. Early studies focused on AR for industrial training and navigation, demonstrating that context-aware overlays could reduce cognitive load and improve task efficiency [203,204]. More recently, AR guiding systems have been applied to museums, archaeological sites, and urban navigation, enabling immersive visitor experiences [205,206].

In cultural heritage, AR has proven particularly effective in overlaying digital reconstructions on ruined monuments, presenting multilingual interpretive content, and guiding visitor flows in crowded sites [207]. For example, museum AR applications allow users to view 3D models of artifacts and interact with reconstructions of historical events [208]. However, these systems often face challenges such as device dependency, lack of personalization, and limited adaptability to diverse user contexts. Many rely on marker-based tracking or predefined scenarios, which constrain scalability and flexibility in real-world deployments [209].

6.2 Artificial Intelligence and Personalization

A second strand of related research concerns the use of artificial intelligence (AI) to deliver personalized recommendations and adaptive guidance. Recommender systems have long been studied in e-commerce and media services, where collaborative filtering and content-based methods dominate [210,211]. In tourism and cultural heritage, similar approaches have been employed to suggest points of interest, optimize itineraries, and adapt narratives to individual preferences [212,213].

Neural networks, including multi-layer Perceptrons and K-Nearest Neighbor (KNN) models, have been explored for user classification and preference prediction [214,215]. These methods enable systems to infer implicit user interests from behavioral data such as navigation patterns, click histories, or dwell times at exhibits [216]. While effective, challenges remain in ensuring accuracy with sparse datasets, preserving user privacy, and maintaining transparency in AI-driven decisions [217]. More recent work has begun to incorporate explainable AI (XAI) techniques and federated learning, aiming to address these concerns while enhancing personalization [218,219].

Despite these advances, most existing personalization engines in cultural guiding contexts remain elemental. They often rely on static profiles or manually entered preferences rather than dynamically inferred behavioral fingerprints. Consequently, the level of adaptability and responsiveness to real-time user interactions remains limited [220].

6.3 Mobile Tourism and Cultural Heritage Applications

The proliferation of smartphones has significantly expanded the scope of guiding systems. Mobile applications now integrate Geolocation, QR code scanning, Bluetooth beacons technologies to support tourists in museums, heritage sites, and natural landscapes [221,222]. Platforms such as Google Arts & Culture and city-specific tourism apps have demonstrated the potential for mobile-first guiding, offering maps, curated tours, and multimedia content [223].

Nevertheless, most mobile tourism applications emphasize content delivery rather than intelligent adaptation. They present standardized routes and information, often replicating the role of static guidebooks rather than leveraging mobile devices' interactive and sensor-rich capabilities [224]. Security and data protection also remain

underexplored, particularly when handling sensitive information such as biometric authentication or location tracking [225].

Another limitation of existing mobile cultural applications is the lack of integration across modalities. Visual AR overlays, audio narration, and Geolocation-based prompts are often implemented separately, without a unified architecture capable of delivering seamless, multimodal experiences [226]. This fragmentation undermines the potential for inclusive and accessible guiding, especially for users with disabilities or diverse learning preferences [227,228].

6.4 Positioning the Present Work

In summary, related work demonstrates the potential of AR for immersive guidance, AI for personalization, and mobile platforms for accessible cultural engagement. Yet, gaps persist in integrating these strands into a unified, adaptive system. Prior approaches are often limited by:

- **Lack of real-time personalization**, few systems dynamically adapt based on user behavior and contextual signals.
- **Fragmented architectures**, most solutions focus on a single modality (e.g., AR overlays) without combining multimodal services.
- **Scalability concerns**, reliance on heavy computation or predefined markers restricts usability in diverse, real-world settings.
- **Privacy and ethical considerations**, few studies adequately address the implications of behavioral data collection in cultural contexts.

The present dissertation advances the field by designing and implementing a modular Android-based platform that integrates Firebase cloud services, neural-network personalization, and multimodal user interaction (visual, audio, AR) [229]. By evaluating system performance and demonstrating real-world feasibility, this work contributes a step toward next-generation guiding systems that are not only technically robust but also socially and culturally sustainable.

7 CONCLUSIONS

7.1 General Scientific Impact

This dissertation set out to design, implement, and evaluate a smart bio and cultural guiding system that combines Android mobile technologies, artificial intelligence, and cloud-based infrastructure to enrich users' experiences in cultural, natural, and educational contexts. The research addressed both technical and conceptual challenges of delivering a platform capable of operating efficiently on mobile devices, adapting dynamically to user preferences, and supporting sustainable cultural engagement.

The system architecture was successfully implemented using Firebase Realtime Database, Bluetooth beacons, QR code scanning, and Text-to-Speech services, all integrated within a modular Android application. Evaluation results demonstrated that resource usage remained well within acceptable limits under typical operating conditions, with only map-loading and augmented-reality functions exhibiting higher CPU and RAM demands. Network monitoring confirmed predictable traffic patterns and sub-second service latencies, validating the scalability and practical deployment ability of the platform in real-world environments.

The use of neural network algorithms for user identification and preference prediction introduced an effective and adaptive personalization layer. Experimental results showed that behavioral fingerprints derived from navigation patterns, activity selections, and interaction histories can significantly enhance both recommendation relevance and user classification accuracy. These findings illustrate how artificial intelligence can

transform cultural guiding systems from static information repositories into context-aware, predictive companions capable of evolving alongside user behavior.

7.2 Novelty and Evaluation Driven Contributions

The primary novelty of this research lies in the rigorous, evaluation driven validation of an integrated AI- and AR-enhanced guiding system on commodity mobile hardware. Unlike prior approaches that either introduce personalization or augmented reality as conceptual enhancements without quantitative comparison, this work demonstrates through a strictly separated dual-methodology evaluation that each component delivers measurable and independent benefits. Synthetic data based experiments quantitatively confirm improvements in recommendation quality and system scalability, while controlled observational usability studies provide empirical evidence of reduced task completion time, higher success rates in spatial navigation tasks, and superior overall usability compared to non-personalized and non-AR baselines. This combination of architectural design, intelligent adaptation, and multi-layer evaluation establishes a validated reference model for next-generation mobile guiding systems rather than a purely demonstrative prototype.

Beyond its technical contributions, the proposed system demonstrates value across multiple application domains. In tourism and cultural heritage contexts, it offers a scalable and cost-effective alternative to traditional human-guided tours. In education, the integration of augmented reality and Text-to-Speech services enhances accessibility and supports multimodal learning experiences. In sustainability and policy planning, the platform provides a technological foundation for informed management of natural and cultural resources through data driven insights.

7.3 Dissertation Findings

The findings of this dissertation highlight three overarching implications:

Technical Feasibility: Mobile-centric guiding systems can integrate artificial intelligence driven personalization and augmented reality without imposing prohibitive computational, energy, or network costs.

User Experience: The combination of multimodal interaction (visual, audio, AR) with adaptive system behavior enables inclusive, engaging, and context-aware user journeys.

Social Value: Intelligent guiding systems can act as bridges between technology, culture, and ecology, encouraging respectful and sustainable interaction with both people and places.

Despite its promise, several limitations must be acknowledged. The reliance on Firebase introduces concerns related to cost, vendor lock-in, and limited querying capabilities in large-scale deployments. Although the employed Neural Networks proved effective, they raise important considerations regarding privacy, transparency, and fairness in the collection and interpretation of behavioral data. Furthermore, the computational demands of real-time map rendering and augmented-reality processing remain performance bottlenecks for lower-end mobile devices.

Future research directions include the exploration of privacy preserving artificial intelligence techniques such as federated learning and differential privacy to enhance user trust, as well as explainable AI methods that improve transparency and interpretability of preference predictions. Additional avenues involve energy efficient neural and neuromorphic models, expanded AR applications in domains such as education and healthcare, and hybrid positioning technologies based on multi-sensor fusion to improve localization accuracy in environments where GPS reliability is limited.

In conclusion, this dissertation provides a comprehensive and empirically validated framework for a next generation bio and cultural guiding system that is technically robust, contextually adaptive, and socially relevant. By merging bio-cultural awareness with artificial intelligence and augmented reality, the work contributes not only to the field of mobile and ubiquitous computing but also to a broader vision of sustainable, human centered technology. The results demonstrate that the future of guiding systems lies not merely in delivering information, but in shaping meaningful, personalized experiences that respect both human diversity and environmental responsibility.

8 REFERENCES

[1] Cetin, G. and Yarcan, S., 2017. The professional relationship between tour guides and tour operators. Scandinavian Journal of Hospitality and Tourism, 17(4), pp.345-357.

[2] Black, R. and Ham, S., 2005. Improving the quality of tour guiding: Towards a model for tour guide certification. *Journal of Ecotourism*, *4*(3), pp.178-195.

[3] Ap, J. and Wong, K.K., 2001. Case study on tour guiding: Professionalism, issues and problems. *Tourism management*, *22*(5), pp.551-563.

[4] Sofronov, B., 2018. The development of the travel and tourism industry in the world. *Annals of Spiru Haret University. Economic Series*, *18*(4), pp.123-137.

[5] Camilleri, M.A., 2017. The tourism industry: An overview. *Travel marketing, tourism economics and the airline product: An introduction to theory and practice*, pp.3-27.

[6] Leiper, N., 1979. The framework of tourism: Towards a definition of tourism, tourist, and the tourist industry. *Annals of tourism research*, *6*(4), pp.390-407.

