

HELLENIC MEDITERRANEAN UNIVERSITY

Department of Electrical and Computer Engineering

Department of Informatics Engineering T.E.

BSc Diploma Thesis

DEVELOPMENT OF A RIDE-SHARING APPLICATION

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΥΠΟΣΤΗΡΙΞΗΣ ΔΙΑΜΟΙΡΑΣΜΟΥ ΘΕΣΕΩΝ ΕΠΙΒΑΤΙΚΩΝ ΟΧΗΜΑΤΩΝ

Written by

Symeon Georgios Chinopoulos

Register ID: TP4502

Supervised by

Eleftherios Kosmas

Heraklion, March 2024

© Symeon Georgios Chinopoulos, 2024

Copyright © Symeon Georgios Chinopoulos, 2024

All rights reserved.

The approval of the thesis by the Department of Electrical & Computer Engineering of the Hellenic Mediterranean University does not necessarily imply acceptance of the views of the author by the Department.

Acknowledgements

A special thanks to my family for being there for me and my professor for his decisive contribution to the completion of this thesis.

© Symeon Georgios Chinopoulos, 2024

Abstract

Daily commuting is an important part of people's daily lives, as there is a real need to get to and from work, school and beyond. Transportation engineers often propose various solutions to meet this growing need which, apart from better serving people, also help to reduce traffic congestion. Therefore, innovative solutions such as ride-sharing are being developed. Ride-sharing is a way of splitting the moving costs of a vehicle and enjoying the journey with other people. In this thesis, we focus on the design and implementation of an academic ride-sharing application by taking advantage of the possibilities offered by smartphones in terms of broad usage and capabilities. This application allows users to post, search, and participate in routes, whilst at the same time promoting vehicle sharing among fellow students who mainly have similar time schedules in university courses and thus, follow similar daily routes. Moreover, we study and compare the current most popular technologies and we justify our selections to design and develop our ride-sharing application. We then present the set of the scenarios we used to test and guarantee the correct and seamless functionality of our application. Finally, we discuss possible future extensions of the developed application, to further promote ride-sharing applications.

© Symeon Georgios Chinopoulos, 2024

Περίληψη

Οι μετακινήσεις αποτελούν ένα σημαντικό και χρονοβόρο μέρος της καθημερινότητας των ανθρώπων, καθώς υπάρχει πραγματική ανάγκη για μετάβαση από και προς την δουλειά, την σχολή, το σχολείο και όχι μόνο. Για την συνεχώς αυξανόμενη αυτή ανάγκη προτείνονται συχνά διάφορες λύσεις οι οποίες, πέρα από την καλύτερη εξυπηρέτηση του κόσμου, συμβάλλουν και στην μείωση της κυκλοφοριακής συμφόρησης. Με αυτόν τον τρόπο, αναπτύσσονται καινοτόμες λύσεις, όπως η κοινή χρήση ιδιωτικών μεταφορικών μέσων (ride-sharing). Το ride-sharing αποτελεί έναν τρόπο να επιμεριστεί το κόστος μετακίνησης ενός οχήματος μεταξύ δύο ή παραπάνω ατόμων, ενώ ταυτόχρονα τα συμβαλλόμενα μέρη απολαμβάνουν το ταξίδι με άλλους ανθρώπους. Στην παρούσα διπλωματική εργασία, εστιάζουμε στο σχεδιασμό και την υλοποίηση μιας εφαρμογής διαμοιρασμού διαδρομών, εκμεταλλευόμενοι τις δυνατότητες που προσφέρουν τα έξυπνα κινητά τηλέφωνα όσον αφορά την ευρεία χρήση και τις δυνατότητές τους. Η εφαρμογή αυτή επιτρέπει στους χρήστες να αναρτούν, να αναζητούν και να συμμετέχουν σε διαδρομές, ενώ ταυτόχρονα προωθεί την κοινή χρήση οχημάτων μεταξύ συμφοιτητών που έχουν κυρίως παρόμοιο ωράριο στα πανεπιστημιακά μαθήματα και, ως εκ τούτου, ακολουθούν παρόμοιες καθημερινές διαδρομές. Ταυτόχρονα, μελετάμε και συγκρίνουμε τις πιο δημοφιλείς τεχνολογίες που υπάρχουν σήμερα και αιτιολογούμε τις επιλογές μας για την σχεδίαση και ανάπτυξη της εφαρμογής ride-sharing. Ακόμη, παρουσιάζουμε το σύνολο των σεναρίων που εφαρμόσαμε για να ελέγξουμε και να εγγυηθούμε τη σωστή και απρόσκοπτη λειτουργία της εφαρμογής μας. Τέλος, συζητάμε πιθανές μελλοντικές επεκτάσεις της εφαρμογής που αναπτύξαμε, ώστε να προωθήσουμε ακόμη περισσότερο την φιλοσοφία των εφαρμογών διαμοιρασμού διαδρομών (ride-sharing).

© Symeon Georgios Chinopoulos, 2024

This page was intentionally left blank.

Contents

Acknowledgements 3
Abstract 4
Περίληψη 5
Contents 7
1. Introduction 10
1.1 Existing Applications 11
1.1.1 UberX/UberPool 11
1.1.2 BlaBlaCar 12
1.2 Our proposal 13
1.3 Mobile-Frontend 15
1.4 Backend 16
1.5 Database 16
2. Technologies 17
2.1 Mobile 17
2.1.1 React Native 17
2.1.2 Flutter 18
2.2 Server-side 19
2.2.1 Node.js 19
2.2.2 Spring Boot (Java) 20
2.2.3 Flask (Python) 21
3. Design 24
3.1 Mobile Mock-up 24
3.2 Server-side 26
3.2.1 Database 26
3.2.2 Server 29
4. Implementation 30

4.1 User Interface 30
4.2 Server implementation 37
4.3 Application-to-Server Communication 38
4.4 Problems and Solutions 39
5. Testing 41
5.1 Server-side Application 41
5.2 Mobile Application 41
5.2.1 Mobile Application Testing 41
5.2.2 Scenarios 42
5.2.3 Flutter Testing Environment 45
6. Conclusions & Future Work 46
6.1 Conclusions 46
6.2 Future Work 46
7. References 48

Figures

Figure 1: Sign in 25
Figure 2: Search Page 25
Figure 3: Navigation Menu 25
Figure 4: Search Results 25
Figure 5: Route Detailed Page 25
Figure 6: Messaging Page 25
Figure 7: Create Ride Page 26
Figure 8: Database Schema 28
Figure 9: Sign up Page 31
Figure 10: Sign in Page 31
Figure 11: Search Page 32
Figure 12: Menu 32
Figure 13: Searched Route Details 32
Figure 14: Ride Request Form 32
Figure 15: User's Profile Page 33

© Symeon Georgios Chinopoulos, 2024

Figure 16: User's Ratings Page 33
Figure 17: Ride Creation Steps 33
Figure 18: Wish list Route and Creation Page 34
Figure 19: Add Vehicle Page 35
Figure 20: Users Vehicle List 35
Figure 21: User's Route Page as Driver only 36
Figure 22: User's Routes Page as Participant 36
Figure 23: Route Details Page 36
Figure 24: Participants profile Page 36
Figure 25: Notification Page 37
Figure 26: Notification Details 37

1. Introduction

In the past few years, the need for sustainable transportation solutions has significantly increased. Rising personal commuting costs, extremely increased fuel prices, and growing awareness of climate change among young adults, especially in developed countries, have underlined the need for a new commuting option. A prominent choice that has spread in the group mentioned above is ride-sharing [1].

Ride-sharing is the concept of vehicle-sharing among two or more people who have the same or a similar destination. It has been proved to be a promising solution to deal with a variety of prominent economic, environmental, and social challenges encountered in urban areas. It can reduce (due to sharing) both travel costs, such as fuel and tolls, and general costs, like maintenance vehicle costs. Moreover, it contributes to the decrease of air pollution and traffic congestion, while making parking spaces availability higher. Also, ride-sharing can make commuting more comfortable than using public transport, while enjoying the company of other passengers. Studies [2] have shown that commuting involving more than one passenger (as company) helps to reduce stress levels caused from driving and loneliness. It also forms social opportunities and strengthens bonds within the users' communities.

As is commonly believed, the major competitor of ride-sharing is the traditional (usually) public transportation means. Ride-sharing offers more personalized routes and schedules, catering to specific needs, while traditional means of transportation, like public buses or trains, offer a more structured and regulated environment, operating on fixed schedules and routes. In terms of safety, traditional transportation is frequently run by reputable privately or publicly held companies which hire experienced and professional drivers and must comply with safety guidelines. In terms of cost-effectiveness, ride-sharing frequently involves cost-sharing among participants, which may result in lower individual costs when compared to taxis or monthly transportation fees. The influence on the environment is another noticeable consideration. While public transportation systems frequently give preference to environmentally friendly means of transportation, ride-sharing reduces total carbon emissions by optimizing vehicle occupancy and eventually comes in line with sustainable practices. Furthermore, there are differences in the social aspects with ride-sharing offering passengers the chance of networking and socializing, establishing community bonds, in comparison to the most likely anonymous travel experiences when choosing the public transportation.

© Symeon Georgios Chinopoulos, 2024

However, there are major potential doubts and concerns raised when considering using an application that requires trusting strangers to drive you somewhere or letting them in your car. The two concerns of outmost importance, being personal safety and security, which holds both ways (i.e. for the drivers and for the passengers), need to be addressed. Additionally, there is a certain lack of trust and familiarity, which makes it more likely that one may run into strange behavior or awkward circumstances. It is also important to think about the possible repercussions of letting someone you do not know into your personal space since in certain cases, a person can feel overwhelmed or insulted.

To generalize, entering a vehicle of an unknown person bears rare but comparable hazards to one's personal safety and security. A lack of prior knowledge or a firm grasp of the driver's abilities and personality puts you at risk. These risks might include being the target of criminal activity, careless driving, driving style or simply taken advantage of in precarious circumstances. We make a first step to address these concerns by giving the application community the ability to rate and comment on each user, either driver or passenger and make those interactions available for every user to read them.

The goal of this thesis is to contribute to the growth of ride-sharing in Heraklion, while taking into consideration the mobility preferences of the new generation. The main idea is to focus on creating a viable and appealing application to be offered by the Hellenic Mediterranean University to its students who reside away from the campus, given that the campus is far from the city center and only few students own private vehicles at such young age. The application has been designed to support academic user’s profile, using academic credentials.

1.1 Existing Applications

In this chapter, we describe some of the well-known ride-sharing applications and discuss their main advantages and disadvantages, which are summarized in Table 1.

