

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή εργασία

Η χρήση NoSQL λύσεων σε σύγχρονες εφαρμογές: Μελέτη Περίπτωσης Neo4j

Χατζηιωάννου Παναγιώτης-Ιωσήφ
ΤΠ3584

Abstract

The present thesis is concerned with the use of NoSQL systems for the development of data-intensive social applications. Specifically, the thesis elaborates on the Neo4j graph data storage, its underlying data model and its querying language. The report begins with a thorough analysis of the different NoSQL technologies and their data models focusing on graph databases and the property graph data model. Basic examples were demonstrated to understand its core concepts.

The specific goal of the work presented here within is to study how a graph database can be used to store and manage data that reflect real life use cases. This is done by referring to a prototypical implementation of an application broadly related to the domains of tourism, culture and / or agri-food. For purposes of illustration we rely on small hypothetical datasets which however suffice to showcase both the type of problems arising as well as the solutions favored by Neo4j and the advanced path querying mechanisms intended to support some of the complex data retrieval scenarios.

The fact that we used small datasets on a relatively small graph allows for further improvements in the practical as well as the theoretical aspects of the study. Multiple questions are of interest in cases where the volume of data is significant. Future work can contemplate on how a graph can be structured to efficiently support large volumes of data. Can a graph be partitioned to work in a distributed environment and how would that affect the path queries?

The development of this project exposed me to a plethora of different state of the art database technologies that solve complex data storage problems as well as software development libraries and frameworks. New problem-solving skills were acquired in both designing and developing modern scalable web applications.

Σύνοψη

Η παρούσα πτυχιακή εργασία μελετά τη χρήση NoSQL συστημάτων διαχείρισης και οργάνωσης κοινωνικών δεδομένων. Συγκεκριμένα η εργασία μελετά το σύστημα διαχείρισης δεδομένων γράφων Neo4j, το θεμελιώδης μοντέλο δεδομένων του, και την γλώσσα υποβολής ερωτημάτων την οποία υποστηρίζει. Η αναφορά αρχίζει με μια πλήρης ανάλυση των διαφορετικών NoSQL μοντέλων δεδομένων που έχουν αναπτυχθεί τα τελευταία χρόνια επικεντρώνοντας στις βάσεις δεδομένων γράφων και το μοντέλο property graph. Για την αποτελεσματική κατανόηση των βασικών εννοιών του property graph, καταδείχθηκαν απλά παραδείγματα σεναρίων ανάκτησης δεδομένων.

Στόχος της έρευνας είναι να αναδείξει το πώς μπορεί να χρησιμοποιηθεί μια βάση δεδομένων γράφων για την επίλυση σύνθετων προβλημάτων διαχείρισης δεδομένων τα οποία αντικατοπτρίζουν αληθινές περιπτώσεις χρήσης. Αυτό επιτυγχάνεται με την ανάπτυξη μίας πρωτότυπης εφαρμογής που σχετίζεται με κλάδους ιδιαίτερου ενδιαφέροντος όπως είναι ο τουρισμός, ο πολιτισμός ή/και αγροδιατροφικός τομέας. Για λόγους επίδειξης, η εφαρμογή βασίζεται σε μικρά σύνολα υποθετικών δεδομένων τα οποία ωστόσο επαρκούν για την παρουσίαση τόσο του είδους των προβλημάτων των οποίων προκύπτουν, όσο και των λύσεων που ευνοούνται από τη Neo4j και τους προηγμένους μηχανισμούς ερωτημάτων τροχιάς οι οποίοι προορίζονται να υποστηρίξουν ορισμένα από τα σύνθετα σενάρια ανάκτησης δεδομένων.

Το γεγονός ότι χρησιμοποιήθηκαν μικρά σύνολα δεδομένων δημιουργώντας έναν σχετικά μικρό γράφο, επιτρέπει περαιτέρω βελτιώσεις τόσο στις πρακτικές όσο και στις θεωρητικές πτυχές της μελέτης. Πολλά ενδιαφέροντα ερωτήματα προκύπτουν όταν ο όγκος των δεδομένων είναι αρκετά μεγάλος. Μελλοντικές εργασίες θα μπορούσαν να εξετάσουν τον τρόπο με τον οποίο ένας γράφος πρέπει να δομηθεί για την αποτελεσματική υποστήριξη μεγάλων όγκων δεδομένων. Είναι εφικτός ο διαχωρισμός ενός γράφου έτσι ώστε να επιτευχθεί η αποτελεσματική λειτουργία του σε ένα κατανεμημένο περιβάλλον και πως αυτό θα επηρέαζε τα ερωτήματα τροχιάς.

Η θεωρητική μελέτη της εργασίας και η πρακτική ανάπτυξη της εφαρμογής έφερε σε επαφή των φοιτητή με μια πληθώρα διαφορετικών τεχνολογιών βάσεων δεδομένων της τρέχουσας τεχνολογικής τροχιάς που επιλύουν σύνθετα προβλήματα διαχείρισης δεδομένων, αλλά και με ένα σύνολο βιβλιοθηκών και frameworks που αφορούν την ανάπτυξη λογισμικού. Αποκτήθηκαν νέες δεξιότητες όσον αφορά την επίλυση προβλημάτων τόσο στον σχεδιασμό όσο και στην ανάπτυξη σύγχρονων διαδικτυακών εφαρμογών.

Πίνακας περιεχομένων

1 Εισαγωγή 9
1.1 Το Σχεσιακό Μοντέλο και οι αδυναμίες του 9
1.2 Η ανάγκη για νέα μοντέλα δεδομένων 10
1.3 Στόχοι πτυχιακής εργασίας 11
1.4 Δομή πτυχιακής εργασίας 12
2 Θεωρητικό υπόβαθρο 13
2.1 ACID συναλλαγές και θεώρημα CAP 13
2.3 Πότε είναι χρήσιμα 15
2.4 Προβεβλημένα μοντέλα δεδομένων 16
2.4.1 Key-Value Stores 16
2.4.2 Document Stores 16
2.4.3 Column Stores 17
2.4.4 Graph Databases 17
2.5 Συστήματα NoSQL Βάσεων Δεδομένων 17
2.5.1 MongoDB 17
2.5.2 Cassandra 18
3 Βάσεις Δεδομένων Γράφων και Neo4j 20
3.1 Βάσεις δεδομένων γράφων και συνδεδεμένα δεδομένα 20
3.1.1 Property Graph 21
3.1.2 Γλώσσες υποβολής ερωτημάτων γράφων 22
3.2 Neo4j 24
3.2.1 Cypher 25
4 Μελέτη περίπτωσης 26
4.1 Ανάλυση Προβλήματος 26
4.2 Τεχνολογικά εργαλεία 26
4.2.1 Node.js 26
4.2.2 Vue.js 27
4.2.3 NPM 27
4.2.4 APIs 28
4.2.5 Git/Github 28
4.2.6 Πακέτα και plugins - Front-End 28
4.2.7 Πακέτα και plugins - Backend 29
4.3 Σχεδιασμός υλοποίησης 29
4.4 Το property graph με τα ενδεικτικά δεδομένα 30
4.3.1 User domain model 30

4.3.2 Posts domain model 31
4.3.3 Tags domain model 33
4.3.4 Places domain model 34
4.3.5 Notifications domain model 35
4.3.6 Προτάσεις φιλίας βάσει κοινών γνωστών 36
4.3.7 Χρονολόγιο χρήστη 37
4.3.8 Προφίλ Χρήστη 37
4.4 Υλοποίηση back-end 38
4.4.1 Επικοινωνία με την βάση δεδομένων 38
4.4.2 Models 39
4.4.3 Controllers 40
4.4.4 Routes 41
4.5 Υλοποίηση front-end 41
4.5.1 Χρησιμοποιώντας το back-end API 42
4.5.2 Ροή δεδομένων 43
4.6 Πιλοτική επίδειξη εφαρμογής 46
4.6.1 Χρονολόγιο προφίλ χρήστη 46
4.6.2 Αρχική σελίδα εφαρμογής 47
4.6.3 Τοποθεσίες 49
5 Επίλογος 51
5.1 Γνώσεις που αποκτήθηκαν 51
5.2 Μελλοντική Εργασία και Επεκτάσεις 51