[7] Kapa, M.G., de Crom, E.P. and Hermann, U.P., 2022. An exploration of tourist guides' competencies to create memorable tourist experiences. *GeoJournal of Tourism and Geosites*, *44*(4), pp.1350-1358.

[8] McDonnell, I., 2001. The role of the tour guide in transferring cultural understanding.

[9] Feldman, J. and Skinner, J., 2018. Tour guides as cultural mediators. *Ethnologia Europaea*, *48*(2), pp.5-13.

[10] Boyle, A. and Arnott, A., 2004. What tour guide stories can tell us about learning, education and training. *Creating Tourism Knowledge*, *75*.

[11] Chitima, S.S., 2025. Tour guides as facilitators of learning under field trips. *Journal of Teaching in Travel & Tourism*, *25*(1), pp.41-59.

[12] Tsaur, S.H. and Teng, H.Y., 2017. Exploring tour guiding styles: The perspective of tour leader roles. *Tourism Management*, *59*, pp.438-448.

[13] Poudel, S. and Nyaupane, G.P., 2013. The role of interpretative tour guiding in sustainable destination management: A comparison between guided and nonguided tourists. *Journal of Travel Research*, *52*(5), pp.659-672.

[14] Mujio, M., Rosandi, V.B., Waskitaningsih, N., Zahirah, K.R. and Chaniago, I.Y., 2025. Economic advancement of rural communities through tourism guide services as a paradigm for sustainable business transformation in the tropics. In *BIO Web of Conferences* (Vol. 186, p. 02001). EDP Sciences.

[15] Hu, W., 2007. Tour guides and sustainable development: The case of Hainan, China. University of Waterloo.

[16] Düz, B., 2025. Professional Ethics in Tour Guiding: Issues, Challenges, and Policies. In Modern Social Challenges Impacting Tourism: Ethics, Societal Disruption, and Intimacy (pp. 55-80). IGI Global Scientific Publishing.
[17] Nyahunzvi, D.K. and Njerekai, C., 2013. Tour guiding in Zimbabwe: Key issues and challenges. *Tourism Management Perspectives*, *6*, pp.3-7.
[18] Chowdhary, N. and Prakash, M., 2008. Challenges of tourist guiding-an assessment of situation in India.
[19] Prakash, M. and Chowdhary, N., 2010. Tour guides: Roles, challenges and desired competences a review of literature. *International Journal of Hospitality & Tourism Systems*, *3*(1).
[20] Hughes, K. and Ballantyne, R., 2001. Interpretation in ecotourism settings: investigating tour guides' perceptions of their role, responsibilities and training needs. *Journal of Tourism Studies*, *12*(2), pp.2-9.
[21] Schütz, K., 2017. The Logistical Challenges of Tour Management.
[22] Arslan, A. and Akkuş, G., 2024. A mixed methods research study on occupational safety and health of tour guides. *Current Issues in Tourism*, *27*(12), pp.1973-1988.
[23] Abd El Kafy, J., 2020. Challenges facing tour guide profession and their impacts on the Egyptian guides performance. *Journal of Association of Arab Universities for Tourism and Hospitality*, *19*(3), pp.113-130.
[24] Zejda, D. and Pásková, M., 2023. Destination Management Support System and Intelligent Destination Guide for Natural Destinations. In Public Recreation and Landscape Protection–With Environment Hand in Hand? Proceedings of the 14th Conference (pp. 130-134).
[25] Sengoz, A., Dogru, T., Mody, M. and Isik, C., 2025. Guiding the path to sustainable tourism development: Investigating the role of tour guides within a social exchange theory paradigm. *Tourism Management*, *110*, p.105162.
[26] Yamada, N., 2011. Why tour guiding is important for ecotourism: Enhancing guiding quality with the ecotourism promotion policy in Japan. *Asia Pacific Journal of Tourism Research*, *16*(2), pp.139-152.
[27] Örnek, N.A. and Karamustafa, K., 2025. Scenarios for the future of tour guiding: A Delphi-based analysis for 2040. *Futures*, *165*, p.103497.
[28] Abe, Y., Sato, S., Kato, K., Hyakumachi, T., Yanagibashi, Y., Ito, M. and Abumi, K., 2013. A novel 3D guidance system using augmented reality for percutaneous vertebroplasty. *Journal of Neurosurgery: Spine*, *19*(4), pp.492-501.
[29] Katz, B.F., Dramas, F., Parseihian, G., Gutierrez, O., Kammoun, S., Brilhault, A., Brunet, L., Gallay, M., Oriola, B., Auvray, M. and Truillet, P., 2012. NAVIG: Guidance system for the visually impaired using virtual augmented reality. *Technology and Disability*, *24*(2), pp.163-178.
[30] Vertucci, R., D'Onofrio, S., Ricciardi, S. and De Nino, M., 2023. History of augmented reality. In *Springer Handbook of Augmented Reality* (pp. 35-50). Cham: Springer International Publishing.
[31] Wei, X., Weng, D., Liu, Y. and Wang, Y., 2016, March. A tour guiding system of historical relics based on augmented reality. In *2016 IEEE Virtual Reality (VR)* (pp. 307-308). IEEE.
[32] Laviola, E., 2023. Optimization of augmented reality interfaces for technical documentation.
[33] Saha, N., 2024. *Augmented Reality (AR) and Virtual Reality (VR)-Based Data Visualization Frameworks for the Manufacturing Industry* (Master's thesis, University of South Carolina).

[34] Jasche, F., Hoffmann, S., Ludwig, T. and Wulf, V., 2021, May. Comparison of different types of augmented reality visualizations for instructions. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1-13).

[35] Gerstweiler, G., 2018, March. Guiding people in complex indoor environments using augmented reality. In *2018 IEEE conference on virtual reality and 3D user interfaces (VR)* (pp. 801-802). IEEE.

[36] Aukstakalnis, S., 2016. *Practical augmented reality: A guide to the technologies, applications, and human factors for AR and VR*. Addison-Wesley Professional.

[37] Zubizarreta, J., Aguinaga, I. and Amundarain, A., 2019. A framework for augmented reality guidance in industry. *The International Journal of Advanced Manufacturing Technology*, *102*(9), pp.4095-4108.

[38] Steele, P., Burleigh, C., Kroposki, M., Magabo, M. and Bailey, L., 2020. Ethical considerations in designing virtual and augmented reality products—Virtual and augmented reality design with students in mind: Designers' perceptions. *Journal of Educational Technology Systems*, *49*(2), pp.219-238.

[39] Blokša, J., 2017. Design guidelines for user interface for augmented reality. *Masaryk University*.

[40] Miyashita, T., Meier, P., Tachikawa, T., Orlic, S., Eble, T., Scholz, V., Gapel, A., Gerl, O., Arnaudov, S. and Lieberknecht, S., 2008, September. An augmented reality museum guide. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality* (pp. 103-106). IEEE.

[41] Chen, C.Y., Chang, B.R. and Huang, P.S., 2014. Multimedia augmented reality information system for museum guidance. *Personal and ubiquitous computing*, *18*(2), pp.315-322.

[42] Serres, B., Létienne, D., Roussey, O. and Venturini, G., 2023. Visit: A content management and exploration system for mobile augmented reality in the context of digital humanities and cultural heritage. *ACM Journal on Computing and Cultural Heritage*, *16*(2), pp.1-19.

[43] Siewert, J.L., Wolf, M., Böhm, B. and Thienhaus, S., 2020, February. Usability study for an augmented reality content management system. In *International Conference on Remote Engineering and Virtual Instrumentation* (pp. 274-287). Cham: Springer International Publishing.

[44] Diegmann, P., Schmidt-Kraepelin, M., Eynden, S. and Basten, D., 2015. Benefits of augmented reality in educational environments-a systematic literature review.

[45] Reinhart, G. and Patron, C., 2003. Integrating augmented reality in the assembly domain-fundamentals, benefits and applications. *CIRP Annals*, *52*(1), pp.5-8.

[46] Lee, C.I., 2022. Benefit analysis of gamified augmented reality navigation system. *Applied Sciences*, *12*(6), p.2969.

[47] Rose, E., Breen, D., Ahlers, K.H., Crampton, C., Tuceryan, M., Whitaker, R. and Greer, D., 1995. Annotating real-world objects using augmented reality. In *Computer graphics* (pp. 357-370). Academic Press.

[48] Nuernberger, B., Ofek, E., Benko, H. and Wilson, A.D., 2016, May. Snaptoreality: Aligning augmented reality to the real world. In *Proceedings of the 2016 CHI conference on human factors in computing systems* (pp. 1233-1244).

[49] Mekni, M. and Lemieux, A., 2014, April. Augmented reality: Applications, challenges and future trends. In *International Conference, 13th, Applied computer and applied computational science* (pp. 205-214).

[50] Bodhwani, K. and Sharma, A., 2023. The Future of Augmented Reality: Emerging Trends and Challenges. *landscape*, p.63.

[51] Haris, M., Jadoon, B., Yousaf, M. and Khan, F.H., 2018. Evolution of android operating system: a review. *Asia Pacific Journal of Contemporary Education and Communication Technology*, *4*(1), pp.178-188.

[52] Nimodia, C. and Deshmukh, H.R., 2012. Android operating system. *Software Engineering*, *3*(1), p.10.

[53] Chinetha, K.D., Joann, J.D. and Shalini, A., 2015. An evolution of android operating system and its version. *International Journal of Engineering and Applied Sciences*, *2*(2), p.257997.