1.1.1 UberX/UberPool

Uber is a service company that provides two options of car sharing transportation: UberPool [3] and UberX [4]. Their main difference is that in the case of UberPool a single ride can be shared by one or more passengers, reducing the closing price per route. On the contrary, UberX offers only solo passenger rides on demand, with minimal waiting time, which in the

essence is the basic service that Uber offers. As a result, UberPool is closer to the ride-sharing philosophy compared to UberX, which provides a dedicated car for each passenger, or per group of passengers, being the least eco-friendly transportation solution in discussion. In the following, we provide more details about UberPool, due to its resemblance to the idea on which our application is based.

UberPool has the advantage of being an on-demand ride service with minimal waiting time. This means that passengers do not have to plan for any rides, but they can just hop on and off from a ride instead. Any passenger that requests for a ride can also invite one friend (with a maximum of two passengers) and each ride can serve up to two requests and four passengers in total. In that case UberPool can claim to significantly reduce the total carbon footprint. However, provided no external users join the ride (therefore one request per ride), the experience is close to UberX, but with lower fare cost. Additionally, fare prices are transparent, predictable, and the passengers know the cost prior to the ride, while transactions are cashless and convenient.

Sharing a ride with other people comes with a variety of disadvantages. Usually sharing a ride can result in longer travel times compared to solo rides. Uber in that case says that the UberPool should not take more than 10 min longer for the same trip compared to UberX, including any relative delays, e.g. drop off and pick up of passengers, and rerouting the ride to serve the passengers before reaching the destination. On top of that sharing rides with strangers can have a negative effect on the experience, as some passengers might not be in their best behavior all the time. This can affect the rest of passengers' comfort and privacy since they would most certainly wish for a quite ride. Finally, since it is extremely hard to change the route destination on the fly, this option is not supported by UberPool [5].

To address the previously mentioned safety concerns regarding both UberPool and UberX, Uber has introduced several incident prevention tools included in the app from a rider's perspective. These include an emergency assistance button, ability to share one's trip with trusted contacts - fellow Uber users - using GPS tracking, RideCheck - uses sensors and GPS data – to help with unexpected long stops and phone number anonymization [6].

1.1.2 BlaBlaCar

BlaBlaCar [7] [8] is a popular French Ride-sharing platform that allows drivers to post their journeys, and passengers to catch a ride along the way. To sustain the company, BlaBlaCar levies a 25% commission fee (min 1€) to the passenger, on top of the prices that drivers set.

It just provides the platform for both drivers and passengers instead of the service. It is mostly used for journeys between cities rather than daily rides in the city.

One of the main advantages of BlaBlaCar is that the cost of transportation is divided among the - up to four - passengers and the driver. So, in comparison with traditional transportation, it usually is more cost effective. In addition, as more people travel with the same car the journey becomes more eco-friendly, leading to a smaller carbon footprint. Also, traveling with company instead of solo, it can help build community and it provides company. One more advantage comes from the fact that the routes are more flexible for traveling between cities and regions, since the public transportation option may be limited or less convenient, while cars are usually more comfortable. Finally, according to acclaimed BestReferral Team [9] , BlaBlaCar platform ensures safety, by obligating both drivers and passengers to verify their identities and email addresses and provide credentials, such as age, driving record, vehicle type and license plate number. In addition, to address the fact that for many women the possibility of ridesharing can cause insecurity, BlaBlaCar has created the “Ladies Only” tool [10], which allows female members to organize a ride where both the driver and all the passengers are women.

On the negative side, availability of BlaBlaCar may be smaller for smaller towns and rural areas, since less people will make a journey to that destination. Since BlaBlaCar also relies on individual drivers and their schedules, finding journeys that match desired time and routes can become challenging in small towns and rural areas. Also, it may lead to inconvenient drop-off/pickup locations, resulting in huge delays especially for non-direct/multi-city trips.

1.2 Our proposal

Inspired by other ride sharing platforms that are immensely popular abroad, like the BlaBlaCar application (Section 1.1.2), this thesis explores the potential to improve and adjust a platform from a national grade service to a limited university community by designing and implementing a pilot application. The goal is to provide an application that can become popular among university students all over the country, especially in cities with similar university locations as Heraklion. This requires the design of an easy-to-use, great looking user interface, as described in Table 2 and User Interface and an overall cost-effective and environmentally friendly result, which encourages collective mobility, reduces individual carbon footprints, and promotes sustainable living.

The application provides a platform in which users can create and/or participate in shared rides for daily commute. Unlike UberPool, these rides are created ahead of time, just like BlaBlaCar but for short distance rides, forming a concept that is beneficial for people with

repeated commutes, like co-workers or university students that now have the option for an alternative commuting method. Meanwhile, passengers are able to express their interest in a route that is not yet available, forming a wish-list of routes. In this way, when a driver schedules/creates a new ride that coincides with a passenger's interest, the passenger will receive notification of the new route available. This would provide a matchmaking type function, a process that increases the usefulness of the application, since it allows the efficient matching of passengers with drivers.

Furthermore, the app provides the ability for communication between a driver and a passenger that requests to ride along, via text messages. This creates a channel of communication, protecting the privacy of users, since there is no need for sharing their social media profiles or other personal data. To further improve safety during rides, we also implement a two-step confirmation feature, for entering a vehicle, which includes both the confirmation of the driver and the passenger, that the passenger has entered the vehicle and is indeed the person shown in their credentials.

Finally, since the application will be piloted at the University level, the potential sign in of a user with their institutional account, as we discuss in Section 6.2, which increases safety and security, and ensures that the real details of the person with whom we share the vehicle are known and verified. In this way, any problems that may arise with the uncontrolled creation of fake or anonymous profiles are being minimized. The only additional information required is for the user, who desires to create a driver profile, to register their vehicle. This means that users can be divided into two types of users: drivers and passengers and only drivers can create a route. This design should be enhanced in case the application is further expanded outside of the University.

© Symeon Georgios Chinopoulos, 2024

<table>
<tr><th>UberPool</th><th>UberX</th><th>BlaBlaCar</th><th>Our Application</th></tr>
<tr><td>Single ride is shared with strangers</td><td>Solo passenger rides</td><td colspan="2">Single ride is shared with strangers</td></tr>
<tr><td>1 or 2 passenger requests per ride</td><td>Up to 4 passengers per ride request</td><td colspan="2">Up to 4 passengers, depends on the driver</td></tr>
<tr><td colspan="2">Minimal waiting pick-up time</td><td colspan="2">Depends on the driver credibility</td></tr>
<tr><td>Up to 10 minutes delay (claimed by Uber) to destination arrival due to pick-ups/drop-offs</td><td>No extra delay</td><td>Huge delays, especially for non-direct/multi-city trips</td><td>Small delays due to passenger pickups/drop offs</td></tr>
<tr><td colspan="2">On demand service</td><td colspan="2">Prescheduled rides, relies on schedules of individual drivers</td></tr>
<tr><td>Shared car without the ability to detour or change destination</td><td>Dedicated car per request with the ability to detour or change destination</td><td colspan="2">Shared car, depends on the driver about detours</td></tr>
<tr><td colspan="3">Upfront cost prediction and cashless transactions</td><td rowspan="2">Free of charge</td></tr>
<tr><td>Lower fare prices per request compared to UberX</td><td>Higher cost per request compared to UberPool</td><td>The cost of the ride is divided between driver and passengers. Platform charging fee is 25% on top of driver’s charge</td></tr>
<tr><td colspan="2">Used for in cities transport</td><td>Used for intercity transport</td><td>Used for transport from/to University</td></tr>
<tr><td>Slightly more eco-friendly than UberX</td><td>Least eco-friendly</td><td colspan="2">Eco-friendly option, since drivers and passenger are served from a route compared to just passenger for Uber</td></tr>
<tr><td colspan="2">Convenient pick-up/drop-off points</td><td colspan="2">May lead to inconvenient pick-up/drop-off locations</td></tr>
<tr><td colspan="2">Suitable for daily commute</td><td>Suitable for intercity commute</td><td>Suitable for daily commute if it fits one’s needs</td></tr>
<tr><td colspan="2">Users are not verified by the app</td><td>Users need to be verified using ID and valid email address</td><td>Verified using university credentials</td></tr>
</table>

Table 1: Comparing Existing Applications.

1.3 Mobile-Frontend

We needed to set some requirements to make a choice about which framework we are about to use for our application development. The goal is to create an application that is snappy with good looks, which are maintained across different platforms with a single code base.

This will also allow us in the future to maintain the application more easily since most of the code is reusable. Additional we want to keep we want to keep the application size to the minimum, but it's not a priority. Finally, a reasonable learning curve of the technologies would be considered a bonus.

1.4 Backend

The requirements for the backend are a bit more straightforward. Since we needed a framework that should have a very good ORM -Object Relational Mapping- for our database, be lightweight enough without compromising and provide exceptional security. Another desired property is strong data type safety, while promoting data integrity. Also, having a built-in server suggests a nice addition and finally the framework needs to be suitable for a REST API.

1.5 Database

The application requires a basic set of data to be stored, which are generated from user activity. This dataset consists of simple information about each user such as credentials, name, and birthdate; vehicle information such as model, color, and license plate; route information that includes but not limited to departure and destination of the ride, date-time, passengers, vehicle, and driver. Also, information about each user's activity like ride participation requests, messages, notifications, and comments needs to be included in this dataset.

In addition, we want a database that offers strict data types and integrity, while the application operates as a pilot on a university basis with a small number of users. Thus, a solution for a simple database technology will be sufficient for this application, such as SQL. Therefore, we select PostgreSQL as it a high-performance solution and we already have a working knowledge.

2. Technologies

The choice of the right technology is a crucial aspect of the development process and must be aligned with the application requirements, since it has a direct impact on the overall functionality, performance, and scalability of it.

2.1 Mobile

In the last few years, the popularity of mobile applications has increased and so has the number of mobile application technologies. The choice of programming language is crucial for the development of the application. After considering the popularity [11] of the current used technologies we conducted research on the benefits and drawbacks for the two most popular UI frameworks. The two most popular options for 2023, by developers preferred choice, React Native is chosen with 42 percent and Flutter with 39 percent. Also, as flutter being a recent technology compared to React Native, there is a smaller collection of apps that are developed by Flutter.

2.1.1 React Native

React Native [12] [13] is a JavaScript open-source UI framework created by Facebook, which is used for developing multi-platform apps like Android, iOS, MacOS and Windows while using a single codebase to create the apps. React Native uses native components to create the UI, but it is not a native language, for both Android and iOS meaning that the app will look slightly different in every platform, there are also a lot of external UI kits to make the development easier.