Λίστα Εικόνων

Εικόνα 1: Scale-up vs. Scale-out 11
Εικόνα 2: ACID μοντέλο συναλλαγών 13
Εικόνα 3: Θεώρημα CAP 14
Εικόνα 4: MongoDB Data Model 18
Εικόνα 5: Cassandra Data Model 19
Εικόνα 6: Το πρόβλημα της αναζήτησης μηνυμάτων 19
Εικόνα 7: Σχέσεις σε NoSQL βάσεις εγγράφων 20
Εικόνα 8: Labeled Property Graph 22
Εικόνα 9: Ένα απλό μοτίβο γράφου 23
Εικόνα 10: Το πρόβλημα "friend-of-a-friend" 24
Εικόνα 11: VueJS components 27
Εικόνα 12 : Μοντέλο δεδομένων 30
Εικόνα 13: Users domain model 31
Εικόνα 14: Δημιουργία χρήστη 31
Εικόνα 15: Δημιουργία σχέσης :FOLLOWS μεταξύ δύο χρηστών 31
Εικόνα 16: Posts domain model 32
Εικόνα 17: Δημιουργία νέου post 32
Εικόνα 18: Δημιουργία μοναδικού αναγνωριστικού 32
Εικόνα 19: Δημιουργία post με μοναδικό αναγνωριστικό 32
Εικόνα 20: Δημιουργία σχολίου με μοναδικό αναγνωριστικό 33
Εικόνα 21: Tags domain model 33
Εικόνα 22: Δημιουργία post με tags 34
Εικόνα 23: Places domain model 34
Εικόνα 24: Δημιουργία σχέσης :VISITED 34
Εικόνα 25: Δημιουργία σχέσης :RATED 35
Εικόνα 26: Δημιουργία ενδιαφέροντος 35
Εικόνα 27: Notifications domain model 35
Εικόνα 28: Αναζήτηση φίλων που έχουν επισκεφθεί ένα μέρος 36
Εικόνα 29: Δημιουργία notification 36
Εικόνα 30: Σύστημα προτάσεων φιλίας βάσει κοινών γνωστών 36
Εικόνα 31: Δημιουργία χρονολογίου 37
Εικόνα 32: Άτομα που ακολουθούν έναν χρήστη 37
Εικόνα 33: Μέρη που έχει επισκεφθεί ο χρήστης 38
Εικόνα 34: Αρχιτεκτονική server 38
Εικόνα 35: Σύνδεση με την βάση δεδομένων 39
Εικόνα 36: Παράδειγμα μοντέλου Tag 40
Εικόνα 37: Παράδειγμα controller TagController 40
Εικόνα 38: Παράδειγμα route 41
Εικόνα 39: Παράδειγμα single-file component 42
Εικόνα 40: Ιεραρχία Vue components (Μέρος 1) 42
Εικόνα 41: Ιεραρχία Vue components (Μέρος 2) 42
Εικόνα 42: Επικοινωνία client-server μέσω του Axios 43
Εικόνα 43: Παράδειγμα `TagService.js` 43
Εικόνα 44: Παράδειγμα ExtendedPost 44
Εικόνα 45: Παράδειγμα ροής δεδομένων μέσω props 44
Εικόνα 46: Παράδειγμα ροής δεδομένων μέσω events 45
Εικόνα 47: Οπτική αναπαράσταση ροής δεδομένων 45
Εικόνα 48: Χρονολόγιο προφίλ χρήστη - Ερώτημα 46

Εικόνα 49: Χρονολόγιο προφίλ χρήστη – Αποτέλεσμα.....46
Εικόνα 50: Προσωπικό προφίλ χρήστη.....47
Εικόνα 51: Αρχική σελίδα εφαρμογής.....47
Εικόνα 52: Βασικές πληροφορίες χρήστη - Ερώτημα.....48
Εικόνα 53: Βασικές πληροφορίες χρήστη - Αποτέλεσμα.....48
Εικόνα 54: Τα πιο δημοφιλή tags - Ερώτημα.....48
Εικόνα 55: Τα πιο δημοφιλή tags - Αποτέλεσμα.....48
Εικόνα 56: Σελίδα τοποθεσίας.....49
Εικόνα 57: Φίλοι που έχουν επισκεφτεί ένα μέρος.....50
Εικόνα 58: Στατιστικά τοποθεσίας - Ερώτημα.....50
Εικόνα 59: Στατιστικά τοποθεσίας - Αποτέλεσμα.....51

1 Εισαγωγή

Το σχεσιακό μοντέλο και οι σχεσιακές βάσεις δεδομένων χαρακτηρίζονται από μακρόχρονη παρουσία στον κλάδο της Πληροφορικής, ενώ κατέχουν και μία περίοπτη θέση στα πληροφορικά συστήματα και τις εφαρμογές επιχειρήσεων και οργανισμών με αποκλειστική ευθύνη την αποθήκευση δεδομένων. Με την ραγδαία εξάπλωση του διαδικτύου και την έλευση της big data εποχής τα δεδομένα που δημιουργούνται καθημερινά αυξάνονται εκθετικά, με αποτέλεσμα να προκύπτουν νέες προκλήσεις τόσο σε σχέση με την αποθήκευσή τους όσο κυρίως σε σχέση με την διαχείριση τους για την εξαγωγή χρήσιμης πληροφορίας. Οι προκλήσεις αυτές εδράζονται στο γεγονός ότι το επί σειρά ετών προβεβλημένο μοντέλο σχεσιακής διαχείρισης δεδομένων αδυνατεί να ανταποκριθεί στις απαιτήσεις που προβάλουν οι σύγχρονοι σύνθετοι τύποι δεδομένων αλλά η αποτελεσματική τους επεξεργασία. Ειδικότερα, ολοένα και περισσότερο οι βάσεις δεδομένων που σχεδιάζονται για να καλύψουν τις σημερινές ανάγκες απαιτείται να είναι όχι μόνο αρκετά ευέλικτες έτσι ώστε να χειρίζονται διαφορετικές μορφές δεδομένων, αλλά επίσης να έχουν εξαιρετικές επιδόσεις και να είναι επεκτάσιμες έτσι ώστε να μπορούν να διαχειριστούν τεράστιους όγκους δεδομένων.

1.1 Το Σχεσιακό Μοντέλο και οι αδυναμίες του

Το σχεσιακό μοντέλο προτάθηκε σε μία δημοσίευση του Edgar Codd το 1970 [1] ως ένας μηχανισμός διαχείρισης και αποθήκευσης δεδομένων ο οποίος θα έλυνε τα προβλήματα που παρουσίαζαν τα μέχρι τότε γνωστά μοντέλα δεδομένων (ιεραρχικό μοντέλο και μοντέλο δικτύου). Τα λογισμικά που ακολουθούν το μοντέλο αυτό ονομάζονται Σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων ή πιο απλά σχεσιακές βάσεις δεδομένων. Η βασική δομή αναπαράστασης δεδομένων στο σχεσιακό μοντέλο είναι ο πίνακας. Συγκεκριμένα, μία σχεσιακή βάση δεδομένων αποτελείται από ένα σύνολο δυσδιάστατων πινάκων που περιέχουν δεδομένα οργανωμένα σε σειρές και στήλες. Κάθε σειρά αποτελεί μία εγγραφή ενώ κάθε στήλη αναπαριστά μία ιδιότητα ή χαρακτηριστικό της εγγραφής αυτής. Χρήστες έχουν την δυνατότητα δημιουργίας σχεσιακών βάσεων δεδομένων και διαχείρισης των δεδομένων μέσω της σχεσιακής γλώσσας SQL. Ειδικότερα, η δημιουργία μιας σχεσιακής βάσης δεδομένων υποστηρίζεται από το τμήμα της SQL που αφορά στον ορισμό πινάκων, γνωρισμάτων και εγγραφών. Η πρόσβαση και επεξεργασία των δεδομένων αυτών επιτυγχάνεται μέσω του τμ΄λήματος της SQL που αφορά στην υποβολή ερωτημάτων.

Το πλούσιο σύνολο λειτουργιών, ο εύκολος τρόπος επερώτησης δεδομένων και ο σεβασμός του ACID μοντέλου συναλλαγών (το οποίο θα αναλυθεί στο κεφάλαιο 2), κάνει τις σχεσιακές βάσεις να φαίνονται η ιδανική λύση για κάθε πρόβλημα που προκύπτει στον τομέα της αποθήκευσης δεδομένων. Ωστόσο παρουσιάζουν τους εξής περιορισμούς:

- *Επεκτασημότητα*: Όταν μία σχεσιακή βάση καλείται να διαχειριστεί μεγάλο όγκο δεδομένων και χρειάζεται περισσότερο αποθηκευτικό χώρο ή περισσότερη υπολογιστική ισχύ, θα πρέπει να μεταφερθεί σε έναν ισχυρότερο server ή να αναβαθμιστεί ο ήδη υπάρχων. Εάν η χωρητικότητα ή η υπολογιστική ισχύ του server

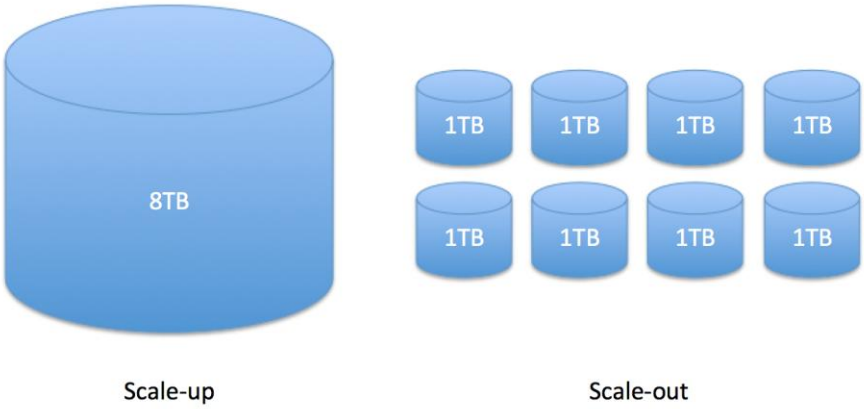
δεν επαρκεί, τότε η βάση θα πρέπει να διανεμηθεί σε πολλούς διαφορετικούς servers. Οι σχεσιακές βάσεις δεν δουλεύουν αποτελεσματικά με κατανεμημένα συστήματα καθώς ο συνδυασμός πολλαπλών πινάκων από διαφορετικούς servers είναι δύσκολος και οδηγεί ως επί το πλείστων σε μειωμένες επιδόσεις.