[54] Farooq, U., 2018. Android operating system architecture. *Lahore, Pakistan: Virtual University of Pakistan*.

[55] Jaiswal, M., 2018. Android the mobile operating system and architecture. *Manishaben Jaiswal," ANDROID THE MOBILE OPERATING SYSTEM AND ARCHITECTURE", International Journal of Creative Research Thoughts (IJCRT), ISSN*, pp.2320-2882.

[56] Ye, R., 2017. *Android System Programming: Porting, customizing, and debugging Android HAL*. Packt Publishing Ltd.

[57] Yadav, R. and Bhadoria, R.S., 2015, April. Performance analysis for Android runtime environment. In *2015 Fifth International Conference on Communication Systems and Network Technologies* (pp. 1076-1079). IEEE.

[58] Lee, S. and Jeon, J.W., 2010, October. Evaluating performance of Android platform using native C for embedded systems. In *ICCAS 2010* (pp. 1160-1163). IEEE.

[59] Obradovic, N., Kelec, A. and Dujlovic, I., 2019, March. Performance analysis on Android SQLite database. In *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)* (pp. 1-4). IEEE.

[60] Shaheen, J.A., Asghar, M.A. and Hussain, A., 2017. Android OS with its architecture and android application with Dalvik virtual machine review. *International Journal of Multimedia and Ubiquitous Engineering*, *12*(7), pp.19-30.

[61] Verma, A., Arora, S. and Verma, P., 2017. Android OS, its security and features. *International Journal of Recent Research Aspects*, *4*(3), pp.241-251.

[62] Lazareska, L. and Jakimoski, K., 2017. Analysis of the advantages and disadvantages of Android and iOS systems and converting applications from Android to iOS platform and vice versa. *American Journal of Software Engineering and Applications*, *6*(5), pp.116-120.

[63] Abdullahi Idris, D.A.A.A. and Muhammad, U.S., Comparative Analysis of Modern Operating Systems.

[64] Higginbotham, J. and Jacobs, S., 2011. The future of the android operating system for augmentative and alternative communication. *Perspectives on augmentative and alternative communication*, *20*(2), pp.52-56.

[65] Radice, R.A., Roth, N.K., O'Hara, A.C. and Ciarfella, W.A., 1985. A programming process architecture. *IBM systems journal*, *24*(2), pp.79-90.

[66] Baumeister, H., Hacklinger, F., Hennicker, R., Knapp, A. and Wirsing, M., 2006. A component model for architectural programming. *Electronic Notes in Theoretical Computer Science*, *160*, pp.75-96.

[67] Pasieka, N., Sheketa, V., Romanyshyn, Y., Pasieka, M., Domska, U. and Struk, A., 2019, October. Models, methods and algorithms of web system architecture optimization. In *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)* (pp. 147-152). IEEE.

[68] Langer, A.M., 2008. Website design and architecture. In *Analysis and Design of Information Systems: Third Edition* (pp. 349-369). London: Springer London.

[69] Fielding, R.T. and Taylor, R.N., 2002. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, *2*(2), pp.115-150.

[70] Wang, C., Duan, W., Ma, J. and Wang, C., 2011, December. The research of Android System architecture and application programming. In *Proceedings of 2011 International Conference on Computer Science and Network Technology* (Vol. 2, pp. 785-790). IEEE.

[71] Gutierrez, A., Dreslinski, R.G., Wenisch, T.F., Mudge, T., Saidi, A., Emmons, C. and Paver, N., 2011, November. Full-system analysis and characterization of interactive smartphone applications. In *2011 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 81-90). IEEE.

[72] Li, C., Neugroschl, J., Zhu, C.W., Aloysi, A., Schimming, C.A., Cai, D., Grossman, H., Martin, J., Sewell, M., Loizos, M. and Zeng, X., 2021. Design considerations for mobile health applications targeting older adults. *Journal of Alzheimer's Disease*, *79*(1), pp.1-8.

[73] Dvorski, D.D., 2007. Installing, configuring, and developing with Xampp. *Skills Canada*, *492*.

[74] Laurie, B. and Laurie, P., 2003. *Apache: The definitive guide*. " O'Reilly Media, Inc.".

[75] Grippa, V.M. and Kuzmichev, S., 2021. *Learning MySQL*. " O'Reilly Media, Inc.".

[76] Lerdorf, R. and Tatroe, K., 2002. *Programming Php*. " O'Reilly Media, Inc.".

[77] Wibowo, M.C. and Putra, T.W.A., 2024. Utilizing phpMyAdmin for System Design in Enterprise Administration. *Journal of Technology Informatics and Engineering*, *3*(2), pp.217-234.

[78] Ehsan, A., Abuhaliqa, M.A.M., Catal, C. and Mishra, D., 2022. RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences*, *12*(9), p.4369.

[79] Moroney, L., 2017. The firebase realtime database. In *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform* (pp. 51-71). Berkeley, CA: Apress.

[80] Li, L., Bissyandé, T.F., Klein, J. and Le Traon, Y., 2016, March. An investigation into the use of common libraries in android apps. In *2016 IEEE 23Rd international conference on software analysis, evolution, and reengineering (SANER)* (Vol. 1, pp. 403-414). IEEE.

[81] Zhan, X., Liu, T., Fan, L., Li, L., Chen, S., Luo, X. and Liu, Y., 2021. Research on third-party libraries in android apps: A taxonomy and systematic literature review. *IEEE Transactions on Software Engineering*, *48*(10), pp.4181-4213.

[82] Rumpe, B., 2016. *Modeling with UML* (Vol. 98). Cham: Springer.

[83] Dobing, B. and Parsons, J., 2006. How UML is used. *Communications of the ACM*, *49*(5), pp.109-113.

[84] Eriksson, H.E. and Penker, M., 2000. Business modeling with UML. *New York*, *12*, p.444.

[85] Berardi, D., Calvanese, D. and De Giacomo, G., 2005. Reasoning on UML class diagrams. *Artificial intelligence*, *168*(1-2), pp.70-118.

[86] Li, X., Liu, Z. and Jifeng, H., 2004, April. A formal semantics of UML sequence diagram. In *2004 Australian Software Engineering Conference. Proceedings.* (pp. 168-177). IEEE.

[87] Ahn, J.W., Brusilovsky, P., Grady, J., He, D. and Syn, S.Y., 2007, May. Open user profiles for adaptive news systems: help or harm?. In *Proceedings of the 16th international conference on World Wide Web* (pp. 11-20).

[88] Adomavicius, G. and Tuzhilin, A., 2001. Expert-driven validation of rule-based user models in personalization applications. *Data Mining and Knowledge Discovery*, *5*, pp.33-58.

[89] Raghu, T.S., Kannan, P.K., Rao, H.R. and Whinston, A.B., 2001. Dynamic profiling of consumers for customized offerings over the Internet: A model and analysis. *Decision Support Systems*, *32*(2), pp.117-134.

[90] Ferguson, P. and Huston, G., 1998. What is a VPN?.

[91] Zhang, Z., Zhang, Y.Q., Chu, X. and Li, B., 2004. An overview of virtual private network (VPN): IP VPN and optical VPN. *Photonic network communications*, *7*, pp.213-225.

[92] Alshalan, A., Pisharody, S. and Huang, D., 2015. A survey of mobile VPN technologies. *IEEE Communications Surveys & Tutorials*, *18*(2), pp.1177-1196.

[93] Berger, T., 2006, April. Analysis of current VPN technologies. In *First International Conference on Availability, Reliability and Security (ARES'06)* (pp. 8-pp). IEEE.

[94] Fryzel, S.R., Henry, L.S. and Quigley, L.E., 2021. Legal Impact of FFIEC Update to Authentication Guidance for Internet-Based Financial Services. *Banking LJ*, *138*, p.576.

[95] Dolby, A., 2007. Multi-factor authentication for Internet banking: The FFIEC guidance aftermath. *Journal of Corporate Treasury Management*, *1*(2).

[96] Markowitz, J., 2007. E-banking security: voice biometrics is not enough as a single layer of defense. *Speech Technology Magazine*, *12*(3), pp.44-45.

[97] Fryzel, S., 2011. The FFIEC's Supplement to Authentication in an Internet Banking Environment: The New Minimum Legal Standard. *Banking LJ*, *128*, p.827.

[98] Juels, A., Jakobsson, M. and Jagatic, T.N., 2006, May. Cache cookies for browser authentication. In *2006 IEEE Symposium on Security and Privacy (S&P'06)* (pp. 5-pp). IEEE.

[99] Mobasher, B., Dai, H., Luo, T. and Nakagawa, M., 2002. Discovery and evaluation of aggregate usage profiles for web personalization. *Data mining and knowledge discovery*, *6*, pp.61-82.

[100] Mushtaq, N., Werner, P., Tolle, K. and Zicari, R., 2004, July. Building and evaluating nonobvious user profiles for visitors of Web sites. In *Proceedings. IEEE International Conference on e-Commerce Technology, 2004. CEC 2004.* (pp. 9-15). IEEE.

[101] Rafter, R. and Smyth, B., 2001. Passive profiling from server logs in an online recruitment environment.

[102] Cortes, C. and Pregibon, D., 2001. Signature-based methods for data streams. *Data Mining and Knowledge Discovery*, *5*(3), pp.167-182.