Overall React Native provides more UI parameterizations compared to Flutter. The framework uses JavaScript as a bridge to connect to native components so there is a small added overhead to the performance but with the advantage of fully using the phones hardware, although sometime that is not the case. The use of native components ensures that applications with React Native UI are running as smoothly and fast as native developed applications and can be lighter than native technologies such as Java and Swift resulting usually in smaller application sizes with exceptions. As already mentioned above React Native uses JS as bridge therefore uses it to compile as well, that allows most of the code to be reused. The only drawback being that while reusing the codes between platforms there is a lack of resources and experienced developers. Finally, React Native on top of a great and growing community offers documentation with general information but a bit disorganized

compared to Flutter, along with the community comes a big number of third-party community driven plugins.

Flutter		React Native	
+	-	+	-
Uses important build in UI components that replace the platforms native components	Creating complex and custom UI components is more difficult and must more well documented	Uses platforms native UI components which ensures smooth and fast applications	Using JavaScript as a bridge for interacting with native components, adds a performance overhead because it can really use the device hardware
Supports many hardware acceleration features such as improving performance	Debugging environment is hard to get used to and differs from other platforms	Smaller size apps, even from native apps, as it uses JavaScript which is lighter than native languages like Java and Swift	Requires higher SDK version of 21 that Flutter which is 16
No bridge for interacting with native components improving the overall performance	Applications are usually larger in size	React Native declarative programming style allows for better UI development that is responsive and great looking	UI development may require both React Native and native components knowledge thus being more complex and time consuming
Provides various UI components and tools resulting in easier development with more beautiful and dynamic user interfaces	Framework is using Dart, and it is a relatively new language and very popular	Allows code to be reused between platforms	Is harder to debug that other frameworks cause some features may not be available in some platforms
Hot reload and hot restart features reduce debugging time as developer can see changes faster	Not native	Extensive collection of UI kits to use instead of native UI components	Code reusability often is more difficult cause there is a lack of experienced developers and support
Code is written in single programming language and compiled for each platform – easier to maintain		More popular while having a bigger and more active community	
Detailed and organized documentation		JavaScript is easier to learn	
Increasing popularity and community size			

Table 2: Comparison of Flutter and React Native.

2.1.2 Flutter

Flutter [12] [13] [14] is a Dart open-source UI framework that was created by Google and released in 2017. The framework is mostly used by developers for application User-Interface development for multiple platforms, including – but not limited to – Android, iOS, web, and Windows with a single codebase. Flutter has a similar philosophy to React Native and it uses widgets, which are small parts of the UI that composite the whole app.

The first big advantage of Flutter is that it helps create and maintain a similar UI across multiple platforms of devices and provides several built-in widgets out of the box resulting in a great User-Interface that replaces native UI components. However, Flutter only supports 2D graphics but that is not a problem in our case. Because Flutter compiles to native Android and iOS languages, there is no need for a bridge to interact with native components. As a result, there is no overhead delay. In addition to improving development, it reuses the same codebase for different platforms, and offers high performance applications with the drawback of applications being bigger in size compared to native apps. Still though, as a newer framework, codebase reusability can be improved. Flutter has well-structured and straightforward documentation provided by Google alongside a strong community, is something that should be taken into consideration since during development with any technology the developer will usually have to do some research for problem solving.

After the comparison conducted in the preferred technology to be used for development is Flutter. This framework is chosen mainly because it has better documentation and is supported by Google. Both technologies fulfill our requirements, but Flutter has better performance and offers a consistent UI look across different platforms; these make it a better choice for creating snappy applications. Additionally, Flutter is a new framework that has grown in popularity.

2.2 Server-side

In this section we discuss the advantages and disadvantages of three of the best and most popular back-end technologies that we found, according to sources [15] [16], and which one we choose after comparing them. Those technologies are, by popularity rate, Node.js, Java with Spring Boot and Python with Flask or Django and both being excellent we choose Flask since it was closer in logic with the Spring Boot framework.

2.2.1 Node.js

Starting with the most popular one, Node.js [17] [18] is an open-source cross-platform server-side JavaScript environment that runs on the V8 JavaScript engine; hence it is considered to have an easier learning curve as JavaScript requires less time and effort to learn the language [17]. One of the key benefits of Node.js is speed as it is based on an event loop runtime environment that manages multiple concurrent requests quickly that are not CPU intensive tasks. Also, it uses non-blocking or asynchronous input/output. As a result, it processes a new request before the previous one is finished and when one request is finished it returns immediately after getting a response, thus allowing the framework to manage many requests at once without any noticeable delays or performance impact. An

adding benefit is that slow connection does not slow down the rest of the requests. Considering the above, Node.js is an event-driven and single-threaded environment, characteristics which make it a great choice for real-time applications such as chat and video streaming applications or highly interactive websites in general.

Another feature of Node.js is the build error handling mechanism that allows catching errors at runtime and managing them. The errors are thrown as events that can be caught by the code, which is similar to Java's try and catch approach but without exceptions, while it supports the use of error objects, providing more control over the event that caused the error. Last of the benefits Node.js is suitable for fast website and web application development. In addition, it is very useful for building microservices, which allows the developers to build many small distinctive parts that work together as a system and have the ability to test them individually.

Node.js' cause of the asynchronous single threaded working philosophy is bound to have some drawbacks. One of which is not suited for heavy CPU workloads while the rest of the request have no CPU time, as a result it will bottleneck the application. However, with an update in 2018 introducing multithreading with the worker thread module in Node.js. This module allowed to run multiple Node.js instances inside a process with shared memory, which helped with CPU intensive tasks, but it did not make Node.js a high-performance multithreading technology. Furthermore, as the APIs used in Node.js constantly changing affects the stability of the framework and sometimes backward-incompatibilities are created which creates bugs. This problem is extended by the lack of library support. Finally, it makes it a bit difficult to use relational databases since it is less popular than non-relational databases, because Node.js is hugely driven by the community and more well-maintained libraries for non-relational databases are available.

2.2.2 Spring Boot (Java)

Spring Boot [19] [20] [21] is the most popular Java framework for building micro services and REST APIs. The framework comes with automatic configuration functions out of the box helping programmers to reduce developing time while offering great flexibility. Additionally, it provides the option to use either XML or annotations for configuration. Using annotations is a much faster configuration choice that requires less steps. Also, it provides unit and integration testing. Another great advantage of Spring Boot is that it comes with an included Tomcat server, which can also be changed, by replacing the default dependencies, to services like Undertow and Jetty. As a framework supporting multithreading features is capable of handling bigger CPU intensive tasks without creating a bottleneck for the rest of

the requests, while saving memory space using bootstrapping techniques that also make the applications load quicker.

On the other hand, Spring Boot fails to provide a versatile ability to manage the dependencies that it adds by design, some of which mostly remain unused. As a result, the developer has little to no control over the file size of the application. Extending on the previous issue, the framework makes modifications very difficult, unless the developer has a strong knowledge of the framework system and history that applies to troubleshooting as well. Another drawback that comes with the use of the autoconfiguration is the unnecessary use of resources that are taken up from enabled features and the framework being fully configured, without them being utilized. As already mentioned above, Spring Boot is designed with microservice in mind, and it works seamlessly on these applications, but is not recommended at all for large monolithic applications. Finally, the Spring Boot has only the basic set of tools, lacking some major automations like converting legacy Spring code.

2.2.3 Flask (Python)

The last framework we will talk about, and an exceedingly popular one is Flask. Flask [22] is a python based open-source micro-framework. It is a lightweight and scalable framework because it allows developers to have a particularly good control over the application architecture, libraries, and extensions. It also reduces the size of the application as it will only pack what is necessary. Another reason that makes it a light framework is because it only packs what is necessary and the developer has a quite simple base to start with, while it supports modular programming which means functionality can be split into interchangeable distinct modules. Speaking of programming advantages, developers can easily find workarounds to tackle problems. Finally, there is a large community supporting the framework and the libraries that come with it. On the other hand, with this lightweight architecture philosophy Flask lacks a large toolbox, there is a workaround to that problem by adding more extensions and libraries which defeats the purpose of lightweight and fast as it will start to slow down the application. The advantage of a framework to provide workarounds comes at the cost that every new developer that undertakes a running application has a hard time understanding what is going on. Finally, the ability to be very flexible even to new technologies makes it very hard and costly to maintain the application and the technologies if they become obsolete.

Node.js		Spring Boot		Flask	
+	-	+	-	+	-
Has its own package module which is already has become huge and continuous to grow – npm	Unstable API with lack of strong Library support system	Production level auto configuration for all components	Lack of control in configuration and dependency managing – more resources are used	Suited for micro services API	Lacking basic tools – uses a lot of extensions
Active community	Community supported libraries	Standalone java app with embedded Tomcat server	Not suited for large scale applications	Simple, flexible, and easy to scale	High complexity app codes caused by versatility
Easy to learn – uses JavaScript	Asynchronous programming model is hard to maintain	Time efficient development	Time consuming framework conversion from Spring-to-Spring Boot	Lightweight framework	Costly to maintain
Scalable and high-performance apps	Less capable with CPU intense task	Easy access to DB and queue services	Lack of tools	Good documentation	Developers create their own Object Relation Management for DB
Cross platform development		Large variety of plugins		Gives developers the ability to have control over the framework	The use of modules is seen as third-party involvement lowering security
Fast and cross platform development		Use of annotations no need for xml configuration		Secure cookies	
		Great for micro services			
		Good documentation			

Table 3: Comparison of backend technologies.

While all the above technologies offer great features, as shown in Table 3, there are some key differences between them as can be seen in Flask despite being lightweight, like Node.js, after installing all the necessary extensions - modules, some of those come as standards in other frameworks, becomes quite overloaded and slow, also the frameworks manage them as third-party involvement lowering security. As a result, Flask is excluded as a lightweight option. Node.js on the other hand is a great overall framework and surely very

popular on but the lack of supported libraries for relational databases makes development slower. Finally, combining the above with our preference choice is a relational type of database, like PostgreSQL. We opted for Spring Boot since it provides a more efficient way to access data, despite its drawback of restricted fine tuning (since it disallows complete control over the framework).

3. Design

3.1 Mobile Mock-up

During the designing phase of the UI, we followed some basic rules with our main goal being to provide an easy-to-use application for the user. To accomplish that, we needed a simple navigation menu that allows users to move seamlessly between different parts of the application and quickly find what they are searching for. That contributes to optimizing the user's experience.

Another general rule we followed was to keep clickable elements large enough so it will be easily clickable by the user. Combined with an uncluttered page that shows only the necessary information, because it is not possible to show every bit of information on one page as it will be overcrowded and keeping elements clickable will be harder. Simultaneously, this allows us to create text larger and easier to read.

All the above combined with keeping a consistent and simple style in the application, help to increase overall user experience, encourages them to continue use and explore the application's features, saves them time and they are more likely to suggest the application and increase the community that are part of. An adding bonus of a simple design that is not over bloated help with faster loading times.