- *Μη δομημένα δεδομένα*: Θεμέλιος λίθος των κλασικών μοντέλων δεδομένων, επομένως και του σχεσιακού, είναι η απαίτηση για αποθήκευση, επεξεργασία και διαχείριση δομημένων δεδομένων. Συγκεκριμένα, σε μία σχεσιακή βάση τα δεδομένα ακολουθούν μία σαφώς καθορισμένη δομή ενώ ταυτόχρονα είναι προδιαγεγραμμένου τύπου και σε μεγάλο βαθμό ομοιόμορφα. Επιπλέων οι ιδιότητες των δεδομένων και οι τυχόν σχέσεις που μπορεί να υπάρχουν μεταξύ τους, είναι απαραίτητο να προκαθοριστούν σε ένα αυστηρό σχήμα. Τα παραπάνω προσδιορίζουν την έννοια των δομημένων δεδομένων που ορίζονται από ένα σχεσιακό σχήμα. Ωστόσο, οι σύγχρονες τάσεις προβάλλουν περιπτώσεις όπου η δημιουργία ενός ενιαίου σχήματος είναι είτε αδύνατη είτε ιδιαιτέρως προβληματική. Παραδείγματος χάριν, αν θεωρήσουμε το πολύπλευρο περιεχόμενο που δημιουργείται καθημερινά από χρήστες του Web 2.0, είναι προφανές ότι η δημιουργία ενός ενιαίου σχήματος δεδομένων για το προφίλ των χρηστών σε μια σειρά εφαρμογές όπως τα μέσα κοινωνικής δικτύωσης είναι μάλλον αδύνατον η πολύ δαπανηρή. εξαιτίας του εύρους των παρατυπιών και της ασάφειας των δεδομένων αυτών. Επομένως, παρόλο που οι σχεσιακές βάσεις μπορούν να αντιμετωπίσουν ορισμένες παρατυπίες ή έλλειψη δομής στα δεδομένα, στο πλαίσιο της διαχείρισης μαζικών συνόλων δεδομένων μη καθορισμένης δομής, το σχεσιακό μοντέλο αρχίζει να δείχνει τις αδυναμίες του.

1.2 Η ανάγκη για νέα μοντέλα δεδομένων

Η παγκόσμια εξάπλωση του διαδικτύου, η προοδευτική στροφή του στην Web 2.0 γενιά, και η πανταχού παρουσία των κοινωνικών δικτύων οδήγησαν στην εκθετική αύξηση του όγκου των δεδομένων που παράγονται από χρήστες καθημερινά. Οι διαδικτυακές εφαρμογές και οι Web 2.0 οργανισμοί κλήθηκαν να διαχειριστούν τεράστιους όγκους πολύπλοκων δεδομένων μεταβαλλόμενης ή αμελητέας δομής, ο χρόνος επεξεργασίας των δεδομένων αυξήθηκε δραματικά και τα ερωτήματα ανάκτησης έγιναν περιπλοκότερα. Αυτό οδήγησε στην δημιουργία νέων απαιτήσεων [2]:

- *Επεκτασιμότητα*: Ως επεκτασιμότητα ορίζεται η ικανότητα ενός συστήματος να μπορεί να χειριστεί έναν μεταβαλλόμενο φόρτο εργασίας. Στην περίπτωση των σχεσιακών βάσεων δεδομένων η επεκτασιμότητα επιτυγχάνεται με κάθετη κλιμάκωση (scale up), το οποίο πρακτικά σημαίνει αναβάθμιση των πόρων του διακομιστή όπου φυλάσσεται η βάση. Ωστόσο λόγω των σύγχρονων απαιτήσεων, ένα σύστημα θα πρέπει να είναι ικανό να κλιμακώνεται οριζόντια (scale out) εκμεταλλευόμενο τους πόρους πολλαπλών διακομιστών σε ένα κατανεμημένο σύστημα εφόσον αυτοί είναι αναγκαίοι (βλ. Εικόνα 1).



Εικόνα 1: Scale-up vs. Scale-out

- *Ευελιξία:* Παρόλο που οι σχεσιακές βάσεις είναι ευέλικτες στο εύρος των προβλημάτων τα οποία επιλύουν, είναι λιγότερο ευέλικτες ως προς τον σχεδιασμό τους. Για τις ανάγκες των σύγχρονων εφαρμογών είναι αναγκαία η χρήση ενός ευέλικτου μοντέλου δεδομένων που μπορεί να αντιμετωπίσει το σχήμα της βάσης ως κάτι που εξελίσσεται εγγενώς με την πάροδο του χρόνου. Είναι επίσης απαραίτητη η δυνατότητα ύπαρξης πολλαπλών χαρακτηριστικών σε μία συλλογή δεδομένων χωρίς να προκύψουν προβλήματα με NULL τιμές.
- *Διαθεσιμότητα*: Στις σύγχρονες εφαρμογές κρίσιμης σημασίας έως και αναγκαία είναι η χρήση συστημάτων υψηλής διαθεσιμότητας τα οποία είναι εξαρχής σχεδιασμένα να λειτουργούν ανά πάσα στιγμή, αδιάλειπτα και χωρίς διακοπές. Ακόμα και σε περίπτωση που κάποιο κομμάτι αποτύχει ή χαθεί είναι απαραίτητο να διασφαλιστεί ότι οι χρήστες θα έχουν πρόσβαση στα δεδομένα.

Οι ανάγκες αυτές οδήγησαν στην δραματική αλλαγή τόσο στο προβεβλημένο μοντέλο διαχείρισης δεδομένων όσο και στις δεσπόζουσες υπολογιστικές υποδομές και στρατηγικές ανάπτυξης εφαρμογών. Αναφορικά με τα μοντέλα δεδομένων, υπήρξαν διάφορες προσπάθειες είτε χαλάρωσης των αυστηρών περιορισμών του σχεσιακού μοντέλου είτε πειραματισμού με νέα μοντέλα όπως γράφοι. Σχετικά με τις υπολογιστικές υποδομές, υπήρξε ραγδαία εξάπλωση των υπολογιστικών νεφών και υπηρεσιών (χαμηλού κόστους) που βασίζονται σ' αυτές με αποτέλεσμα προοδευτικά να αντικαταστήσουν την κάθετη κλιμάκωση. Επιπλέον, οι μηχανικοί λογισμικού χρησιμοποιούν πλέον ευέλικτες μεθόδους ανάπτυξης λογισμικού, οι οποίες στοχεύουν στην γρήγορη ανταπόκριση των απαιτήσεων που έχουν οι χρήστες. Ας σημειωθεί ότι οι σχεσιακές βάσεις δεν μπορούν να εκμεταλλευτούν τις cloud λύσεις αποθήκευσης και υπολογιστικής ισχύος χαμηλού κόστους που υπάρχουν σήμερα. Αυτό μεταξύ άλλων τροφοδότησε την δημιουργία νέων ευέλικτων, μη σχεσιακών, κατανεμημένων και οριζοντίως επεκτάσιμων βάσεων δεδομένων οι οποίες έγιναν γνωστές με τον όρο NoSQL.

1.3 Στόχοι πτυχιακής εργασίας

Στόχος της παρούσας πτυχιακής εργασίας είναι η μελέτη των διαφόρων NoSQL μοντέλων δεδομένων και ειδικότερα της Neo4j, ενός συστήματος διαχείρισης δεδομένων που βασίζεται στο μοντέλο property graph που είναι μια ειδική έκδοση της δομής του γράφου ειδικά προσαρμοσμένη για τη διαχείριση δεδομένων. Θα επιχειρηθεί μία εκτενής μελέτη του τρόπου λειτουργίας της Neo4j καθώς και μια παρουσίαση της Cypher, της γλώσσας

υποβολής ερωτημάτων την οποία υποστηρίζει. Για τα παραπάνω επιλέχθηκε ως πειραματικό πεδίο ο κλάδος του τουρισμού, του πολιτισμού και η σύνδεσή αυτών με την αγροτοδιατροφή.

Για τις ανάγκες της εργασίας θα μελετηθούν τεχνολογίες της τρέχουσας τεχνολογικής τροχιάς με σκοπό την ανάπτυξη μιας διαδικτυακής εφαρμογής κοινωνικής δικτύωσης η οποία θα υλοποιεί διάφορα ερωτήματα τροχιάς για την υποστήριξη σύνθετων σεναρίων ανάκτησης κοινωνικών δεδομένων. Πιο συγκεκριμένα η εφαρμογή θα επιτρέπει στους χρήστες να αναζητούν νέους τουριστικούς προορισμούς προς επίσκεψη ενώ θα κρατάει ένα ιστορικό από μέρη τα οποία ο χρήστης έχει ήδη επισκεφθεί. Οι χρήστες θα έχουν την δυνατότητα να αλληλοεπιδρούν μεταξύ τους μέσω κοινών μηχανισμών κοινωνικής δικτύωσης (posts, κοινοποιήσεων, likes, σχολίων), με στόχο την ανταλλαγή πληροφοριών σχετικά με μέρη ειδικού ενδιαφέροντος.

1.4 Δομή πτυχιακής εργασίας

Η παρούσα πτυχιακή εργασία χωρίζεται σε πέντε κεφάλαια. Το κεφάλαιο 1 αποτελεί το εισαγωγικό κεφάλαιο και συνοψίζει την θεματολογία και τους στόχους της εργασίας. Στο κεφάλαιο 2 γίνεται μια εκτενής μελέτη σχετικά με το θεωρητικό υπόβαθρο του θέματος της πτυχιακής. Αναλυτικότερα, θα αναφερθούμε στους λόγους ύπαρξης των NoSQL βάσεων δεδομένων καθώς και στα διάφορα μοντέλα δεδομένων που υποστηρίζουν. Θα δοθούν επίσης ενδεικτικά παραδείγματα NoSQL βάσεων δεδομένων της τρέχουσας τεχνολογικής τροχιάς. Στο κεφάλαιο 3 επιχειρείται μια συνοπτική παρουσίαση του τρόπου λειτουργίας των βάσεων δεδομένων που υιοθετούν το γράφο ως βασικό μοντέλο δεδομένων. Γίνεται επίσης αναφορά στο property graph μοντέλο δεδομένων, καθώς και στην Cypher, την γλώσσα υποβολής ερωτημάτων της Neo4j. Το κεφάλαιο 4 αποτελεί το βασικό κεφάλαιο της εργασίας όπου γίνεται η ανάλυση της περίπτωσης χρήσης και η υλοποίηση της εφαρμογής. Εδώ παρουσιάζεται αναλυτικό ο τρόπος υλοποίησης των βασικών κομματιών της εφαρμογής με την χρήση ενδεικτικών εικόνων και αποσπάσματα κώδικα, καθώς και τα βασικά τεχνολογικά εργαλεία που χρησιμοποιήθηκαν. Τέλος, στο κεφάλαιο 5 γίνεται μία συνολική αποτίμηση της εργασίας. Αναφέρονται οι βασικές γνώσεις και δεξιότητες που αποκτήθηκαν καθώς και πιθανές μελλοντικές βελτιώσεις και επεκτάσεις.