[103] Albalawi, M., Aloufi, R., Alamrani, N., Albalawi, N., Aljaedi, A. and Alharbi, A.R., 2022. Website defacement detection and monitoring methods: A review. *Electronics*, *11*(21), p.3573.

[104] Yang, Y.C., 2010. Web user behavioral profiling for user identification. *Decision Support Systems*, *49*(3), pp.261-271.

[105] Vardakis, G., Tsamis, G., Koutsaki, E., Haridimos, K. and Papadakis, N., 2022. Smart home: Deep learning as a method for machine learning in recognition of face, silhouette and human activity in the service of a safe home. Electronics, 11(10), p.1622.

[106] Van Laarhoven, T., 2017. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*.

[107] Jouppi, N., Young, C., Patil, N. and Patterson, D., 2018. Motivation for and evaluation of the first tensor processing unit. *ieee Micro*, *38*(3), pp.10-19.

[108] Amari, S.I., 1993. Backpropagation and stochastic gradient descent method. *Neurocomputing*, *5*(4-5), pp.185-196.

[109] Jais, I.K.M., Ismail, A.R. and Nisa, S.Q., 2019. Adam optimization algorithm for wide and deep neural network. *Knowl. Eng. Data Sci.*, *2*(1), pp.41-46.

[110] Graves, A. and Graves, A., 2012. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pp.37-45.

[111] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[112] Rothman, D., 2021. *Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more*. Packt Publishing Ltd.

[113] Le, N.Q.K., Ho, Q.T., Nguyen, T.T.D. and Ou, Y.Y., 2021. A transformer architecture based on BERT and 2D convolutional neural network to identify DNA enhancers from sequence information. *Briefings in bioinformatics*, *22*(5), p.bbab005.

[114] Yates, A., Nogueira, R. and Lin, J., 2021, March. Pretrained transformers for text ranking: BERT and beyond. In *Proceedings of the 14th ACM International Conference on web search and data mining* (pp. 1154-1156).

[115] Topal, M.O., Bas, A. and van Heerden, I., 2021. Exploring transformers in natural language generation: Gpt, bert, and xlnet. *arXiv preprint arXiv:2102.08036*.

[116] Ghojogh, B. and Ghodsi, A., 2020. Attention mechanism, transformers, BERT, and GPT: tutorial and survey.

[117] Sufi, F., 2025. Advanced computational methods for news classification: A study in neural networks and CNN integrated with GPT. *Journal of Economy and Technology*, *3*, pp.264-281.

[118] Adewumi, T., Sabry, S.S., Abid, N., Liwicki, F. and Liwicki, M., 2023. T5 for hate speech, augmented data, and ensemble. *Sci*, *5*(4), p.37.

[119] Nawrot, P., 2023, December. NanoT5: Fast & simple pre-training and fine-tuning of t5 models with limited resources. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)* (pp. 95-101).

[120] Bird, J.J., Ekárt, A. and Faria, D.R., 2023. Chatbot Interaction with Artificial Intelligence: human data augmentation with T5 and language transformer ensemble for text classification. *Journal of Ambient Intelligence and Humanized Computing*, *14*(4), pp.3129-3144.

[121] Sordo, M., 2002. Introduction to neural networks in healthcare. *Open clinical: Knowledge management for medical care*.

[122] Begg, R., Kamruzzaman, J. and Sarker, R. eds., 2006. *Neural networks in healthcare: Potential and challenges: Potential and challenges*. Igi Global.

[123] Zion, I., Ozuomba, S. and Asuquo, P., 2020, March. An overview of neural network architectures for healthcare. In *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)* (pp. 1-8). IEEE.

[124] Shahid, N., Rappon, T. and Berta, W., 2019. Applications of artificial neural networks in health care organizational decision-making: A scoping review. *PloS one*, *14*(2), p.e0212356.

[125] Amuda, K.A. and Adeyemo, A.B., 2019. Customers churn prediction in financial institution using artificial neural network. *arXiv preprint arXiv:1912.11346*.

[126] Duan, J., 2019. Financial system modeling using deep neural networks (DNNs) for effective risk assessment and prediction. *Journal of the Franklin Institute*, *356*(8), pp.4716-4731.

[127] Vikram, A., Chennuru, S., Rao, H.R. and Upadhyaya, S., 2004, January. A solution architecture for financial institutions to handle illegal activities: a neural networks approach. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the* (pp. 181-190). IEEE.

[128] Tsamis, G., Evangelos, G., Papakostas, A., Vassiliou, G., Grafanakis, M., Garefalakis, A., Vassalos, M., Mylona, A. and Papadakis, N., 2025. Cost-Effective Design, Content Management System Implementation and Artificial Intelligence Support of Greek Government AADE, myDATA Web Service for Generic Government Infrastructure, a Complete Analysis. *Algorithms*, *18*(6), p.339.

[129] Tsamis, G., Vassiliou, G., Chatzinikolaou, S., Kondylakis, H., & Papadakis, N. (2025). A Multi-Constrained Knapsack Approach for Educational Resource Allocation: Genetic Algorithm with Category- Specific Optimization. *Electronics*, *14*(19), 3898. https://doi.org/10.3390/electronics14193898

[130] Zhang, H.C. and Huang, S.H., 1995. Applications of neural networks in manufacturing: a state-of-the-art survey. *The International Journal of Production Research*, *33*(3), pp.705-728.

[131] Huang, S.H. and Zhang, H.C., 1994. Artificial neural networks in manufacturing: concepts, applications, and perspectives. *IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part A*, *17*(2), pp.212-228.

[132] Mumali, F., 2022. Artificial neural network-based decision support systems in manufacturing processes: A systematic literature review. *Computers & Industrial Engineering*, *165*, p.107964.

[133] Do, T.D., Duong, M.T., Dang, Q.V. and Le, M.H., 2018, November. Real-time self-driving car navigation using deep neural network. In *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)* (pp. 7-12). IEEE.

[134] Gudla, R., Telidevulapalli, V.S., Kota, J.S. and Mandha, G., 2022. Review on self-driving cars using neural network architectures. *World Journal of Advanced Research and Reviews*, *16*(2), pp.736-746.

[135] Agnihotri, A., Saraf, P. and Bapnad, K.R., 2019, December. A convolutional neural network approach towards self-driving cars. In *2019 IEEE 16th India Council International Conference (INDICON)* (pp. 1-4). IEEE.

[136] Bekey, G.A. and Goldberg, K.Y. eds., 2012. *Neural networks in robotics* (Vol. 202). Springer Science & Business Media.

[137] Horne, B., Jamshidi, M. and Vadiee, N., 1990. Neural networks in robotics: A survey. *Journal of Intelligent and Robotic Systems*, *3*(1), pp.51-66.

[138] Yang, S.X. and Meng, M., 2000. An efficient neural network approach to dynamic robot motion planning. *Neural networks*, *13*(2), pp.143-148.

[139] Tzafestas, S.G., 1995, July. Neural networks in robotics: state of the art. In *1995 Proceedings of the IEEE International Symposium on Industrial Electronics* (Vol. 1, pp. 12-20). IEEE.

[140] Jiang, Y., Yang, C., Na, J., Li, G., Li, Y. and Zhong, J., 2017. A brief review of neural networks based learning and control and their applications for robots. *Complexity*, *2017*(1), p.1895897.

[141] Goldberg, Y., 2016. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, *57*, pp.345-420.

[142] Liu, Y. and Zhang, M., 2018. Neural network methods for natural language processing.

[143] Collobert, R. and Weston, J., 2008, July. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning* (pp. 160-167).

[144] Goldberg, Y., 2017. *Neural network methods in natural language processing*. Morgan & Claypool Publishers.

[145] Tarwani, K.M. and Edem, S., 2017. Survey on recurrent neural network in natural language processing. *Int. J. Eng. Trends Technol*, *48*(6), pp.301-304.

[146] Montejo-Ráez, A. and Jiménez-Zafra, S.M., 2022. Current approaches and applications in natural language processing. *Applied Sciences*, *12*(10), p.4859.

[147] Mah, P.M., Skalna, I. and Muzam, J., 2022. Natural language processing and artificial intelligence for enterprise management in the era of industry 4.0. *Applied Sciences*, *12*(18), p.9207.

[148] de Oliveira, N.R., Pisa, P.S., Lopez, M.A., de Medeiros, D.S.V. and Mattos, D.M., 2021. Identifying fake news on social networks based on natural language processing: trends and challenges. *Information*, *12*(1), p.38.

[149] Roumeliotis, K.I., Tselikas, N.D. and Nasiopoulos, D.K., 2025. Fake News Detection and Classification: A Comparative Study of Convolutional Neural Networks, Large Language Models, and Natural Language Processing Models. *Future Internet*, *17*(1).

[150] Khalif, K.M.N.K., Chaw Seng, W., Gegov, A., Bakar, A.S.A. and Shahrul, N.A., 2024. Integrated generative adversarial networks and deep convolutional neural networks for image data classification: A case study for covid-19. *Information*, *15*(1), p.58.

[151] Laino, M.E., Cancian, P., Politi, L.S., Della Porta, M.G., Saba, L. and Savevski, V., 2022. Generative adversarial networks in brain imaging: A narrative review. *Journal of imaging*, *8*(4), p.83.

[152] La Salvia, M., Torti, E., Leon, R., Fabelo, H., Ortega, S., Martinez-Vega, B., Callico, G.M. and Leporati, F., 2022. Deep convolutional generative adversarial networks to enhance artificial intelligence in healthcare: a skin cancer application. *Sensors*, *22*(16), p.6145.