Initially, the user will open the application and see the welcome page with the credential input fields (email and password), as can be seen in Figure 1. Then, after a successful sign in, the search page will be displayed (Figure 2), where the user can search for a route based on its location, destination, and date. Also, on each page of the application, there is a navigation menu (Figure 3) that can be accessed by clicking on the 3 lines on the top left. There, you can access the user's basic details along with a profile picture, as well as browsing options for messages, history, add route, and profile.

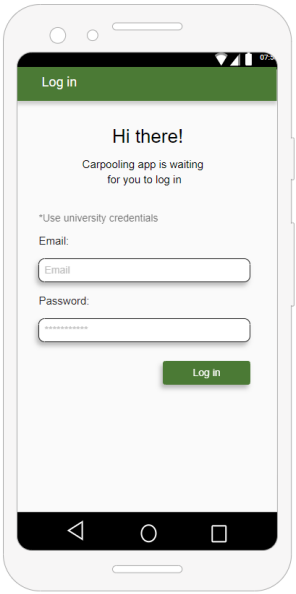


Figure 1: Sign in



Figure 2: Search Page

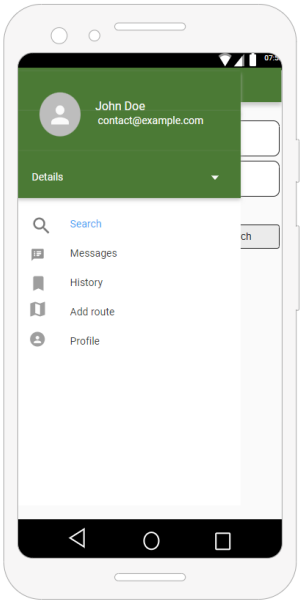


Figure 3: Navigation Menu

According to the user's search criteria, results are displayed (Figure 4) with possible routes along with information about them, added by the corresponding driver.

As can be seen in Figure 5, after the user selects a route, detailed information of this route is displayed, along with its trace on the map. Then, the passenger can join the route by making a request to the driver, which the driver can either accept or reject it. Finally, the user that requests a ride land on a messaging page to communicate with the driver, a process shown in Figure 6.

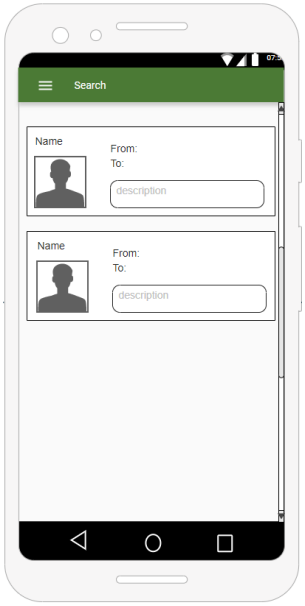


Figure 4: Search Results

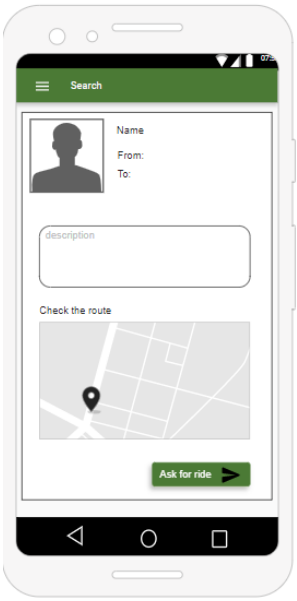


Figure 5: Route Detailed Page

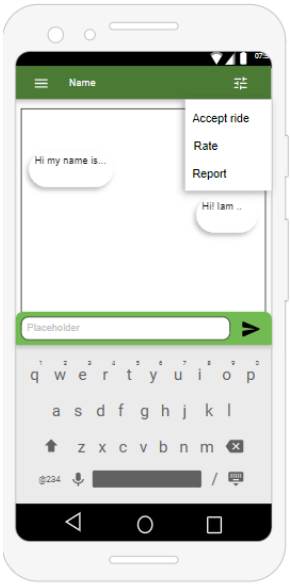


Figure 6: Messaging Page

By selecting the add route option shown in Figure 3, the route creation page appears (Figure 8). In that page the driver adds starting and destination points, the departure and expected arrival time, and can also leave a quick message with more details for the route. As seen in the right part of Figure 7, when a user clicks on the text field to fill the starting point “from”, and on the field “to” to fill the destination point, a map is displayed to provide visual representation. Finally, provided everything is filled up, the user can click on the “Create Ride” button and create the new ride.

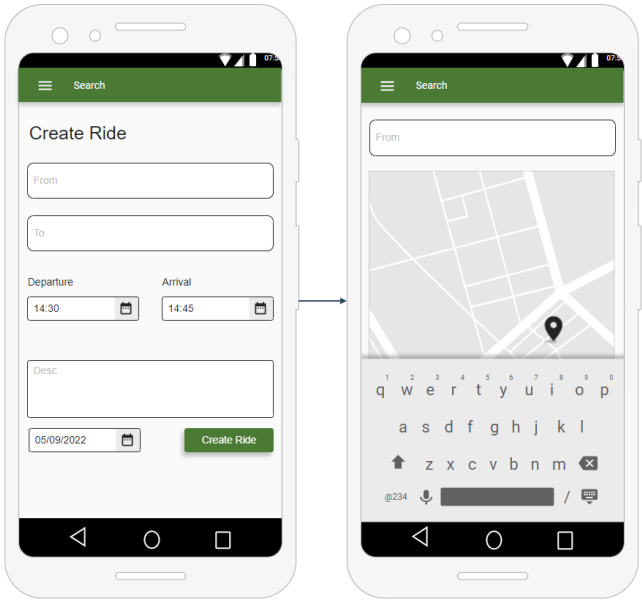


Figure 7: Create Ride Page

3.2 Server-side

3.2.1 Database

After assessing the requirements of the application, we concluded that a basic database schema should include the following data per user as they are described below, in Figure 8. The choice of a relational database and more specifically PostgreSQL was made considering that offers exceptional performance, provides data integrity, it is very easily manageable, and we have no need for large blob data storage yet.

The first and most obvious data that needed to be stored are the user information. For our purposes we only store a set of basic data including the user’s full name (‘fullname’ as it shown in Figure 8) email, address, birthday, and gender with some meta data that automatically created like role, enabled - “DELETED” or “ACTIVE”- and id. All that is stored in the “userinfo” table. Also, there is the need to store some user dependent data like the info

of user's car that he will use for route creation as a driver only. So, we created a table called 'vehicle_info' which includes basic information about a user car like brand, model, color, license plate and the user's id, along with some metadata like the status of the vehicle which can be "deleted" or "active".

Users with a car can create a route as a driver or can join a route as passenger. In the routes table we store all the route basic info like the start and finish addresses, when it is created and when it will depart, how many people are allowed in the car, if they are allowed in the backseats only, the id of the vehicle that will used and the user that created the route but not the passengers that are participating. It also stores the suggested route using coordinates that are automatically created and has a status which can have as value "ACTIVE", "DELETED" and "CANCELLED". Alongside tags that make the route easily searchable.

Every user is stored as a participant with a role describing them as a passenger or a driver, also the start and finish addresses are stored for every user individually and their status which can have as value "LEFT", "PARTICIPATING", "READY", "ENTERED" and "NOT_ENTERED". Also, in this table the route id and participant user id are stored.

Each user has the option to add a wish list route, with starting and finishing addresses, preferred date and created date, along with the street names of the suggested route that is automatically created in the "directions" field. Directions field is used for matchmaking when route is created because it holds all the in between addresses of the route. This is used so there is no need for recreating all this information based on the start and destination of the route dynamically.

The application needs to notify users and save those notifications as users may not be online at the given moment or they need to view a given notification again from inside the application. The "users_notification" table holds data like sender, receiver, type, status, and message. Along with those fields a more versatile context field is needed which holds data as a Json string that can be varied depending on the notification type. The type and status fields can take specific values for the type are like "RIDEREQUEST", "RIDESTART", "FOUNDWISHROUTE" or "ROUTEUPDATE" and for the status are like "PENDING", "ACCEPTED", "DENIED", "DELETED", "STARTED", "FOUND", "UPDATE", "CANCELLED" or "READY".

Also, users can make comments on other users and a table is needed to store these data. In the "ratings" table we store some basic info like the comment of the rating, the rating, timestamp, from which user came from and to which user is referred to by using the user's id. Finally, every table has its own auto increment id and every id that is used in other tables, for

example in the rating table the user id is used as a reference to the user, are foreign keys in order to have some data integrity in the database.

The last table, “user_token”, stores all the necessary information about user-token keys that are given to users by the Firebase Cloud Messaging platform and are used for pushing notifications to users. Each user-token key is unique, and it gets updated every time a user logs in.



Figure 8: Database Schema

3.2.2 Server

In this paragraph, we will discuss good practices, which helped us in the building process of our REST API server. A REST API is a programming architecture style that guidelines and restricts the application towards stateless communications and cacheable data. While this type of application can be accessed through variety of communications, the most used way is over HTTPS requests via the available endpoints.

First of all, it is very important that a REST API accepts and responds with JSON payloads with data and not hypertext, as almost any technology has the ability to use JSON to send or retrieve data.

Also, the server endpoints should be named according to what entity type of data are manipulating and use nouns instead of verbs. Because the HTTP request method already has the verb that describes the action of the request, for example GET methods retrieve data, POST methods submit new data, PUT updates data, and DELETE removes data. Furthermore, the endpoints should be nested logically and group those which contain similar information, for example the endpoint '/routes' should return all routes and then a nested endpoint should exist that return a route by id "/routes/:id". Endpoints should respond with standard HTTP error codes and correspond to the problem that our server is encountering at the moment to avoid any confusion.

Finally, a REST API application should maintain good security by updating SSL certificates, user role restriction for each group and keep user in the right access group level, so each user should only access what they are supposed to access. [23]

4. Implementation

In the following section, we describe the implementation of the User Interface (Section 4.1), as well as the server (Section 4.2), and the application-to-server communication (Section 4.3).

The User Interface (UI) is the point of human-computer interaction (HCI) and communication in a device, and more precisely in our case, the way a user will interact with our application. Provided we create a mobile application, the type of UI in discussion is called mobile UI, which we aim to create easy-to-use, one-handed use, efficient, interactive interfaces in smartphone screens, optimizing the functionality of the features of touch controls, fast page loadings etc.

Our aim is to ensure our application is appealing and simplistic. To achieve that, we apply a logical content hierarchy to the application menu (Figure 12) and make sure of the presence of uniformity on the style and the theme of the application. We chose green to be the main color of the application, as a symbolization for sustainability for an environmentally friendly practice, such as ride-sharing.

Regarding the application-to-server communication and the server itself, we will describe the working principle of the server when request occurs, along with the communication between the server and the mobile application.