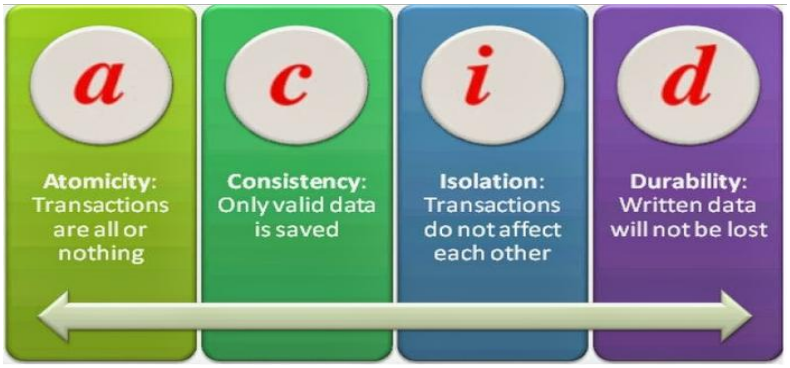
2 Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό επιδιώκεται η περιγραφή και ανάλυση των βασικών εννοιών που συνιστούν το θεωρητικό υπόβαθρο της πτυχιακής εργασίας. Ειδικότερα, θα αναφερθούν τα βασικά χαρακτηριστικά των NoSQL βάσεων δεδομένων και θα επιχειρηθεί η κατηγοριοποίησή τους βάσει των διαφορετικών μοντέλων δεδομένων που υποστηρίζουν.

2.1 ACID συναλλαγές και θεώρημα CAP

Οι πρώιμες εφαρμογές που βασιζόταν στο σχεσιακό μοντέλο αντιμετώπιζαν προβλήματα συγχρονισμού και ακεραιότητας δεδομένων. Προκειμένου να διασφαλιστεί η ακεραιότητα των δεδομένων προτάθηκε ένα μοντέλο συναλλαγών. Ως συναλλαγή θεωρείται μία διεργασία στην οποία πολλαπλά ερωτήματα μπορούν να εκτελεστούν σε σειρά. Το μοντέλο συναλλαγών που επικράτησε έγινε γνωστό με το ακρωνύμιο ACID που αποδίδει τις βασικές ιδιότητες που πρέπει να ικανοποιεί κάθε συναλλαγή. Οι ιδιότητες αυτές είναι (βλ. Εικόνα 2) [3]:

- *Ατομικότητα (Atomicity)*: Η ατομικότητα απαιτεί από την κάθε συναλλαγή να ακολουθεί τον κανόνα "όλα ή τίποτα". Μια ατομική συναλλαγή είτε εκτελείται πλήρως, είτε δεν εκτελείται καθόλου. Αν ένα μέρος της συναλλαγής αποτύχει, τότε θα αποτύχει όλη η συναλλαγή και το σύστημα θα επιστρέψει στην αρχική κατάσταση που βρισκόταν πριν η συναλλαγή αυτή αρχίσει [4].
- *Συνέπεια (Consistency)*: Μία συνεπής συναλλαγή εξασφαλίζει ότι στο τέλος κάθε συναλλαγής το σύστημα βρίσκεται σε μια συνεπή κατάσταση. Αν η συναλλαγή ολοκληρωθεί με επιτυχία, τότε όλες οι αλλαγές του συστήματος έχουν πραγματοποιηθεί και η κατάσταση του είναι συνεπής. Αν σε μια συναλλαγή υπάρξει κάποιο σφάλμα, τότε τυχόν αλλαγές που έχουν γίνει θα επανέλθουν αυτόματα και έτσι το σύστημα θα επανέλθει στην αρχική του κατάσταση [4].
- *Απομόνωση (Isolation)*: Αν τυχόν μία συναλλαγή εκτελεστεί ταυτόχρονα με μία άλλη, πρέπει να συμπεριφέρεται σαν να ήταν η μόνη συναλλαγή που εκτελείται στο σύστημα. Αυτό σημαίνει ότι κατά την διάρκεια εκτέλεσής της, μία συναλλαγή δεν πρέπει να παρεμβαίνει με τις υπόλοιπες [4].
- *Ανθεκτικότητα (Durability)*: Όταν μια συναλλαγή ολοκληρωθεί επιτυχώς, όλες οι αλλαγές που πιθανόν έγιναν στο σύστημα είναι μόνιμες, και δεν θα υπάρξει απώλεια πληροφοριών ακόμη και σε περίπτωση βλάβης του συστήματος [4].



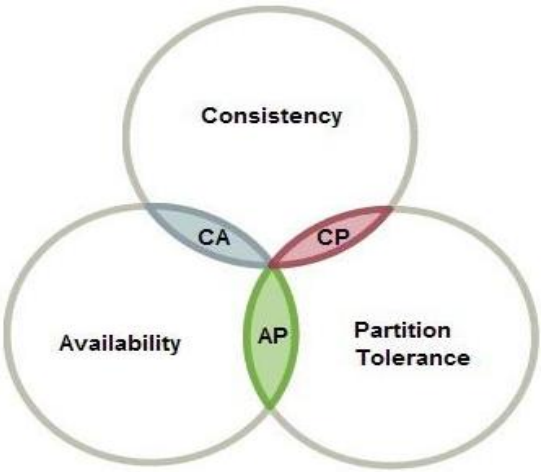
Εικόνα 2: ACID μοντέλο συναλλαγών

Η δραματική αύξηση των δεδομένων που χειρίζονται οι σύγχρονες εφαρμογές κατέδειξε τις αδυναμίες του σχεσιακού μοντέλου και η ανάγκη για την χρήση κατανεμημένων συστημάτων αποθήκευσης ήταν προφανής. Η αξιοποίηση κατανεμημένων συστημάτων αποθήκευσης δεδομένων επιτρέπει την χρήση πολλαπλών λιγότερο ισχυρών μηχανημάτων για τον περιορισμό κόστους και αποτρέπει μεμονωμένα σημεία αποτυχίας. Ωστόσο η εφαρμογή των ACID ιδιοτήτων σε ένα κατανεμημένο σύστημα είναι δύσκολη. Οι βασικές απαιτήσεις ενός κατανεμημένου συστήματος είναι οι εξής:

- *Συνοχή (Consistency)*: Κάθε κόμβος σε ένα κατανεμημένο σύμπλεγμα επιστρέφει την ίδια, πιο πρόσφατη, επιτυχή εγγραφή.
- *Διαθεσιμότητα (Availability)*: Το σύστημα είναι πάντα διαθέσιμο. Για κάθε αίτημα που λαμβάνεται, παράγεται μία απάντηση είτε επιτυχής είτε όχι.
- *Ανοχή Διαμερισμάτων (Partition Tolerance)*: Το σύστημα συνεχίζει να λειτουργεί παρά τον αυθαίρετο αριθμό μηνυμάτων που χάνονται ή καθυστερούν λόγω της δυσλειτουργίας ενός κόμβου.

Αυτοί οι παράγοντες αποτελούν τους τρεις βασικούς πυλώνες του θεωρήματος ακεραιότητας συναλλαγών του Brewer (ή θεώρημα CAP) το οποίο δηλώνει ότι σε ένα οριζοντίως κλιμακούμενο κατανεμημένο σύστημα, είναι αδύνατο να εγγυηθούν και τα τρία αυτά χαρακτηριστικά ταυτόχρονα (βλ. Εικόνα 3). Πρέπει λοιπόν να γίνουν συμβιβασμοί και να θυσιαστεί τουλάχιστον ένα υπέρ του άλλου [5]. Για τον λόγο αυτό τα συστήματα βάσεων δεδομένων μπορούν να χωριστούν στις εξής κατηγορίες:

- *CA*: Συστήματα όπως οι σχεσιακές βάσεις τα οποία θυσιάζουν την ανοχή διαμερισμάτων για καλύτερη συνοχή και διαθεσιμότητα δεδομένων.
- *CP*: Συστήματα τα οποία επιτυγχάνουν την ανοχή διαμερισμάτων και θυσιάζουν την διαθεσιμότητα για καλύτερη συνοχή.
- *AP*: Συστήματα τα οποία επιτυγχάνουν την ανοχή διαμερισμάτων και θυσιάζουν την συνοχή των δεδομένων για να είναι διαθέσιμα ανά πάσα στιγμή.