[153] Park, S.W., Ko, J.S., Huh, J.H., Kim, J.C. and Dimauro, G., 2021. Review on generative adversarial networks: Focusing on computer vision and its applications. *Electronics (2079-9292)*, *10*(10).

[154] Kosarkar, U., Sarkarkar, G. and Gedam, S., 2023. Revealing and classification of deepfakes video's images using a customize convolution neural network model. *Procedia Computer Science*, *218*, pp.2636-2652.

[155] Kolagati, S., Priyadharshini, T. and Rajam, V.M.A., 2022. Exposing deepfakes using a deep multilayer perceptron–convolutional neural network model. *International Journal of Information Management Data Insights*, *2*(1), p.100054.

[156] Güera, D. and Delp, E.J., 2018, November. Deepfake video detection using recurrent neural networks. In *2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS)* (pp. 1-6). IEEE.

[157] Han, S., Pool, J., Tran, J. and Dally, W., 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, *28*.

[158] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W. and Keutzer, K., 2022. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision* (pp. 291-326). Chapman and Hall/CRC.

[159] Hitzler, P., Eberhart, A., Ebrahimi, M., Sarker, M.K. and Zhou, L., 2022. Neuro-symbolic approaches in artificial intelligence. *National Science Review*, *9*(6), p.nwac035.

[160] Bhuyan, B.P., Ramdane-Cherif, A., Tomar, R. and Singh, T.P., 2024. Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, *36*(21), pp.12809-12844.

[161] Chen, J., Mi, R., Wang, H., Wu, H., Mo, J., Guo, J., Lai, Z., Zhang, L. and Leung, V.C., 2024. A review of few-shot and zero-shot learning for node classification in social networks. *IEEE Transactions on Computational Social Systems*.

[162] Kadam, S. and Vaidya, V., 2020. Review and analysis of zero, one and few shot learning approaches. In *Intelligent Systems Design and Applications: 18th International Conference on Intelligent Systems Design and Applications (ISDA 2018) held in Vellore, India, December 6-8, 2018, Volume 1* (pp. 100-112). Springer International Publishing.

[163] Chen, J., Geng, Y., Chen, Z., Pan, J.Z., He, Y., Zhang, W., Horrocks, I. and Chen, H., 2023. Zero-shot and few-shot learning with knowledge graphs: A comprehensive survey. *Proceedings of the IEEE*, *111*(6), pp.653-685.

[164] Orozco, B.P. and Roberts, S.J., 2020. Zero-shot and few-shot time series forecasting with ordinal regression recurrent neural networks. *arXiv preprint arXiv:2003.12162*.

[165] Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D. and Zhu, J., 2019. Explainable AI: A brief survey on history, research areas, approaches and challenges. In *Natural language processing and Chinese computing: 8th cCF international conference, NLPCC 2019, dunhuang, China, October 9–14, 2019, proceedings, part II 8* (pp. 563-574). Springer International Publishing.

[166] Holzinger, A., Saranti, A., Molnar, C., Biecek, P. and Samek, W., 2020, July. Explainable AI methods-a brief overview. In *International workshop on extending*

explainable AI beyond deep models and classifiers (pp. 13-38). Cham: Springer International Publishing.

[167] de Sousa Ribeiro, M. and Leite, J., 2021, May. Aligning artificial neural networks and ontologies towards explainable AI. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 35, No. 6, pp. 4932-4940).

[168] Das, A. and Rad, P., 2020. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371*.

[169] Javanshir, A., Nguyen, T.T., Mahmud, M.P. and Kouzani, A.Z., 2022. Advancements in algorithms and neuromorphic hardware for spiking neural networks. *Neural Computation*, *34*(6), pp.1289-1328.

[170] Ma, D., Shen, J., Gu, Z., Zhang, M., Zhu, X., Xu, X., Xu, Q., Shen, Y. and Pan, G., 2017. Darwin: A neuromorphic hardware co-processor based on spiking neural networks. *Journal of systems architecture*, *77*, pp.43-51.

[171] Ji, Y., Zhang, Y., Li, S., Chi, P., Jiang, C., Qu, P., Xie, Y. and Chen, W., 2016, October. NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (pp. 1-13). IEEE.

[172] Ji, Y., Zhang, Y., Chen, W. and Xie, Y., 2018, March. Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler. In *Proceedings of the twenty-third international conference on architectural support for programming languages and operating systems* (pp. 448-460).

[173] Kramer, O. and Kramer, O., 2016. Scikit-learn. *Machine learning for evolution strategies*, pp.45-53.

[174] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, pp.2825-2830.

[175] Bisong, E., 2019. Introduction to Scikit-learn. In *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners* (pp. 215-229). Berkeley, CA: Apress.

[176] Hao, J. and Ho, T.K., 2019. Machine learning made easy: a review of scikit-learn package in python programming language. *Journal of Educational and Behavioral Statistics*, *44*(3), pp.348-361.

[177] Melnichuk, M., Kornienko, Y. and Boytsova, O., 2018. Web-service. restful architecture. *Автоматизация технологических и бизнес-процессов*, (10, Iss. 1), pp.17-22.

[178] Foping, F., Walsh, J. and Roche, D., 2013, December. Design and Implementation of a Private RESTful API to Leverage the Power of an eCommerce Platform. In *Proceedings of International Conference on Information Integration and Web-based Applications & Services* (pp. 681-685).

[179] Lachgar, M., Benouda, H. and Elfirdoussi, S., 2018, November. Android rest APIS: volley vs retrofit. In *2018 international symposium on advanced electrical and communication technologies (ISAECT)* (pp. 1-6). IEEE.

[180] Subramanian, H. and Raj, P., 2019. *Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs*. Packt Publishing Ltd.

[181] Abdellatif, M., Tighilt, R., Belkhir, A., Moha, N., Guéhéneuc, Y.G. and Beaudry, É., 2020. A multi-dimensional study on the state of the practice of REST APIs usage in Android apps. *Automated Software Engineering*, *27*(3), pp.187-228.

[182] Aditya, S.K. and Karn, V.K., 2014. *Android sQLite essentials* (pp. 88-90). Packt Publishing.

[183] Vogel, L., 2010. Android sqlite database and contentprovider-tutorial. *Java, Eclipse, Android and Web programming tutorials*, *8*, p.5.

[184] Ferreira, C., Lopes, M., Correia, L., Wanzeller, C., Sá, F., Martins, P. and Abbasi, M., 2023. Database Performance on Android Devices, A Comparative Analysis. In Marketing and Smart Technologies: Proceedings of ICMarkTech 2022, Volume 2 (pp. 261-273). Singapore: Springer Nature Singapore.

[185] Dixon, T.H., 1991. An introduction to the Global Positioning System and some geological applications. *Reviews of geophysics*, *29*(2), pp.249-276.

[186] McNeff, J.G., 2002. The global positioning system. *IEEE Transactions on Microwave theory and techniques*, *50*(3), pp.645-652.

[187] Mooney, P. and Minghini, M., 2017. A review of OpenStreetMap data. *Mapping and the citizen sensor*, pp.37-59.

[188] Haklay, M. and Weber, P., 2008. Openstreetmap: User-generated street maps. *IEEE Pervasive computing*, *7*(4), pp.12-18.

[189] Soon, T.J., 2008. QR code. *synthesis journal*, *2008*(3), pp.59-78.

[190] Tiwari, S., 2016, December. An introduction to QR code technology. In *2016 international conference on information technology (ICIT)* (pp. 39-44). IEEE.

[191] Bhagavatula, R., Ur, B., Iacovino, K., Kywe, S.M., Cranor, L.F. and Savvides, M., 2015. Biometric authentication on iphone and android: Usability, perceptions, and influences on adoption.

[192] Zhang, X., Cheng, D., Jia, P., Dai, Y. and Xu, X., 2020. An efficient android-based multimodal biometric authentication system with face and voice. *Ieee Access*, *8*, pp.102757-102772.

[193] Ahlawat, S. and Dahiya, R., 2013. A NOVEL APPROACH OF TEXT TO SPEECH CONVERSION UNDER ANDROID ENVIRONMENT. *International Journal of Computer Science & Management Studies*, *13*(5).

[194] Sonawane, A.R., Wankhede, A., Rasane, K., Baraskaev, V. and Borde, G., 2016. Android Application for Visually Impaired People using Text-To-Speech. *International Journal of Advanced Research in Computer and Communication Engineering*, *5*, p.12.

[195] Song, W., Lee, H., Lee, S.H., Choi, M.H. and Hong, M., 2018. Implementation of android application for indoor positioning system with estimote BLE beacons. *Journal of Internet Technology*, *19*(3), pp.871-878.

[196] Kriz, P., Maly, F. and Kozel, T., 2016. Improving indoor localization using bluetooth low energy beacons. *Mobile information systems*, *2016*(1), p.2083094.

[197] Zuo, C., Wen, H., Lin, Z. and Zhang, Y., 2019, November. Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1469-1483).

[198] Chougale, P., Yadav, V., Gaikwad, A. and Vidyapeeth, B., 2021. Firebase-overview and usage. *International Research Journal of Modernization in Engineering Technology and Science*, *3*(12), pp.1178-1183.

[199] Khawas, C. and Shah, P., 2018. Application of firebase in android app development-a study. *International Journal of Computer Applications*, *179*(46), pp.49-53.