4.1 User Interface

Sign-in/Sign-up This screen consists of two tabs, the “Sign-in” (Figure 10) and the “Sign-up” (Figure 9). Also, it is the first screen a user comes across before entering the application. To use the application, a new user must first follow the sign-up process, by creating a new account, using his/her academic email as can be seen in Figure 9. The sign-up process occurs in the “Sign-up” tab and requires three steps. During these steps, the user provides the following information: email address, full name, preferred username, password, birthday, home address, and gender. After filling in the form, by clicking the sign-up button the user lands on the Home page where he can search for available routes. To sign-up you must have an HMU academic email, as the application checks on input the email domain. However, it does not use the University authentication portal yet (more in Section 6.2**)**. Users that are already signed-up they can just sign-in via the “Sign-in” tab (Figure 10), using the email and password provided in the sign-up process.

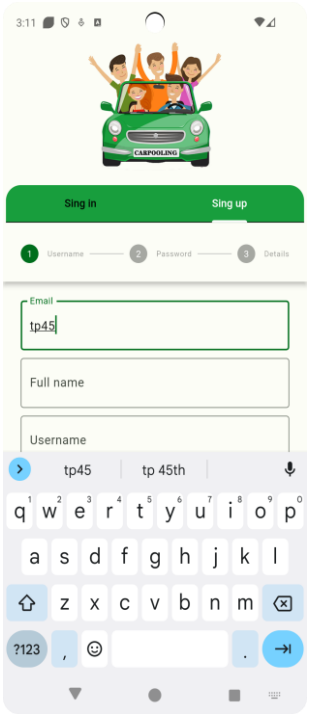


Figure 9: Sign up Page

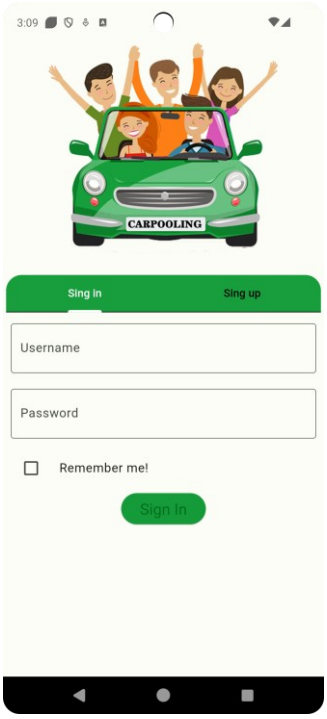


Figure 10: Sign in Page

Search: Users can search for available routes. The search (Figure 11) can be parameterized by starting route address and by destination route address. The information appears to the user as a list of routes and can be sorted by date. After clicking on one of those routes, the application shows more details regarding this route (Figure 13). In this page the user is given the options to view the driver's profile by pressing on the profile card and request to ride along by pressing the 'Ride along' button (Figure 14). Also, like the Home page (and any other page), the user can access the menu by clicking the 3 lines on the top left corner. From the drawer menu (Figure 12), the user can navigate to any page.

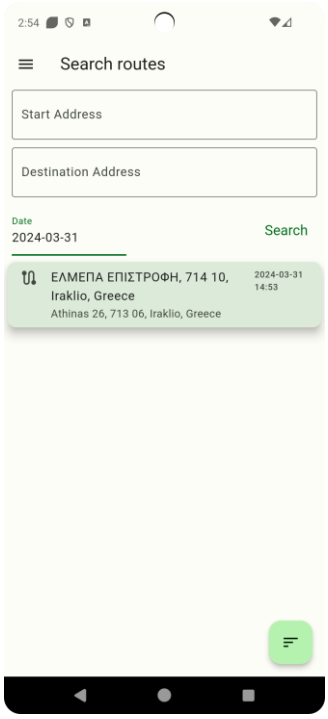


Figure 11: Search Page

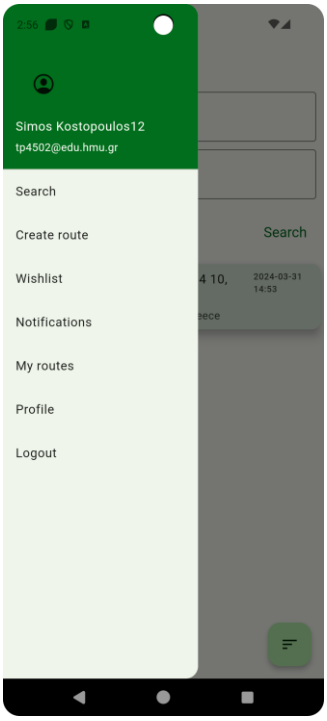


Figure 12: Menu

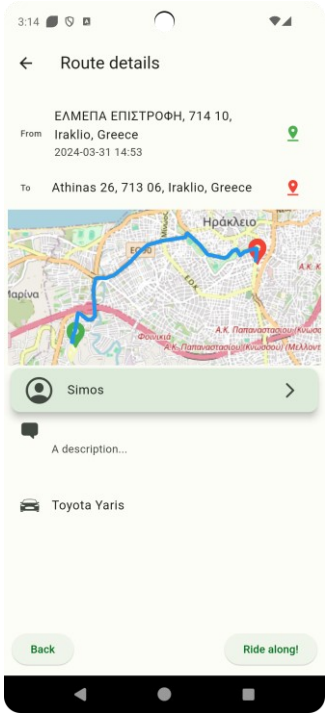


Figure 13: Searched Route Details

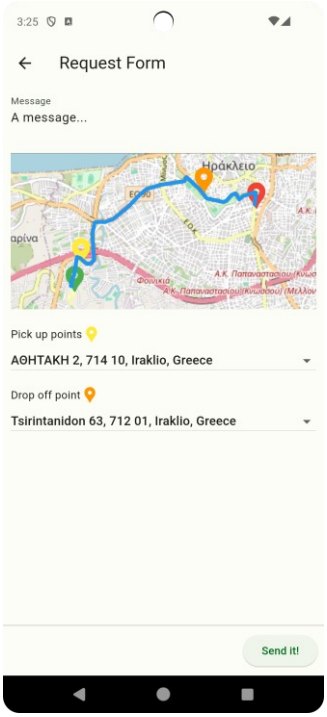


Figure 14: Ride Request Form

Profile: in Figure 15 the user can view his/her information. The user can register a car from the profile page and manage his/her already registered vehicle if any. Additionally, via this page, the user can check ratings and comments others user left in his/her profile (Figure 16).

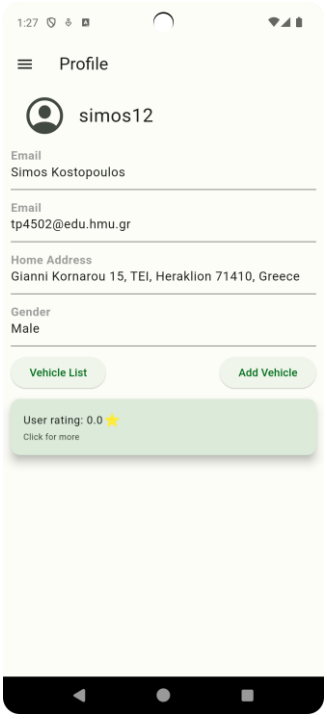




Figure 15: User's Profile Page ***Figure 16: User's Ratings Page***

Create Ride In Figure 17 user can create a ride by following the steps that are needed. The first and second steps are about the starting and destination points of the route. In the third and last step along with depart time, a user must also select one among their (registered) vehicle to use and declare the number of available seats, for this ride. Additionally, the user includes add any extra information she/he needs in the description text. Also, in case a user forgot to add a vehicle before, they can do it at the last step of the ride creation procedure.

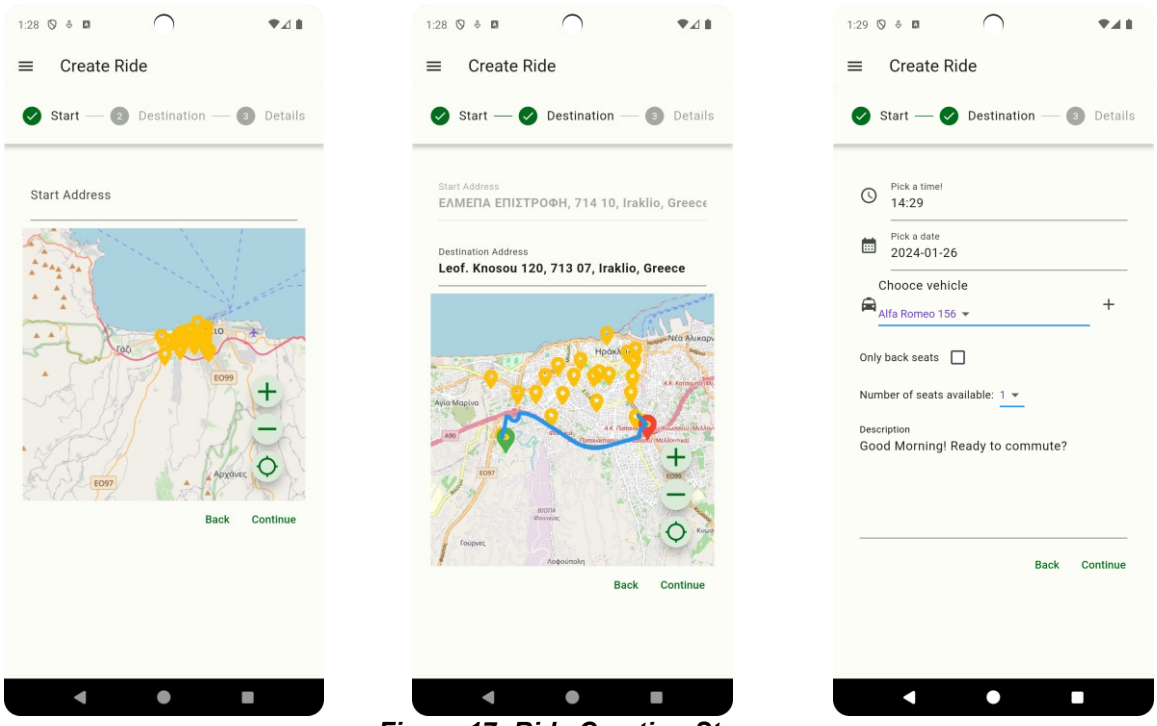


Figure 17: Ride Creation Steps

My Wishlist: the application provides the ability for users to create potentially interesting routes, which can be used as matchmaking mechanism in case they do not have a vehicle and cannot find a ride to fit their commute. To do so, as seen in Figure 18, a user must follow a three-step procedure, similar with the one found in route creation, with the difference being that during the last step -as it is shown in the right screenshot of Figure 18 - the user chooses the departure date and preferred time and the wish list route can be matched with a ride after it is created by a user. When a passenger registers a wish-list route, it does not appear in the search but can only be seen by its creator, as it is a route that a user wants to do but they want to do it as a passenger. Regardless of whether it is entered by users with or without a vehicle, should include basic information such as departure and destination point for each user and driver, as well as time and date. The matchmaking conditions are that Wishlist route, and the actual route matched to it must have a departure time difference up to one hour, be on the same date and the first must be a sub-part of the latter. Users can delete any of their Wishlist routes, simply by navigating to the Route Wishlist as can be seen in the left screenshot of Figure 18.