Εικόνα 3: Θεώρημα CAP

2.2 NoSQL Συστήματα βάσεων δεδομένων

Τα NoSQL συστήματα αντιπροσωπεύουν μία νέα κατηγορία τεχνολογιών διαχείρισης δεδομένων οι οποίες αποτελούν μια διαφορετική προσέγγιση στον τρόπο αποθήκευσης δεδομένων από το σχεσιακό μοντέλο. Η εκθετική αύξηση του όγκου των δεδομένων, η

συχνότητα πρόσβασης των δεδομένων αυτών καθώς και οι αυξημένες απαιτήσεις επεξεργασίας και απόδοσης αποτέλεσαν τα βασικά κίνητρα για την δημιουργία των NoSQL βάσεων δεδομένων. Οι NoSQL βάσεις παρέχουν ελαστικότητα, επεκτασιμότητα, μπορούν να διαχειριστούν μεγάλους όγκους δεδομένων και μπορούν να εκμεταλλευτούν τις φθηνές λύσεις των cloud υπηρεσιών αποθήκευσης. Παρόλο που ο όρος NoSQL αναφέρεται σε μία κατηγορία πολλών διαφορετικών τεχνολογιών διαχείρισης δεδομένων, όλες οι NoSQL βάσεις έχουν κάποια κοινά χαρακτηριστικά [6]:

- *Δυναμικό σχήμα*: Αντίθετα με τις σχεσιακές βάσεις όπου πρέπει να προκαθοριστεί ένα σχήμα πριν την προσθήκη των δεδομένων, οι NoSQL βάσεις επιτρέπουν την αλλαγή του σχήματος των δεδομένων κατά την διάρκεια εκτέλεσης. Αυτό επιτρέπει την εύκολη αλλαγή της δομής των δεδομένων χωρίς διακοπές λειτουργίας.
- *Αυτόματοι διαχωρισμοί δεδομένων*: Λόγω της δομής τους, οι σχεσιακές βάσεις είναι δύσκολο να επεκταθούν οριζοντίως. Οι NoSQL βάσεις μπορούν να διανέμουν αυτόματα τα δεδομένα σε έναν αυθαίρετο αριθμό από server αφαιρώντας την λογική του διαχωρισμού από την ίδια την εφαρμογή. Τα δεδομένα και τα ερωτήματα που τίθενται εξισορροπούνται αυτόματα, ενώ σε περίπτωση αποτυχίας ενός server, ο εν λόγω server μπορεί να αντικατασταθεί γρήγορα χωρίς πτώση της απόδοσης του συστήματος.
- *Caching μηχανισμοί*: Οι περισσότερες NoSQL βάσει παρέχουν caching μηχανισμούς οι οποίοι διατηρούν τα δεδομένα που χρησιμοποιούνται συχνότερα στη μνήμη του συστήματος όσο το δυνατόν περισσότερο.
- *Αντιγραφή*: Πολλές NoSQL βάσεις κρατάνε αυτόματα αντίγραφα επιτρέποντας την υψηλή διαθεσιμότητα των δεδομένων ενώ ταυτόχρονα υπάρχει η δυνατότητα ανάκτησης σε περίπτωση αποτυχίας ενός server χωρίς την χρήση επιπλέων εξειδικευμένων εφαρμογών για την διαχείριση τέτοιων εργασιών.

2.3 Πότε είναι χρήσιμα

Όσον αφορά τις διάφορες τεχνολογίες βάσεων δεδομένων, δεν υπάρχει μια λύση που να καλύπτει όλο το εύρος των προβλημάτων. Είναι λογικό να σκεφτεί κανείς ότι τα NoSQL συστήματα ήρθαν για να αντικαταστήσουν το σχεσιακό μοντέλο, κάτι το οποίο δεν ισχύει. Ο σκοπός των NoSQL βάσεων δεν είναι να αντικαταστήσουν τις σχεσιακές, αλλά να τις συμπληρώσουν. Κάθε ένα από τα δύο μοντέλα είναι κατάλληλο για διαφορετικές εφαρμογές και διαφορετικό φόρτο εργασίας, το οποίο σημαίνει ότι κανένα από τα δύο μοντέλα δεν μπορεί να εκτοπίσει το άλλο. Παρόλο που οι NoSQL βάσεις έχουν κερδίσει δημοτικότητα για την ταχύτητα και την επεκτασιμότητά τους, εξακολουθούν να υπάρχουν περιπτώσεις στις οποίες είναι προτιμότερη η χρήση μιας δομημένης SQL βάσης δεδομένων. Σε περιπτώσεις που η αξιοπιστία και η συνοχή των δεδομένων είναι απαραίτητη σε μία εφαρμογή, τότε οι σχεσιακές βάσεις αποτελούν την προφανή επιλογή. Οι NoSQL βάσεις συνήθως δεν υποστηρίζουν εγγενώς το ACID μοντέλο συναλλαγών, κάτι το οποίο αποτελεί πρόβλημα σε συγκεκριμένες περιπτώσεις όπως οι τραπεζικές συναλλαγές, όπου τα ACID χαρακτηριστικά είναι αναγκαία. Ωστόσο, σε περιπτώσεις όπου η αξιοπιστία και η συνοχή δεν είναι απαραίτητη, ενώ τα δεδομένα είναι αδόμητα και ο όγκος τους πολύ μεγάλος, είναι προτιμότερη η χρήση μίας NoSQL βάσης δεδομένων καθώς λόγω της έλλειψης των ACID συναλλαγών έχουν καλύτερες επιδόσεις, μειωμένους χρόνους ανάγνωσης και εγγραφής δεδομένων και αυξημένες δυνατότητες επεκτασιμότητας [7].

2.4 Προβεβλημένα μοντέλα δεδομένων

Κάποτε η απάντηση σε όλες τις ανάγκες αποθήκευσης και διαχείρισης δεδομένων ήταν οι σχεσιακές βάσεις. Όταν αναφερόμαστε στις NoSQL βάσεις δεν μιλάμε για μία συγκεκριμένη βάση δεδομένων αλλά για μια οικογένεια μη σχεσιακών βάσεων δεδομένων που το μόνο κοινό στοιχείο που έχουν οι περισσότερες από αυτές μεταξύ τους είναι ότι δεν ακολουθούν το σχεσιακό μοντέλο. Υπάρχουν διαφορετικοί τύποι NoSQL βάσεων για να καλύψουν τις ανάγκες των σύγχρονων εφαρμογών ανάλογα με την περίπτωση χρήσης και την λειτουργικότητα που απαιτείται. Παρόλο που οι τεχνολογίες, οι τύποι δεδομένων και οι περιπτώσεις χρήσης ποικίλλουν στις διάφορες NoSQL βάσεις, είναι γενικά αποδεκτό ότι χωρίζονται σε τέσσερις κατηγορίες.

2.4.1 Key-Value Stores

Οι βάσεις δεδομένων κλειδιού-τιμής (key-value stores) αποτελούν την πιο απλή και θεμελιώδη εφαρμογή των NoSQL βάσεων. Τα δεδομένα αποτυπώνονται ως ζεύγη κλειδιού-τιμής χωρίς κάποια συγκεκριμένη δομή ή σχέση μεταξύ τους. Δεδομένου ότι οι τιμές είναι ασαφής συστοιχίες από byte χωρίς κάποια ερμηνεία, τα κλειδιά αποτελούν τον μοναδικό τρόπο ανάκτησης δεδομένων. Λόγω του απλού μοντέλου δεδομένων, οι βάσεις κλειδιού-τιμής προσφέρουν υψηλές αποδόσεις, εύκολη επεκτασιμότητα και δεν έχουν τους περιορισμούς ενός σχήματος. Νέες τιμές διαφόρων τύπων μπορούν να προστίθενται στην βάση, χωρίς να επηρεάζεται η διαθεσιμότητά της [8]. Αυτό το απλό μοντέλο συνήθως δεν επαρκεί για σύνθετες εφαρμογές, ωστόσο αυτή η απλότητα προσφέρει την ταχύτητα οπού σε άλλα συστήματα είναι ελλιπής και καθιστά το συγκεκριμένο μοντέλο ελκυστικό σε συγκεκριμένες περιπτώσεις. Παραδείγματα βάσεων δεδομένων κλειδιού-τιμής αποτελούν η Redis η οποία επιτυγχάνει εξαιρετικές επιδόσεις αποθηκεύοντας τα δεδομένα στην μνήμη, και η Riak - μία κατανεμημένη key-value βάση υψηλής διαθεσιμότητας. Κάποιες περιπτώσεις χρήσης των βάσεων κλειδιού-τιμής είναι οι εξής [8]:

- Αποθήκευση απλών δεδομένων σε μικρή κλίμακα (όπως user sessions ή ηλεκτρονικά καλάθια αγορών).
- Αποθήκευση δεδομένων σε προσωρινή μνήμη για άμεση ή μελλοντική χρήση.
- Αποθήκευση δεδομένων σε κρυφή μνήμη (cache) για γρηγορότερες ανακτήσεις.

2.4.2 Document Stores

Οι βάσεις δεδομένων εγγράφων είναι παρόμοιες με τις βάσεις κλειδιού-τιμής με την έννοια ότι τα δεδομένα αποθηκεύονται ως τιμές, ενώ υπάρχει ένα κλειδί το οποίο δρα ως μοναδικό αναγνωριστικό για την κάθε τιμή. Η κύρια διαφορά τους βρίσκεται στο ότι οι βάσεις δεδομένων εγγράφων αποθηκεύουν τα δεδομένα μέσα σε έγγραφα κωδικοποιημένα συνήθως σε JSON μορφή. Ως έγγραφο ορίζεται ένα σύνολο ημι-δομημένων ζευγών κλειδιού-τιμής. Η ημι-δομημένη φύση των δεδομένων σε μια βάση εγγράφων επιτρέπει τον πιο αποτελεσματικό χειρισμό πιο σύνθετων δομών όπως είναι τα εμφωλευμένα αντικείμενα. Η αποθήκευση δεδομένων σε JSON έγγραφα επιτρέπει την υποστήριξη πολλαπλών τύπων δεδομένων, ενώ όπως και στις βάσεις κλειδιού-τιμής δεν υπάρχουν περιορισμοί σχήματος [8].