[200] Chatterjee, N., Chakraborty, S., Decosta, A. and Nath, A., 2018. Real-time communication application based on android using Google firebase. Int. J. Adv. Res. Comput. Sci. Manag. Stud, 6(4).

[201] I. E. Sutherland, “A head-mounted three dimensional display,” in *Proc. December 9–11, 1968, Fall Joint Computer Conference, Part I*, 1968, pp. 757–764, doi: 10.1145/1476589.1476686.

[202] P. Milgram and F. Kishino, “A taxonomy of mixed reality visual displays,” *IEICE Transactions on Information Systems*, vol. 77, no. 12, pp. 1321–1329, 1994.

[203] R. T. Azuma, “A survey of augmented reality,” *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 355–385, 1997, doi: 10.1162/pres.1997.6.4.355.

[204] S. J. Henderson and S. K. Feiner, “Exploring the benefits of augmented reality documentation for maintenance and repair,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 10, pp. 1355–1368, 2011, doi: 10.1109/TVCG.2010.245.

[205] C. D. Kounavis, A. E. Kasimati, and E. D. Zamani, “Enhancing the tourism experience through mobile augmented reality: Challenges and prospects,” *International Journal of Engineering Business Management*, vol. 4, p. 10, 2012, doi: 10.5772/51644.

[206] M. C. tom Dieck and T. Jung, “Value of augmented reality at cultural heritage sites: A stakeholder approach,” *Journal of Destination Marketing & Management*, vol. 6, no. 2, pp. 110–117, 2017, doi: 10.1016/j.jdmm.2017.03.002.

[207] N. Chung, H. Han, and Y. Joun, “Tourists’ intention to visit a destination: The role of augmented reality (AR) application for a heritage site,” *Computers in Human Behavior*, vol. 87, pp. 273–283, 2018, doi: 10.1016/j.chb.2018.06.017.

[208] S. Sylaiou, K. Mania, A. Karoulis, and M. White, “Exploring the relationship between presence and enjoyment in a virtual museum,” *International Journal of Human–Computer Studies*, vol. 68, no. 5, pp. 243–253, 2010, doi: 10.1016/j.ijhcs.2009.11.002.

[209] M. Billinghurst, A. Clark, and G. Lee, “A survey of augmented reality,” *Foundations and Trends in Human–Computer Interaction*, vol. 8, no. 2–3, pp. 73–272, 2015, doi: 10.1561/1100000049.

[210] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*, 2nd ed. Cham: Springer, 2015.

[211] C. C. Aggarwal, *Recommender Systems: The Textbook*. Cham: Springer, 2016.

[212] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, “Mobile recommender systems in tourism,” *Journal of Network and Computer Applications*, vol. 39, pp. 319–333, 2014, doi: 10.1016/j.jnca.2013.04.006.

[213] K. Kabassi, “Personalizing recommendations for tourists,” *Tourism Management*, vol. 62, pp. 437–446, 2017, doi: 10.1016/j.tourman.2017.06.009.

[214] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[215] Y. Zhang, X. Chen, and S. Wang, “Personalized recommendation in tourism based on deep learning,” *Journal of Intelligent Information Systems*, vol. 52, no. 1, pp. 27–45, 2019, doi: 10.1007/s10844-018-0502-y.

[216] P. Resnick and H. R. Varian, “Recommender systems,” *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997, doi: 10.1145/245108.245121.

[217] D. Shin, “The effects of explainability and causability on perception, trust, and acceptance: Implications for explainable AI,” *International Journal of Human–Computer Studies*, vol. 146, p. 102551, 2020, doi: 10.1016/j.ijhcs.2020.102551.

[218] A. Adadi and M. Berrada, “Peeking inside the black-box: A survey on explainable artificial intelligence (XAI),” *IEEE Access*, vol. 6, pp. 52138–52160, 2018, doi: 10.1109/ACCESS.2018.2870052.

[219] P. Kairouz et al., “Advances and open problems in federated learning,” *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021, doi: 10.1561/2200000083.

[220] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, *Developing Recommender Systems*. Cham: Springer, 2015.

[221] T. Jung, M. C. tom Dieck, N. Moorhouse, and D. Dieck, “Tourists’ experience using mobile augmented reality: A case study of the Roman Baths,” *Tourism Management*, vol. 51, pp. 430–440, 2015, doi: 10.1016/j.tourman.2015.05.003.

[222] M. T. Linaza, D. Marimon, P. Carrasco, R. Álvarez, and J. Montesa, “Evaluation of mobile augmented reality applications for tourism destinations,” in *Information and Communication Technologies in Tourism 2014*, Cham: Springer, 2014, pp. 431–445, doi: 10.1007/978-3-319-03973-2_32.

[223] M. Mortara, C. E. Catalano, F. Bellotti, G. Fiucci, M. Houry-Panchetti, and P. Petridis, “Learning cultural heritage by serious games,” *Journal of Cultural Heritage*, vol. 15, no. 3, pp. 318–325, 2014, doi: 10.1016/j.culher.2013.04.004.

[224] D. Wang, S. Park, and D. R. Fesenmaier, “The role of smartphones in mediating the touristic experience,” *Journal of Travel Research*, vol. 51, no. 4, pp. 371–387, 2012, doi: 10.1177/0047287511410320.

[225] I. Tussyadiah, “The influence of innovative technologies on travel decisions,” in *Information and Communication Technologies in Tourism 2015*, Cham: Springer, 2015, pp. 335–347, doi: 10.1007/978-3-319-14343-9_24.

[226] A. Damala, P. Cubaud, A. Bationo, P. Houlier, and I. Marchal, “Bridging the gap between the digital and the physical: Design and evaluation of a mobile augmented reality guide for the museum visit,” *Personal and Ubiquitous Computing*, vol. 23, no. 2, pp. 167–181, 2019, doi: 10.1007/s00779-018-1186-5.

[227] F. Liarokapis, P. Petridis, D. Andrews, and S. de Freitas, “Multimodal serious games technologies for cultural heritage,” *Mediterranean Archaeology and Archaeometry*, vol. 17, no. 3, pp. 73–87, 2017.

[228] Vassiliou, G., Tsamis, G., Chatzinikolaou, S., Nipurakis, T. and Papadakis, N., 2025. Enhancing Discoverability: A Metadata Framework for Empirical Research in Theses. *Algorithms*, 18(8), p.490.

[229] Tsamis, G., Vassiliou, G., Garefalakis, A., Rousaki, M., Papakostas, A., Tzagkarakis, H., White, D.C., Papadakis, N., 2026. Enhanced Bio and Cultural Tourist Navigation and Guiding Application System for Android OS Smartphones, Supporting Augmented Tour Operating Experience. *Multimedia*.

9 APPENDICES

APPENDIX 1 CLASS DIAGRAM 245

APPENDIX 2 AI ALGORITHMS AND DATA 248

APPENDIX 1 CLASS DIAGRAM

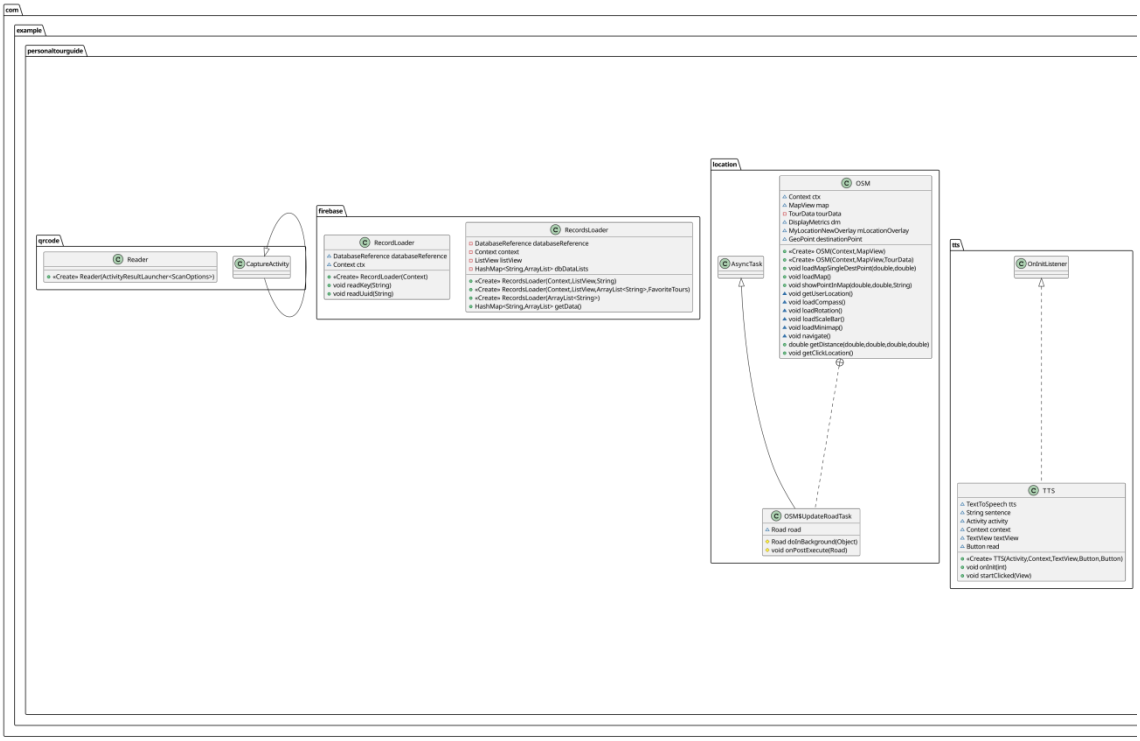


Figure 9.1: Class Diagram Section (a)