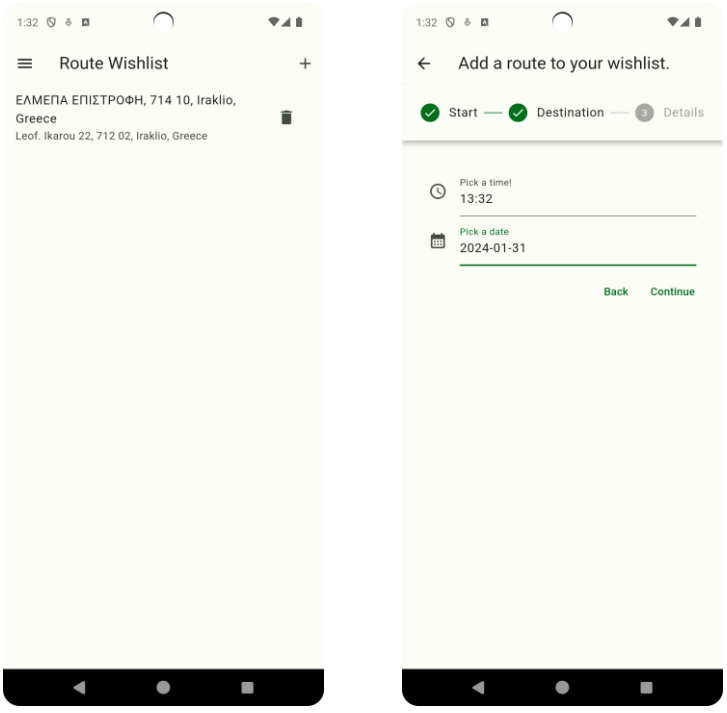
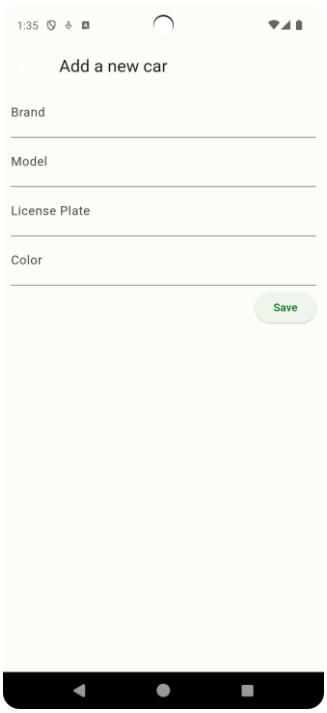


Figure 18: Wish list Route and Creation Page

Add Vehicles: it is possible to register one or more vehicles for carpooling. Details such as brand, model, year and license plate number are required as seen in

Figure **19.** This information helps to ensure the safety of all users and the passengers on a ride can recognize the car much easier. Each user can see their registered vehicles in a list, as shown in Figure 20. Both pages can be accessed via user profile's page (Figure 15).



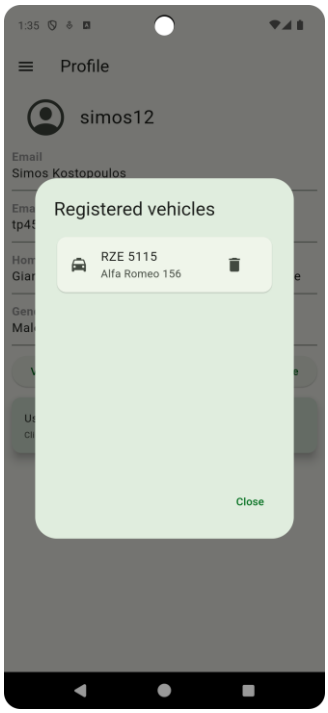


Figure 19: Add Vehicle Page ***Figure 20: Users Vehicle List***

My routes: consists of two tabs. In Figure 21, the page, which is shown by default, shows the routes created by the user-driver. This tab shows more information about the route that has been created and its completely organized to help the driver-creator edit, recreate, and delete the route.

From a passenger's perspective Figure 22, the page shows the history of routes that a user participates in, and for each one of them it contains a list of all the users that participated in a specific route. The route details page (Figure 23) is designed to be viewed by both the passenger(s) and the driver, showing this list of participants and some basic information for the route like start and destination. including a map with pick-up and drop-off points. Furthermore, in this page users can view each other's profiles and write comments (Figure 24) after the route is finished, to create a reliable network. Finally, the driver has some extra functionalities, such as initiating the route, which notifies the passengers and check the picked-up passengers.

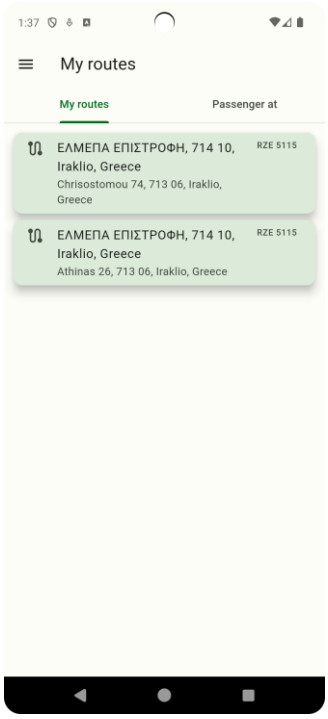


Figure 21: User's Route Page as Driver only.

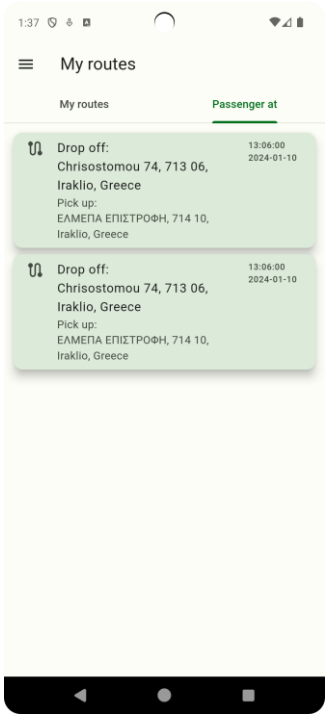


Figure 22: User's Routes Page as Participant.

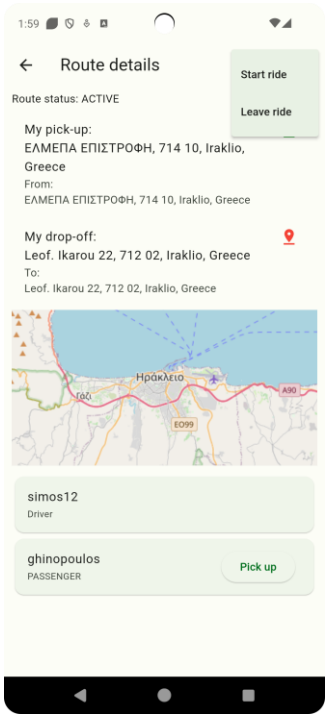


Figure 23: Route Details Page.

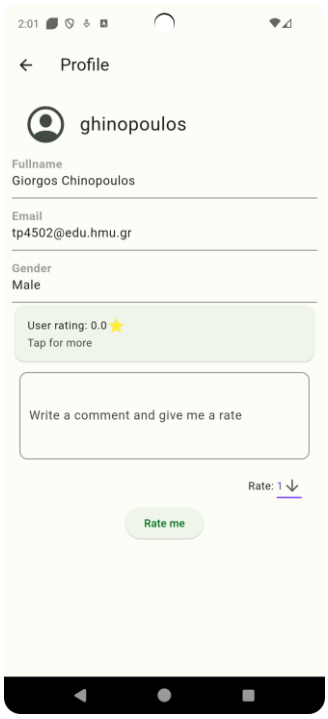


Figure 24: Participants profile Page.

Notification: as seen in Figure 25 users can check notifications and respond to requests, if necessary, as some of the notifications have the form of requests and are not only used to provide information (Figure 26)

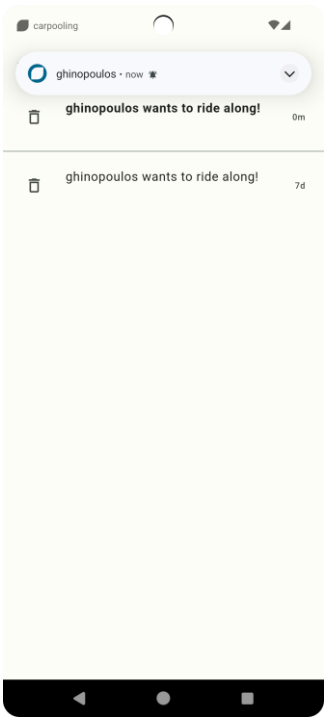


Figure 25: Notification Page

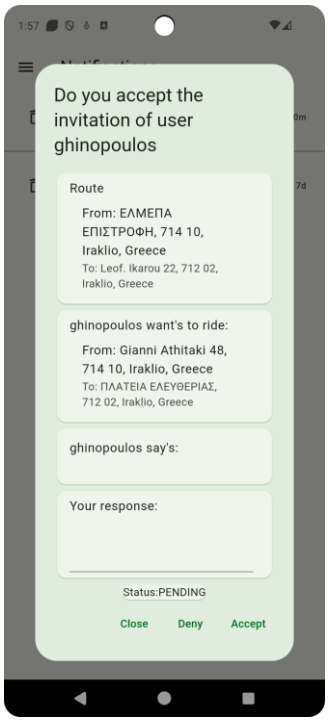


Figure 26: Notification Details

4.2 Server implementation

In this section, we are going to talk about the framework that we used to create our server, describe its architecture-structure and how it works. Spring Boot, as we already mentioned earlier, is the most popular Java framework for creating standalone, spring-based applications. The structure of the application consists of five layers.

The first layer of the application is the Security layer where it has a security filter chain that we can use to allow only specific requests to reach the controller. The most obvious reason for not allowing a request is for security reasons, like an unauthorized user that does not have the privileges or a user that is not logged in and try to access the REST API. Also, it handles light tasks like the login authentication of the users.

The Security layer is followed by the Controller layer. This is the layer that the client makes the requests to. Also, all the exposed endpoints are declared here and mapped to specific function. The Controller uses the Services layer that has all the business logic of the application. All the necessary processing of the data happens in Service layer. The Service layer needs data for all the processing. It accesses data via the Repository layer, which handles all the CRUD transactions, and it is responsible for transforming the data to objects from the Data layer. We will later examine how the data is transformed.