2.4.3 Column Stores

Εμπνευσμένες από την BigTable της Google, οι βάσεις δεδομένων στηλών αποτελούν ένα κατανεμημένο σύστημα αποθήκευσης δεδομένων σχεδιασμένο να χειρίζεται τεράστιο όγκο δεδομένων. Τα δεδομένα δεν αποθηκεύονται σε πίνακες όπως σε μία σχεσιακή βάση στηλών, αλλά σε μαζικά κατανεμημένες αρχιτεκτονικές. Τα δεδομένα αποθηκεύονται ανά στήλη και όχι ανά σειρά. Οικογένειες στηλών (column families) ομαδοποιούν σχετικές στήλες για καλύτερη οργάνωση και διαμερισμό των δεδομένων. Κάθε οικογένεια στηλών μπορεί να έχει απεριόριστο αριθμό στηλών. Κάθε στήλη μπορεί να δημιουργηθεί κατά τον χρόνο εκτέλεσης ενώ οι οικογένειες στηλών συχνά πρέπει να είναι προκαθορισμένες. Αυτό πρακτικά σημαίνει ότι προσφέρουν λιγότερη ευελιξία από τις βάσεις κλειδιού-τιμής και τις βάσεις εγγράφων [9]. Παρόλο που το συγκεκριμένο μοντέλο μοιάζει με το σχεσιακό, η κύρια διαφορά του είναι στον χειρισμό των null τιμών. Οι σχεσιακές βάσεις αποθηκεύουν null τιμές για τα κελιά που δεν έχουν τιμή ενώ μια οικογένεια στηλών αποθηκεύει μόνο τα ζευγάρια τιμής-κλειδιού που χρειάζεται. Αυτό κάνει τις βάσεις στηλών ιδιαίτερα αποτελεσματικές σε περιπτώσεις που υπάρχει τεράστιος όγκος δεδομένων με διαφορετικά χαρακτηριστικά. Τέτοια δεδομένα μπορούν να αποθηκευτούν αποτελεσματικά και από τις βάσεις εγγράφων, ωστόσο οι σύνθετες δομές ενός JSON εγγράφου όπως τα εμφωλευμένα αντικείμενα ή οι σύνδεσμοι μπορεί να είναι πιο δαπανηρά όσον αφορά την χωρητικότητα [8].

2.4.4 Graph Databases

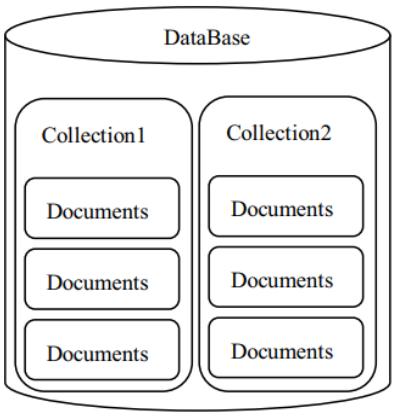
Οι τρεις κατηγορίες βάσεων δεδομένων που προαναφέρθηκαν είναι σχεδιασμένες να αποθηκεύουν αποτελεσματικά μη δομημένα δεδομένα για λόγους επεκτασιμότητας διαθεσιμότητας και ευελιξίας. Συνήθως όμως δεν μπορούν να αποτυπώσουν συνδέσεις μεταξύ δεδομένων. Πιθανών σχέσεις που μπορεί να έχουν τα δεδομένα μεταξύ τους πρέπει να αποτυπώνονται στο κομμάτι της εφαρμογής. Αντίθετα οι βάσεις δεδομένων τύπου γράφου είναι σχεδιασμένες να χειρίζονται αποτελεσματικά συνδεδεμένα δεδομένα [8]. Βασισμένες στην μαθηματική έννοια του γράφου, οι βάσεις δεδομένων τύπου γράφου υιοθετούν ως βασικές έννοιες τον κόμβο ή κορυφή (vertex) και τις συνδέσεις ή άκρα (edges) τα οποία χρησιμοποιούνται για την ταξινόμηση και αποθήκευση των δεδομένων. Οι βάσεις δεδομένων τύπου γράφου είναι κατάλληλες για δεδομένα που μπορούν να αποτυπωθούν με την μορφή ιεραρχίας ή για εφαρμογές που βασίζονται και αξιοποιούν περισσότερο τις σχέσεις μεταξύ των δεδομένων, καθώς αντικαθιστούν δαπανηρές λειτουργίες όπως τα αναδρομικά joins, με αποτελεσματικότερες διασχίσεις (traversals) γράφων. Περιπτώσεις χρήσης αποτελούν υπηρεσίες τοποθεσίας, εφαρμογές όπως συστήματα πλοήγησης όπου απαιτείται εύρεση συντομότερων μονοπατιών, κοινωνικά δίκτυα και συστήματα προτάσεων [8].

2.5 Συστήματα NoSQL Βάσεων Δεδομένων

2.5.1 MongoDB

Η MongoDB είναι μια open source βάση δεδομένων εγγράφων η οποία συλλέγει έγγραφα σε collections (συλλογές) ανάλογα με την δομή τους κωδικοποιημένα σε BSON (Binary JSON) μορφή. Έγγραφα με διαφορετική δομή μπορούν να συνυπάρχουν στο ίδιο collection, ωστόσο συνιστάται η αποθήκευση παρόμοιων εγγράφων στο ίδιο collection για καλύτερη αποδοτικότητα. Κύρια χαρακτηριστικά της MongoDB αποτελούν η ανθεκτικότητα

και η διαθεσιμότητα των δεδομένων λόγω των μηχανισμών αντιγράφων (replicas) τους οποίους παρέχει. Κάθε replica αποτελείται από δύο ή περισσότερα αντίγραφα των δεδομένων. Κάθε λειτουργία εγγραφής η ανάγνωσης γίνεται στο πρωτεύων αντίγραφο. Σε περίπτωση που το πρωτεύων αντίγραφο αποτύχει εκτελείται μια εκλογική διαδικασία κατά την οποία προσδιορίζεται ποιο δευτερεύων αντίγραφο θα γίνει πρωτεύων. (MongoDB Wiki) Η MongoDB χρησιμοποιείται κυρίως για αναλύσεις πραγματικού χρόνου, για αποθήκευση δεδομένων καταγραφής (logging data) ή για διαχείριση δεδομένων προϊόντων όπως καταλόγους προϊόντων ή διαχείριση εμπορευμάτων [10]. Το μοντέλο δεδομένων της MongoDB φαίνεται στην Εικόνα 4.



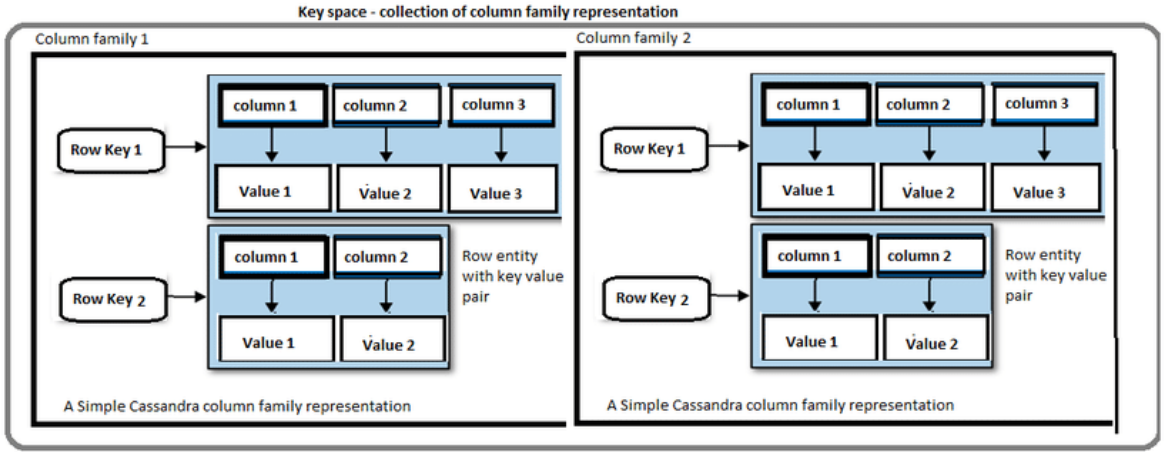
Εικόνα 4: MongoDB Data Model

Ας πάρουμε ως παράδειγμα το πρόβλημα του καταλόγου προϊόντων. Ένας κατάλογος προϊόντων πρέπει να αποθηκεύει πολλά προϊόντα τα οποία μπορεί να έχουν διαφορετικά χαρακτηριστικά το καθένα. Σε μία σχεσιακή βάση μία πιθανή λύση είναι η δημιουργία ενός πίνακα για κάθε προϊόν όπου κάθε πίνακας έχει τα δικά του πεδία ανάλογα με τα χαρακτηριστικά του προϊόντος. Αυτή η λύση δεν είναι ιδανική διότι για κάθε νέο προϊόν πρέπει να δημιουργηθεί ένας νέος πίνακας ενώ κάθε ερώτημα πρέπει να είναι ειδικά διαμορφωμένο για κάθε διαφορετικό προϊόν. Μία δεύτερη προσέγγιση θα ήταν η δημιουργία ενός πίνακα για όλα τα προϊόντα όπου ο κάθε πίνακας θα έχει ως πεδία όλα τα χαρακτηριστικά όλων των προϊόντων. Αυτή η προσέγγιση παρόλο που είναι πιο ευέλικτη, είναι εξίσου προβληματική στην περίπτωση που προστεθεί ένα νέο προϊόν με καινούργια χαρακτηριστικά αφού θα ήταν αναγκαίος ο εκ νέου σχεδιασμός του πίνακα. Το ίδιο πρόβλημα στην MongoDB λύνεται με την χρήση ενός collection για την αποθήκευση όλων των προϊόντων. Το ευέλικτο σχήμα της MongoDB επιτρέπει την ύπαρξη εγγράφων με διαφορετικά πεδία στο ίδιο collection.