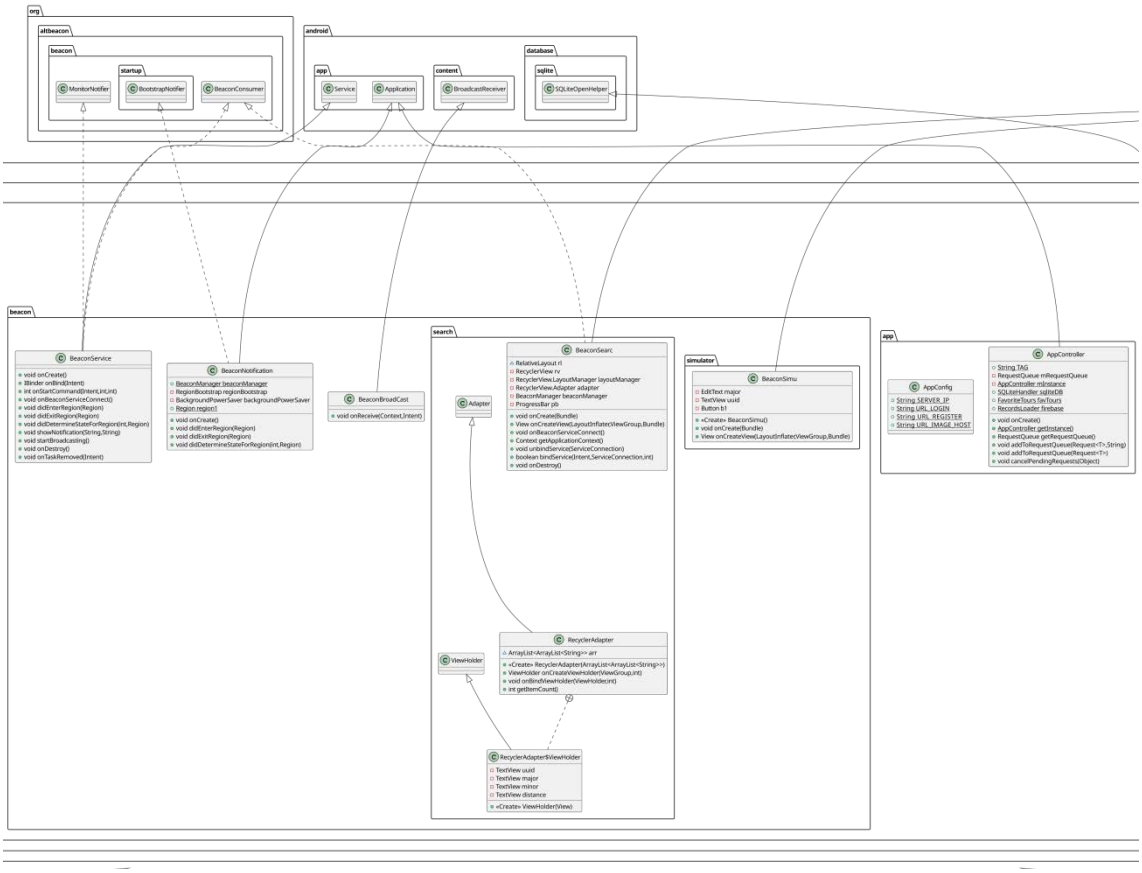


Figure 9.2: Class Diagram Section (b)

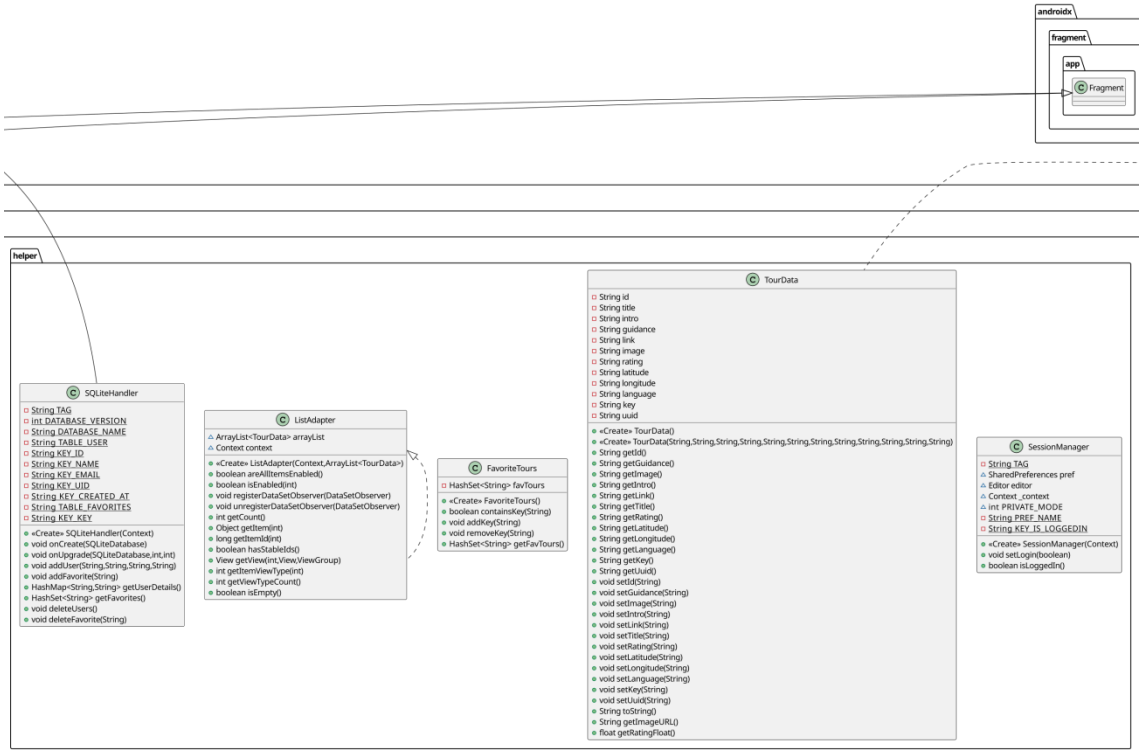


Figure 9.3: Class Diagram Section (c)

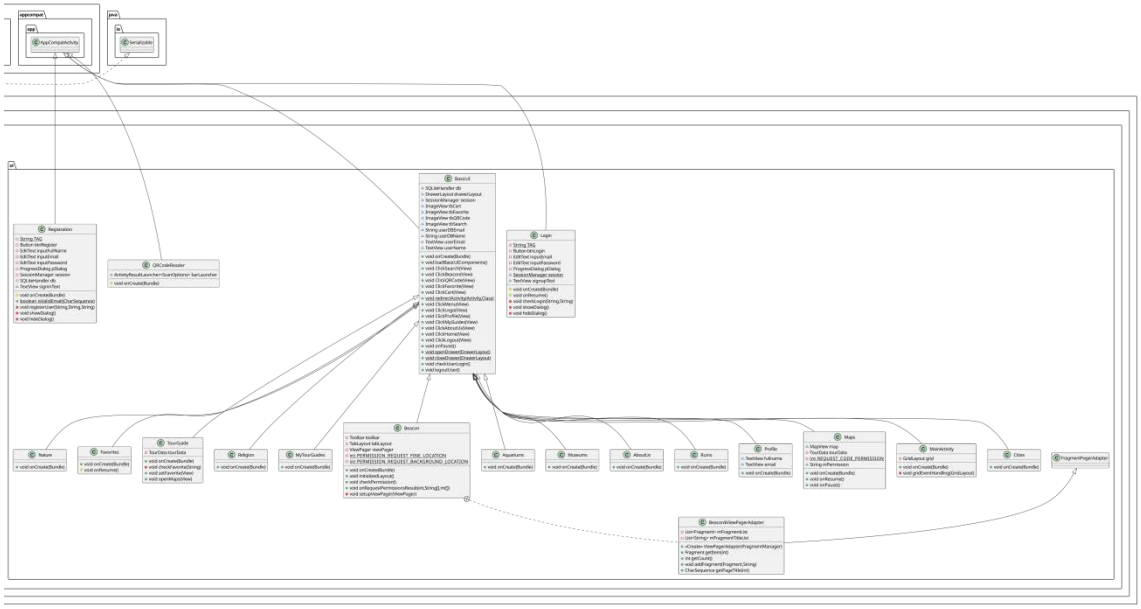


Figure 9.4: Class Diagram Section (d)