The Data layer is where all the data are stored with the help of a database. In our case, we use an SQL database and more specifically we use PostgreSQL where all our data are stored. Repository uses JPA that is a set of interfaces and follows the Object-Relation Mapping, ORM for short. JPA does all the mapping between the POJO classes and the database with the help of annotations, so its POJO class is a representation of a table in our database. Also, it handles the transactions between the objects and the DB as an entity manager.

To sum up, a client makes a request call at an endpoint which is initially filtered before reaching the controller. After filtering, the request is either rejected or proceeds to be processed. In the first case, the request is rejected with an error of an unauthorized user. In the second case, the server tries to fulfill the request by using a service to retrieve the necessary data from the database and process them if needed. Finally, the server constructs an answer in JSON format for the purpose of fulfilling the user's request. In the case that the request is a notification sent, it pushes the notification to the receiver before answering the client.

4.3 Application-to-Server Communication

The mobile application uses the available exposed http endpoints to send/receive, in JSON format, all the necessary information to/from the server. The only exception is that in this way the server can push a notification to device, especially when application is not running on foreground. For this purpose, we implemented and configured the Google Firebase cloud messaging service, also known as FCM. This service is widely used to push notification to applications, and it is the go-to solution for solving the problem of receiving notification when application is running on the background. So, in this scenario our server implements service to send the notification to the user using FCM as the middleman. This works after FCM assigns a token to each device which we keep in our database assigned to a username. The above architecture is necessary for the push notification feature to work normally and without consuming a lot of mobile resources by keeping an application running in the background all the time.

As we already mentioned, the server-side application exposes several endpoints that the clients can use to do CRUD based operations. These endpoints can be categorized by the entity that they do the CRUD operation on. The categories are user, vehicle, route, participation, wish list, and notification endpoints. First of we have the user data, which contain the user profile data (like email, username, password and so on). This category

includes necessary functionality for sign-in and sign-up, but it doesn't include a delete user endpoint. The Route endpoint apart for the classic CRUD operation allows to search in the dataset base on the generated tags on creation. This helps to improve the searching experience of the user. Also, on route creation after the route is saved the server tries to matchmake a wish list route with the route that is just created. If there is a match the server notifies the user/creator of the wish list route. Finally, vehicle, route participation and wish list are a classic CRUD set of endpoints, but the route participation can filter the result of the read endpoint based on route id, user id or both.

4.4 Problems and Solutions

In this section, we discuss the problems we came across during the development of the mobile application and the server. For the mobile application, we needed to solve numerous problems, such as keeping data transfers with the server to the bare minimum, not requesting data that remain unchanged over time, and the one of reliable geocoding. At the same time, in the server-side development, the main problem was to have a flexible data structure to support different types of notifications that are being created in the app.

The first problem that we encountered was to keep the mandatory transfer of data but to maintain for the proper operation of the application for each user as some data need to be altered in real time. This procedure results in an increased cost of resources for data transfer, which may change between user visits to the application page. For this reason, the use of some key-value storage was initially chosen, which has the advantage of being able to be updated whenever needed, while at the same time storing some necessary static data, such as for example the information of each user. Thus, every time a user visits his profile page, he does not need to receive all his profile data, but only the ratings data. Finally, this feature also offers the bonus of shorter loading time since the data is now stored locally for each user. Internet data transfer can be further be reduced by using caching for map loading instead of relying completely on the API database.

A second problem that we encountered which was not apparent from the start, was the task to find an open-source API for geocoding that works reliably, meaning that it provides a consistent road naming format for every road. The process of turning coordinates to addresses and back is called backward and forward geocoding. The most popular geocoding API is the Nominatim [24] that provides more than enough functionality for what we need. After conducting some testing with random clicking and checking whether the addresses come back correctly, we quickly realized that for Greece, and especially for cities the size of

Heraklion, there was a lack of consistency in addresses provided. To be more specific, some roads are written in the API's database with Greek letters and some with English letters, creating a confusion for developers and users that are trying to search routes that fit their commuting needs.

Furthermore, a third problem that appeared was in reverse geocoding. Reverse geocoding is the process from which the coordinates are translated to an address. The problem being the produced address was not representative for the specific spot, but rather the whole street, due to the lack of data that Nominatim API provides. Also, it creates the same problem for forward geocoding, which makes search results very vague and not corresponding to the correct coordinates, resulting in false points shown on the map. Some other options that we had required API keys every few days for the free version, making them hard to use without paying or lacked some of the desirable features.

Considering the above, we concluded that the APIs we found will not provide us with sufficient address data for our needs or will not have some features, without opting for non-open-source solutions. So, the available solution at the time was to use two different APIs combined. The base one has a very good street naming format, that we can rely on to save in the database and use it to search in our database and produce reliable results as users searched for participation in a route. The other one serves the purpose of autocomplete, since the first one does not support query search in the API database. The combination of both is used to provide user suggestions for address in route creation process. Not the most elegant solution to the problem, but a very good one indeed, for the purpose of avoiding payments and API keys expiration management. For a production grade application, there would be the need for greater accuracy, the best solution will probably be Google's API, as it is very reliable, and it has been improving for years.

5. Testing

For every application that is being developed, a big part of the development process is devoted to the testing procedure. This is required since adequate testing can reduce (and ideally eliminate) the possibility of running breakdowns. In this chapter, we describe how we tested the correctness of our application. Our testing environment consists of one server (running the server-side part of our application) and two application users (using the mobile part of our application on their cell phones).

5.1 Server-side Application

We start by testing the functionality of our application's server-side, independently from the mobile application. To do so, we used the Postman [25], which is an external HTTP client, and test each REST API endpoints that our server exposes. Specifically, using Postman we send specific http requests on each endpoint Hand check that the server's answers match the expected responses, to ensure that it works as expected. Additionally, when applicable, we checked both the result of each request on the state of the application and the integrity of the data stored in our database, by directly accessing the database using PgAdmin a tool that is included in the installation of Postgres or any other mean that connects to the database and can make a query.

The testing of an endpoint in general consists of a request to the specific endpoint with a given data or parameters input. The response of the request should have a 200-status code with some data in the body, if any data are expected. For example, an endpoint that saves the user data in the database takes a specific dataset and after saving everything it will return a 200-status code along with the data with just saved. Some endpoints require a parameter instead of data included in body, but the data response may be empty if nothing matches the parameter. For example, if the parameter is a user id and that user with the specific id does not exist the result will be empty.

5.2 Mobile Application

5.2.1 Mobile Application Testing

The application testing took place using predefined scenarios. A scenario contains a series of steps that a user follows for any process inside the application. We tested the following

routines of the application: i) the ride creation, ii) search and send a ride request, iii) check notification, and answer requests if any, iv) edit or reschedule of existing rides, v) start a ride, vi) notify passengers, vii) review the passengers and the driver of a ride, viii) check profile and profile reviews, and ix) add vehicle to profile.

All the above routines were manually tested with a variety of scenarios to demonstrate a more realistic use, i.e. simulate application usage by various users. To do this, during the testing process, we included (on purpose) “mistakes” and interactions that users can make and could lead to an unexpected result or even result to improper operation of the application and errors.

All the scenarios are tested with all the components (mobile application and server-side application) being in a local network, therefore the network delay is kept to a minimum. Also, the mobile application test scenarios are to ensure that all the information is shown correctly in the screen of the phone, since we already tested the integrity of the server in the previous section.

5.2.2 Scenarios

In this section, we describe the set of scenarios we created to test that the mobile application is working and verify that it behaves as expected. The successful outcome (i.e. getting the desired and correct results) of each of the testing scenarios described below, ensures that we have a fully working application without bugs.

Login/Sign-up

In this scenario, we test the creation of a user profile and the user login. A successful outcome includes the following: all the data provided by the user is uploaded, received, and saved successfully from the server into the database. Also, after the Sign-Up process, the user must be automatically logged-in in the application and transferred to the home screen/search page, and so it happens.

Create Route

Secondly, we must check the create route process. For this scenario, it is necessary for the user to have an already registered vehicle. The route creation is a three-step process, and each step should be completed before moving to the next one. In the first two steps, we check the workings of the map as it should be interactive and clickable. After clicking anywhere on the map, a marker should appear, and the text field above should be filled automatically with the address that is picked. The opposite should work as well - the user can write a part of the address and a suggestion box should appear below. After clicking on the

address, the text field should be now filled with the whole address, and a marker should appear on the map. In the third and last step of this process, the user should be able to choose time and date for the ride. To do so, the user should be able to click on “Pick a time!” and “Pick a date!” fields, and a dialog should appear for each, allowing the user to choose time and date and after pressing “ok”, the information should automatically fill in the text fields respectively. After finishing every step, the application should take us to the “My Routes” page. We should be able on this page to view the route that we just created. If the route does not appear in the list in “My Routes” page we should first reload the page by going to the menu and clicking on the page, before failing the test.

Home/Search

Here we test whether the application can “search” for a route, open the route specifics and request a ride with pre-defined pick up and drop of points. It is necessary to initially create several routes, which preferably should be created by another user, so the search will not be empty. To begin with, the route searching is based on three criteria start, finish and date, which date is the only one that is mandatory. The start and finish fields are supposed to suggest autocompleting option upon writing an address, and the suggestions come from the database. After clicking search for the parameters provided, all the routes must be on the selected day. If one provided start and finish addresses the result should have not only a start and finish point which exactly match with the parameters but should rather show only routes that contain these two addresses. The search result can be sorted by ascending or descending date-time.

After clicking on any of the results, we should see all the information of the route that the previous page provided and check the user profile with all the comments and ratings. In that way, we ensure all the information is shown correctly on the screen. Finally, we can request to ride along, and at this point, we should be able to choose any of the pick-up and drop-off point that are available, but the start should not be after the finish the application should enforce that by setting the drop-off to default value “Select a point”. Finally, we hit “Send it!” and the driver of the route should receive a pending request as a notification. This notification can be viewed on the notification page, as described in the next scenario.

Notification

In this page, from the driver’s perspective, we should see a notification in bold, as described in the previous test. When the user opens the notification, the bold should go away as an indication that it is now “seen,” and a dialog should appear that provides all the information of the notification. Also, the user should be able to answer the request (a ride request in this scenario) by accepting it or denying it (the message is optional). The user who requested it,

should receive a notification on their phone accordingly. If the driver accepts the request the passenger should be added as a participant to the ride. Also, this process is followed for any request that needs approval. Finally, all notifications should follow this type with different action buttons and information.