2.5.2 Cassandra

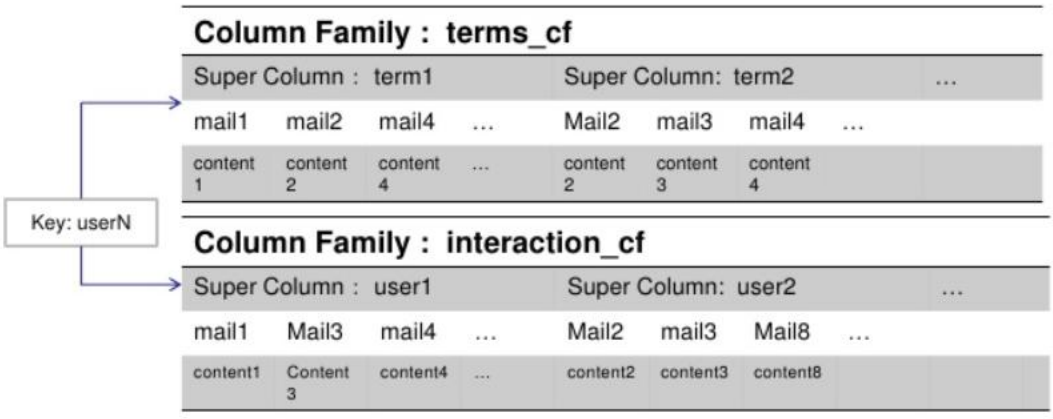
Η Cassandra είναι μια open-source κατανεμημένη βάση δεδομένων στηλών σχεδιασμένη να αποθηκεύει και να χειρίζεται τεράστιο όγκο δεδομένων χωρίς σημεία αποτυχίας. Τα δεδομένα στην Cassandra αποθηκεύονται βάσει στηλών με τη μορφή ενός ζεύγους κλειδιού-τιμής. Υπέρ στήλες (Super Columns) ομαδοποιούν στήλες με ένα κοινό όνομα και χρησιμοποιούνται για τον σχεδιασμό πιο σύνθετων δεδομένων (π.χ. διευθύνσεις). Σειρές (rows) ομαδοποιούν στήλες και υπέρ στήλες και σχηματίζουν μοναδικά δεδομένα. Κάθε σειρά έχει ένα κλειδί που αποτελεί το μοναδικό αναγνωριστικό της. Οι σειρές

οργανώνονται σε οικογένειες στηλών ανάλογα με την λογική τους σημασία ενώ τα Keyspaces συγκεντρώνουν όλες τις οικογένειες στηλών και αποτελούν το ανώτατο επίπεδο πληροφορίας [11]. Το μοντέλο δεδομένων της Cassandra φαίνεται αναλυτικά στην Εικόνα 5.



Εικόνα 5: Cassandra Data Model

Η Cassandra αρχικά δημιουργήθηκε από το Facebook για να λύσει το πρόβλημα της αναζήτησης μηνυμάτων (inbox search). Το πρόβλημα αυτό χωρίζεται σε δύο βασικά σκέλη: την αναζήτηση όρων μέσα σε ένα μήνυμα και την αναζήτηση μηνυμάτων βάσει όνομα χρήστη. Το πρόβλημα επιλύθηκε με την χρήση δύο οικογενειών στηλών όπως φαίνεται στην Εικόνα 6. Στην πρώτη περίπτωση υπάρχουν υπερ-στήλες που αποτελούνται από το σύνολο των λέξεων που συγκροτούν ένα μήνυμα ενώ σε κάθε στήλη αποθηκεύεται κάθε λέξη ξεχωριστά και το id του χρήστη αποτελεί το κλειδί της κάθε σειράς. Στην δεύτερη περίπτωση το id του χρήστη αποτελεί πάλι το κλειδί της κάθε σειράς ενώ το id του κάθε πιθανού παραλήπτη λειτουργεί ως υπέρ στήλη. Σε κάθε στήλη αποθηκεύεται το κάθε μήνυμα ξεχωριστά. Η διαδικασία αναζήτησης γίνεται ιδιαίτερα γρήγορα λόγω των caching μηχανισμών που διαθέτει η Cassandra [12].



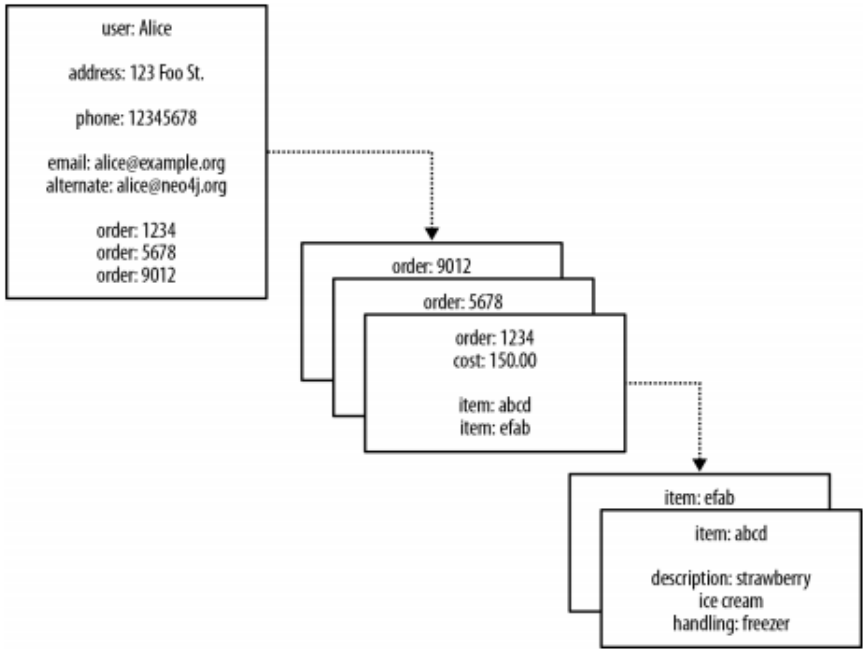
Εικόνα 6: Το πρόβλημα της αναζήτησης μηνυμάτων

3 Βάσεις Δεδομένων Γράφων και Neo4j

Οι βάσεις δεδομένων γράφων λύνουν αποτελεσματικά ένα από τα βασικότερα προβλήματα των τελευταίων ετών όσον αφορά την διαχείριση δεδομένων: την αξιοποίηση πολύπλοκων σχέσεων σε σύνολα άκρος συνδεδεμένων δεδομένων. Για δεδομένα οποιουδήποτε μεγέθους, οι βάσεις δεδομένων γράφων αποτελούν τον βέλτιστο τρόπο διαχείρισης και επερώτησης συνδεδεμένων δεδομένων.

3.1 Βάσεις δεδομένων γράφων και συνδεδεμένα δεδομένα

Ως συνδεδεμένα δεδομένα ορίζονται τα δεδομένα για τα οποία πρέπει να κατανοήσουμε τις μεταξύ τους σχέσεις. Για την αποτελεσματική κατανόηση, πολλές φορές είναι απαραίτητο να ορίσουμε και να ονομάσουμε τις σχέσεις αυτές σε μία βάση. Έχουν γίνει αρκετές απόπειρες διαχείρισης και αποθήκευσης συνδεδεμένων, ημι-δομημένων συνόλων δεδομένων από σχεσιακές βάσεις δεδομένων. Ωστόσο, παρόλο που διαχειρίζονται αποτελεσματικά δομημένα δεδομένα σε δομές πινάκων, αδυνατούν να μοντελοποιούν τα μη δομημένα και συνδεδεμένα δεδομένα του πραγματικού κόσμου. Στο σχεσιακό μοντέλο μπορούν να οριστούν σχέσεις, αλλά μόνο κατά την διάρκεια της μοντελοποίησης και μόνο ως ένα μέσο σύνδεσης πινάκων. Ωστόσο, για την αποτελεσματική διαχείριση συνδεδεμένων δεδομένων, είναι αναγκαίο να προσδιορίσουμε τη σημασιολογία των σχέσεων που συνδέουν οντότητες καθώς και το βάρος που μπορεί να έχει μία σχέση. Αυτές οι λειτουργίες δεν παρέχονται στο σχεσιακό μοντέλο. Αντίθετα, καθώς ο όγκος των δεδομένων αυξάνεται και η δομή τους γίνεται όλο και πιο περίπλοκη, το σχεσιακό μοντέλο επιβαρύνεται με περίπλοκα joins και λογικές ελέγχου null τιμών [13].



Εικόνα 7: Σχέσεις σε NoSQL βάσεις εγγράφων

Από την άλλη, οι NoSQL βάσεις δεδομένων είναι αποτελεσματικές στο να χειρίζονται μη δομημένα δεδομένα σε μη συνδεδεμένα σύνολα εγγράφων στηλών ή ζευγών κλειδιού-

τιμής, κάτι το οποίο καθιστά την δημιουργία σχέσεων εξίσου προβληματική. Ο μοναδικός τρόπος δημιουργίας σχέσεων σε αυτά τα μοντέλα είναι η ενσωμάτωση μοναδικών αναγνωριστικών σε ένα σύνολο, τα οποία αναφέρονται σε ένα άλλο σύνολο (βλ. Εικόνα 7). Με αυτόν τον τρόπο είναι απαραίτητη η ένωση των διάφορων συνόλων στο επίπεδο εφαρμογής κάτι το οποίο είναι εξίσου δαπανηρό [13].