APPENDIX 2 AI ALGORITHMS AND DATA

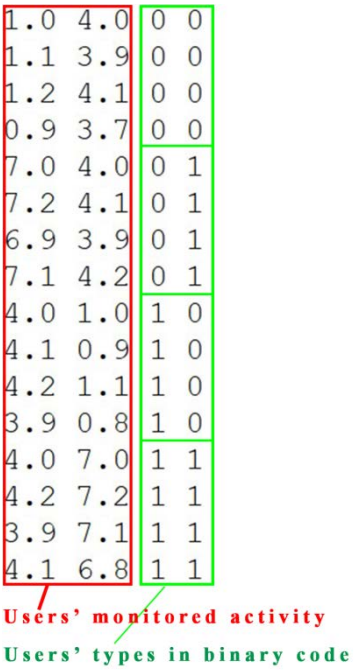


Figure 9.5: Appendix A.1, Input Test Data, Four Columns

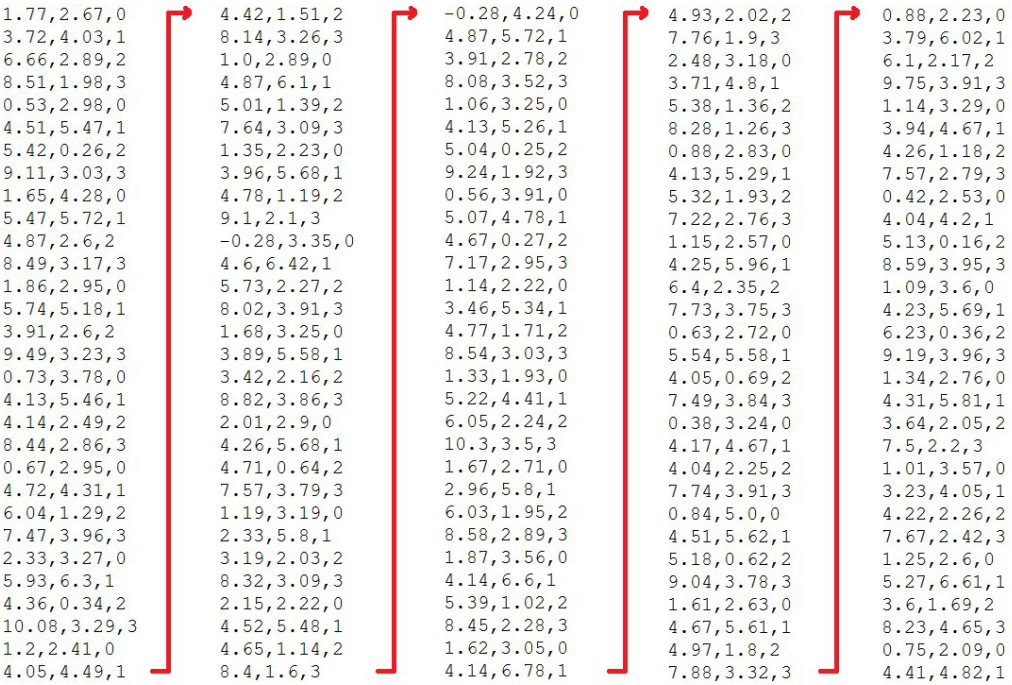


Figure 9.6: Appendix B.1, Input Test Data, Three Columns

```
{
    "David Smith":
    {
        "Vertigo": 4,
        "Scarface": 4.5,
        "Raging Bull": 3.0,
        "Goodfellas": 4.5,
        "The Apartment": 1.0
    },
    "Brenda Peterson":
    {
        "Vertigo": 3.0,
        "Scarface": 1.5,
        "Raging Bull": 1.0,
        "Goodfellas": 2.0,
        "The Apartment": 5.0,
        "Roman Holiday": 4.5
    },
    "Bill Duffy":
    {
        "Vertigo": 4.5,
        "Scarface": 5.0,
        "Goodfellas": 4.5,
        "The Apartment": 1.0
    },
    "Samuel Miller":
    {
        "Scarface": 3.5,
        "Raging Bull": 5.0,
        "The Apartment": 1.0,
        "Goodfellas": 5.0,
        "Roman Holiday": 1.0
    },
    "Chris Duncan":
    {
       "The Apartment": 1.5,
       "Raging Bull": 4.5
    }
}
```

Figure 9.7: Appendix C.1, Training Users' Data

```
import argparse
import json
import numpy as np

def build_arg_parser():
    parser = argparse.ArgumentParser(description='Compute similarity score')
    parser.add_argument('--user1', dest='user1', required=True,
            help='First user')
    parser.add_argument('--user2', dest='user2', required=True,
            help='Second user')
    parser.add_argument("--score-type", dest="score_type", required=True,
            choices=['Euclidean', 'Pearson'], help='Similarity metric to be used')
    return parser
```

Figure 9.8: Appendix C.2, Arguments Parser

```python
# Compute the Euclidean distance score between user1 and user2
def euclidean_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')

    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')

    # Movies rated by both user1 and user2
    common_movies = {}

    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    # If there are no common movies between the users,
    # then the score is 0
    if len(common_movies) == 0:
        return 0

    # Compute the squared differences between the ratings
    squared_diff = []

    for item in dataset[user1]:
        if item in dataset[user2]:
            squared_diff.append(np.square(dataset[user1][item] - dataset[user2][item]))

    # Return the Euclidean score (inverse Euclidean distance and bounded [0,1])
    return 1 / (1 + np.sqrt(np.sum(squared_diff)))
```

Figure 9.9: Appendix C.3, Euclidean Score Algorithm

```python
# Compute the Pearson correlation score between user1 and user2
def pearson_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')

    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')

    # Movies rated by both user1 and user2
    common_movies = {}

    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1

    num_ratings = len(common_movies)

    # If there are no common movies between user1 and user2, then the score is 0
    if num_ratings == 0:
        return 0

    # Calculate the sum of ratings of all the common movies
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])

    # Calculate the sum of squares of ratings of all the common movies
    user1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in common_movies])
    user2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in common_movies])

    # Calculate the sum of products of the ratings of the common movies
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for item in common_movies])

    # Calculate the Pearson correlation score
    # Calculate the various parameters required to compute the Pearson score
    Sxy = sum_of_products - (user1_sum * user2_sum / num_ratings)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_ratings
    Syy = user2_squared_sum - np.square(user2_sum) / num_ratings

    # If there is no deviation, then the score is 0
    if Sxx * Syy == 0:
        return 0

    # Return the Pearson score
    return Sxy / np.sqrt(Sxx * Syy)
```

Figure 9.10: Appendix C.4, Pearson Score Algorithm

```
# if __name__ == "__main__" protects other people to execute test and documentation code
# automatically if they import your module. If you see this kind of code in another module,
# you can import it and the code will not be executed
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user1 = args.user1
    user2 = args.user2
    score_type = args.score_type

    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    if score_type == 'Euclidean':
        print("\nEuclidean score:")
        print(euclidean_score(data, user1, user2))
    else:
        print("\nPearson score:")
        print(pearson_score(data, user1, user2))
```

Figure 9.11: Appendix C.5, the Main Function

```
import argparse
import json
import numpy as np

from compute_scores import pearson_score

def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find users who are similar to the input user')
    parser.add_argument('--user', dest='user', required=True,
            help='Input user')
    return parser
```

Figure 9.12: Appendix C.6, the Argument Parser

```
# Finds users in the dataset that are similar to the input user
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')

    # Compute Pearson score between input user
    # and all the users in the dataset
    scores = np.array([[x, pearson_score(dataset, user,
            x)] for x in dataset if x != user])

    # Sort the scores in decreasing order
    scores_sorted = np.argsort(scores[:, 1])[::-1]

    # Extract the top 'num_users' scores
    top_users = scores_sorted[:num_users]

    return scores[top_users]
```

Figure 9.13: Appendix C.7, Find Similar Users Function

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user = args.user

    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    print('\nUsers similar to ' + user + ':\n')
    similar_users = find_similar_users(data, user, 3)
    print('User\t\t\tSimilarity score')
    print('-'*41)
    for item in similar_users:
        print(item[0], '\t\t', round(float(item[1]), 2))
```

Figure 9.14: Appendix C.8, Similar Users Main Function

```
import argparse
import json
import numpy as np

from compute_scores import pearson_score
from collaborative_filtering import find_similar_users

def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find the movie recommendations for the given user')
    parser.add_argument('--user', dest='user', required=True,
            help='Input user')
    return parser
```

Figure 9.15: Appendix C.9, the Movie Recommendation Parser

```
# Get movie recommendations for the input user
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        raise TypeError('Cannot find ' + input_user + ' in the dataset')

    overall_scores = {}
    similarity_scores = {}

    for user in [x for x in dataset if x != input_user]:
        similarity_score = pearson_score(dataset, input_user, user)

        if similarity_score <= 0:
            continue

        filtered_list = [x for x in dataset[user] if x not in \
                dataset[input_user] or dataset[input_user][x] == 0]

        for item in filtered_list:
            overall_scores.update({item: dataset[user][item] * similarity_score})
            similarity_scores.update({item: similarity_score})

    if len(overall_scores) == 0:
        return ['No recommendations possible']

    # Generate movie ranks by normalization
    movie_scores = np.array([[score/similarity_scores[item], item]
            for item, score in overall_scores.items()])

    # Sort in decreasing order
    movie_scores = movie_scores[np.argsort(movie_scores[:, 0])[::-1]]

    # Extract the movie recommendations
    movie_recommendations = [movie for _, movie in movie_scores]

    return movie_recommendations
```

Figure 9.16: Appendix C.10, the Get Movie Recommendation Function

```
if __name__=='__main__':
    args = build_arg_parser().parse_args()
    user = args.user

    ratings_file = 'ratings.json'

    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())

    print("\nMovie recommendations for " + user + ":")
    movies = get_recommendations(data, user)
    for i, movie in enumerate(movies):
        print(str(i+1) + '. ' + movie)
```

Figure 9.17: Appendix C.11, the Movie Recommendation Main Function