My Routes

This page consists of two tabs “My Routes” and “Passenger at”. In the first tab, there are all the routes that a user has created and can edit, replan, or delete. Upon editing a route, the user must be able to edit only some specifications, like time or vehicle, and if the editing process is cancelled, the route should return to its original state and only be updated if the edit is saved. Any other response for the application is not correct. After saving up the edit, it should update all passengers that the route has been edited. Upon replan, the user can recreate a route with the same start and finish points. After pressing “save” in My Routes page left tab there should be a new route with all the updated information. The action of deleting a route should remove it from the “My Routes” tab, the route should not be available for searching and all the participating passengers should be notified that this route no longer exists. However, the information of the route should still appear in the right tab.

In the right tab, all users can see the routes that they are participating in, either as drivers or passengers. After clicking on each of the routes, more details of the routes should appear. As a driver, you can start a route, which notifies everyone that the route is about to start. At this point we need to check on the passenger phone if they have a new notification. The passengers should be able to answer the notification as ready. If the user does not answer, we assume that he/she is not ready. Depending on whether the passenger answered the notification the pick-up button should be clickable or not. On click, it sends a confirmation notification to the passenger that their driver picked them up, which appears on the notification page.

Also, passengers should be able to exit the route and not make the list anymore. Furthermore, every user can check and rate other users of the route, while they can view their existing ratings and profile information at the same time.

Finally, for every notification that is sent to a user in this process, we should rerun the test for Notification and make sure that the new notifications are shown correctly and can be answered.

Wishlist

In this page, we should test the creation of a route by clicking the “+” button, which appear at a later point on the list of the page. We should also be able to delete this route. If we already

have a wish list route, we should get a relative notification when a new route which matches with ours is created. This process happens successfully in the background on the server side.

Profile

In the profile page, we should be able to see our profile information, the vehicle list, and our ratings. Adding a vehicle was already tested earlier at the “Create Route”, so it should be working properly as well, since it is the same page.

After completing the design and creation of all the pages, we ran all the tests to check if they work as described above in the scenarios. During the testing process, we followed all the steps one by one for each page and concluded that all features work as they should be, with no exceptions.

5.2.3 Flutter Testing Environment

Flutter has a particularly good testing environment, but it has several limitations. It allows testing using automated scenarios, which are built once and then can be re-run when needed. This includes both the tests for the UI and the integration tests that are used for checking the functionality that runs in the background. However, it restricts HTTP requests. Although Flutter offers a workaround on that restriction, its use is not really encouraged by the framework. As a result, we tested the above scenarios manually. The manual application testing was chosen over the integrated and auto testing that the Flutter provides for a main reason: most of them require data that are retrieved from the server via HTTP request while every request that occurs returns a 400-status code, which means that the request never reached its destination.

6. Conclusions & Future Work

6.1 Conclusions

The goal of this thesis was the design and implementation of a fully functional mobile application, providing ride-sharing services among a university community.

The application includes all necessary features to be a fully functional pilot application and to fulfill its purpose as a ride-sharing app, among Hellenic Mediterranean University students, operating in scale for the city of Heraklion. Meanwhile, the design of the user interface is user-friendly and easy to navigate, as intended, with the help of Flutter's clean design. Finally, after completing all the necessary tests, every step included in the processes of the application works as it should be, without any bugs. The features described in Section 4.1 can be summed up in the following: restrict access to email addresses only from Hellenic Mediterranean University domain, integrated map to visualize the route and the points of interest, embedded features for easier and safer communication, rating and commenting system, two-step verification from participants of the route for entering the vehicle, and route wish list for upcoming routes with matchmaking notification.

Completing this thesis, we are confident that this is an important first step for helping university students in Greece foster a more sustainable mindset regarding vehicle carbon footprint and total emissions, while at the same time commuting to and from university becomes easier.

6.2 Future Work

The ride-sharing application developed during this thesis can be extended in several directions. In the following, we briefly describe such extensions for enhancement across various domains, notably focusing on augmentations to the user experience. Each of these extensions presents an opportunity to enrich the user experience and foster greater engagement and satisfaction among users.

University OAuth. The most future crucial update of the application is to include the university OAuth into the login process for university students. As it will further improve the security of the application comparing to just check the email domain (HMU academic email).

iOS Implementation For the time being the application is only tested for android devices. Flutter as UI framework supports compiling the application directly to iOS, but the only limitation is the platform specific code from plugins that will not run on the iOS platform. As a

future implementation, the application should be formatted to run in iOS supporting devices as well.

Pick-up points. For a passenger who is about to request to ride along, the application currently supports only the ability to select pick-up points using a pre-defined set of static points in the map. The next version of the application could implement dynamic pick-up point selection, allowing passengers to choose their pick-up location directly on the map, offering flexibility and convenience. Additionally, integrating geolocation services could enable the app to suggest nearby pick-up points based on the passenger's current location.

In-app navigation. For the driver, it would be very helpful to have in-app navigation, so that they avoid switching applications back and forth (e.g. between google maps and our in-application map). Also, it would be helpful to show both the estimated time of arrival to the destination, and the pick-up and drop-off points of the passengers.

Estimation of arrival time. Additionally, it will be very helpful for the passenger to know when their driver is getting close, and how much time is required until the driver arrives to the pick-up point, while also notifies the user for estimated for specific thresholds for an example 3 min before, and on arrival. Also, when a deviation is needed for picking up a second passenger, the app could calculate the extra time As a result of the above, the ride request procedure is more well-constructed and robust that improves user experience.

Behavior monitoring system. A monitoring system can be implemented to help in cases of unwanted behavior from users. This will be based on reports and comments and will be programmed to either give users a warning, temporally close their account, or ban them from the application permanently.

User expansion outside the University community. Regarding a future launch of the application outside the university community, to maintain the same levels of security, new users' credentials and identification need to be verified, using their ID or driver's license. In this case, the photos uploaded by the user should not remain on the servers after identification and will only be used for identification purposes.

7. References

[1] L. K. A. &. A. G. Mitropoulos, "A systematic literature review of ride-sharing platforms, user factors and barriers.," *European Transport Research Review ,* 2021.

[2] C. D. Parker, "Carpooling and Mental Health," 11 6 2019. [Online]. Available: https://medium.com/tripbuddy/carpooling-and-mental-health-19d7d2b35e97.

[3] Uber, "UberPool," [Online]. Available: https://www.uber.com/au/en/ride/uberpool/.

[4] Uber, "UberX," [Online]. Available: https://www.uber.com/us/en/ride/uberx/.

[5] Hasa, "Difference Between UberPool and UberX," 31 10 2017. [Online]. Available: https://www.differencebetween.com/difference-between-uberpool-and-vs-uberx/

[6] "Uber Safety," [Online]. Available: https://www.uber.com/gr/en/ride/safety/?

[7] BlaBlaCar, "About us," [Online]. Available: https://blog.blablacar.com/about-us.

[8] L. Fang, "Pros and Cons of BlaBlaCar (French Rideshare App)," 07 07 2019. [Online]. Available: https://imperfectidealist.com/pros-and-cons-of-blablacar/

[9] BestReferral Team, "The Ultimate Guide to Blablacar: The Carpool App," BestReferral Team, 03 03 2023. [Online]. Available: https://bestreferraldriver.com/blablacar.html?

[10] BlaBlaCar, "BlaBlaCar Ladies Only," [Online]. Available: https://support.blablacar.com/hc/en-gb/articles/360015527240-Ladies-only.

[11] H. Nguyen, "Top 10 best Cross Platform app development frameworks," TECHVIFY SOFTWARE, 21 07 2023. [Online]. Available: https://techvify-software.com/cross-platform-app-development-frameworks/.

[12] M. Kapias, "Flutter vs. React Native: Which one is better for 2023?," 2020. Updated 2023. [Online]. Available: https://fireart.studio/blog/flutter-vs-react-native-what-app-developers-should-know-about-cross-platform-mobile-development/.

[13] C. Dishpende, "Flutter vs. React Native: Which One to Choose in 2023?," 18 08 2023. [Online]. Available: https://www.simplilearn.com/tutorials/reactjs-tutorial/flutter-vs-react-native .

[14] Google, "Flutter architectural overview," [Online]. Available: https://docs.flutter.dev/resources/architectural-overview.

[15] OS system, "Best Backend Languages for Android App," 21 4 2021. [Online]. Available: https://os-system.com/blog/best-backend-languages-for-android-app/.

[16] j. paul, "10 Best Backend Frameworks for Web Development," 9 3 2022. [Online]. Available: https://medium.com/javarevisited/10-best-backend-frameworks-for-web-development-8d19e337f774.

[17] V. Raj, "The Pros and Cons of Node.js Web App Development: A Detailed Look," 28 02 2022. [Online]. Available: https://javascript.plainenglish.io/the-pros-and-cons-of-node-js-web-app-development-a-detailed-look-c91a22f013c.

[18] bhagavati, "The Pros and Cons of Node.JS in Web Development," 08 12 2022. [Online]. Available: https://www.geeksforgeeks.org/the-pros-and-cons-of-node-js-in-web-development/.

[19] R. Mishal, "6 Advantages and Disadvantages of Spring boot | Limitations & Benefits of Spring boot," 16 08 2022. [Online]. Available: https://www.hitechwhizz.com/2022/08/6-advantages-and-disadvantages-limitations-benefits-of-spring-boot.html .

[20] V. Puzhevich, "Pros and Cons of Using Spring Boot," 3 2 2023. [Online]. Available: https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/.

[21] SCAND, "Benefits and Drawbacks of Using Spring Boot," 13 05 2021. [Online]. Available: https://dev.to/scand/benefits-and-drawbacks-of-using-spring-boot-3kk7.

[22] D. Matthew, "What Is Flask and How Do Developers Use It? A Quick Guide," 30 08 2023. [Online]. Available: https://careerfoundry.com/en/blog/web-development/what-is-flask/

[23] J. Au-Yeung, "Best practices for REST API design," 20 03 2020. [Online]. Available: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/#h-accept-and-respond-with-json.

[24] "Nominatim API - Overview," [Online]. Available: https://nominatim.org/release-docs/develop/api/Overview/.

[25] POSTMAN, [Online]. Available: https://www.postman.com/.

[26] S. Osadchuk, "Flutter vs. Kotlin in 2023: Which Is Best for Cross-Platform App Development?," 26 3 2023. [Online]. Available: https://doit.software/blog/flutter-vs-kotlin-which-is-best-for-cross-platform-app-development#screen1.

[27] G. Samojło, "Pros and Cons of Kotlin for Android App Development," 3 8 2023. [Online]. Available: https://www.netguru.com/blog/kotlin-pros-and-cons.

[28] K. Michal, "React Native vs Flutter: Which One is Better for 2023?," 13 11 2023. [Online]. Available: https://fireart.studio/blog/top-most-popular-programming-languages-for-mobile-app-development/?