Αντίθετα με τις υπόλοιπες προσεγγίσεις βάσεων δεδομένων, οι βάσεις δεδομένων γράφων δεν αποθηκεύουν τα δεδομένα σε διαφορετικούς πίνακες ή σε διαφορετικά έγγραφα, αλλά σε μία μοναδική δομή δεδομένων, τον γράφο. Είναι σχεδιασμένες να χειρίζονται τις σχέσεις μεταξύ των δεδομένων εξίσου σημαντικά με τα ίδια τα δεδομένα εξαλείφοντας έτσι την ανάγκη σύνδεσης των δεδομένων μέσω ειδικών ιδιοτήτων όπως είναι τα ξένα κλειδιά. Είναι επίσης κατασκευασμένες να δουλεύουν με συναλλαγές, συνεπώς ακολουθούν το ACID μοντέλο συναλλαγών και είναι σχεδιασμένες με στόχο την διαθεσιμότητα και την ακεραιότητα συναλλαγών. Τα βασικά στοιχεία μιας βάσης δεδομένων γράφων είναι δύο: οι κορυφές και οι άκρες. Οι κορυφές αντιπροσωπεύουν τις οντότητες του μοντέλου δεδομένων ενώ οι άκρες τις μεταξύ τους σχέσεις. Η δημιουργία κορυφών και άκρων και η σύνθεσή τους σε συνδεδεμένες δομές, επιτρέπει την μοντελοποίηση απλών ή σύνθετων μοντέλων τα οποία κάνουν τα δεδομένα να παραμένουν παρόμοια με την μορφή τους στον πραγματικό κόσμο.

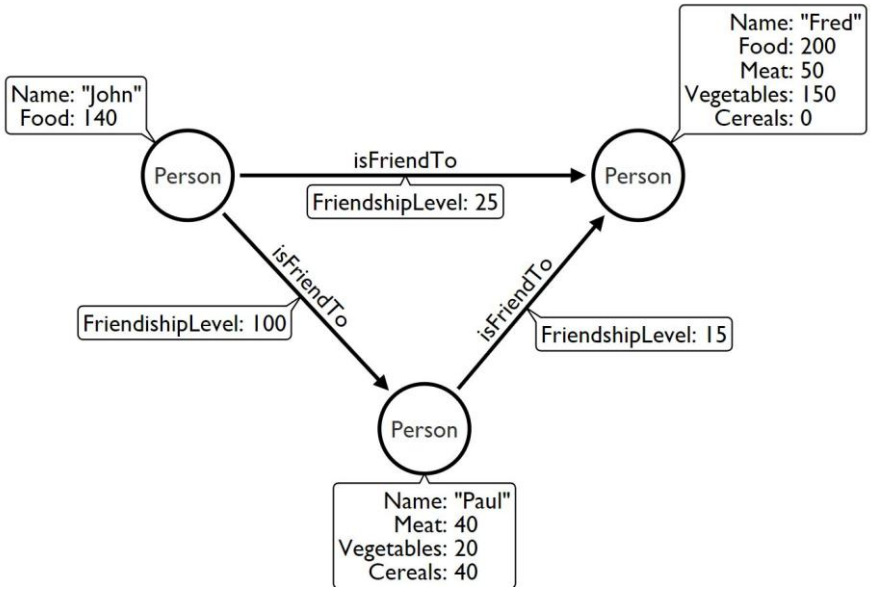
Το βασικό μειονέκτημα του συγκεκριμένου μοντέλου είναι η δύσκολη επίτευξη οριζόντιας κλιμάκωσης (*βλ. ενότητα 1.2*), καθώς δεν υπάρχει ένα προφανές σημείο διαχωρισμού σε έναν συνδεδεμένο γράφο. Ωστόσο, αυτό το μειονέκτημα αντισταθμίζεται από το γεγονός ότι το χρονικό κόστος μιας διαδικασίας ανάγνωσης σε μία κορυφή ή άκρη παραμένει σταθερό ανεξάρτητα από το μέγεθος του γράφου [14].

3.1.1 Property Graph

Οι βάσεις δεδομένων γράφων είναι ένα αποτελεσματικό εργαλείο μοντελοποίησης δεδομένων. Βασισμένες στη μαθηματική έννοια του γράφου, αποτελούνται από κορυφές και άκρες οι οποίες μπορούν να οργανώσουν περίπλοκες δομές. Γενικά οι γράφοι αντιπροσωπεύουν οντότητες ως κόμβους, και τον τρόπο που σχετίζονται μεταξύ τους οι οντότητες αυτές ως σχέσεις. Η ικανότητα αποτύπωσης οντοτήτων και των μεταξύ τους σχέσεων, επιτρέπει την μοντελοποίηση ενός μεγάλου εύρους σεναρίων να μπορεί να επιλυθεί με τη βοήθεια ενός γράφου. Παρόλο που τα βασικά στοιχεία ενός γράφου είναι οι κορυφές και οι άκρες, δεν είναι τα μοναδικά. Υπάρχει μία μεγάλη ποικιλία τύπων γράφων καθένας από τους οποίους προσφέρει επιπλέον χαρακτηριστικά διαμορφώνοντας πιο ευέλικτους και εκφραστικούς γράφους. Παρακάτω αναφέρονται ενδεικτικά κάποιοι βασικοί τύποι γράφων [15][16]:

- *Απλός γράφος*: Ένας μη κατευθυνόμενος γράφος χωρίς παράλληλες άκρες και χωρίς βρόχους.
- *Γράφοι με βάρη στις ακμές (weighted graph)*: Γράφοι όπου με κάθε ακμή τους έχει συσχετιστεί ένας αριθμός που ονομάζεται βάρος.
- *Κατευθυνόμενοι γράφοι*: Γράφοι των οποίων οι άκρες προσδιορίζουν μία κατεύθυνση.
- *Multi-graphs*: Γράφοι στους οποίους υπάρχουν πολλαπλές άκρες μεταξύ δύο κορυφών.
- *Σημασιολογικοί γράφοι*: Γράφοι που χρησιμοποιούνται για τη μοντελοποίηση νοητικών δομών.
- *Resource description framework (RDF) graph*: Ένας τυποποιημένος γράφος που δημιουργήθηκε από το W3C ο οποίος επιτρέπει την κωδικοποίηση, την ανταλλαγή και την επαναχρησιμοποίηση δομημένων μεταδεδομένων [17].

Ένας από τους πιο κοινούς τύπους γράφων είναι ο property graph (γράφος ιδιοτήτων). Οι γράφοι τέτοιου τύπου επιτρέπουν την ανάθεση ετικετών (labels) και ιδιοτήτων (properties) στις κορυφές και τις άκρες του γράφου. Επιπλέον, κάθε κορυφή και άκρη έχει ένα μοναδικό αναγνωριστικό το οποίο μπορεί να χρησιμοποιηθεί ως σημείο αναφοράς για την συσχέτιση επιπλέον μετα-δεδομένων σε κάθε κορυφή ή άκρη, με την μορφή ζευγών κλειδιού-τιμής. Ο property graph αποτελεί μία βολική δομή επειδή περιέχει τα βασικά χαρακτηριστικά όσον αφορά την μοντελοποίηση γράφων ενώ απλές μορφοποιήσεις του έχουν ως αποτέλεσμα την δημιουργία διαφορετικών τύπων γράφων [15]. Έτσι τα συστήματα βάσεων δεδομένων που υποστηρίζουν το μοντέλο του property graph, υποστηρίζουν έμμεσα και άλλους τύπους γράφων. Στην Εικόνα 8 δίνεται μία οπτική αναπαράσταση των βασικών στοιχείων ενός property graph και πως αυτά συνδέονται μεταξύ τους.



Εικόνα 8: Labeled Property Graph

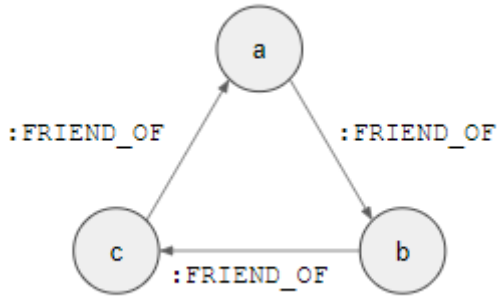
3.1.2 Γλώσσες υποβολής ερωτημάτων γράφων

Βασικό στοιχείο στην ανάλυση δεδομένων είναι οι γλώσσες υποβολής ερωτημάτων οι οποίες μπορούν να θέτουν υψηλού επιπέδου ερωτήματα στην βάση με αποτελεσματικό τρόπο. Όπως έχει αποδείξει η SQL - μία γλώσσα ερωτημάτων που χρησιμοποιείται ευρέως στις σχεσιακές βάσεις δεδομένων - οι γλώσσες υποβολής ερωτημάτων είναι αναγκαίες στην αποτελεσματική διαχείριση δεδομένων. Με την δημοτικότητα που έχουν κερδίσει τα τελευταία χρόνια οι βάσεις δεδομένων γράφων είναι προφανής η ανάγκη για μία γλώσσα υποβολής ερωτημάτων γράφων η οποία θα έχει ένα απλό συντακτικό παρόμοιο με αυτό της SQL και θα μπορεί να διαχειρίζεται αποτελεσματικά συνδεδεμένα δεδομένα σε μορφή γράφου [18].

Υπάρχουν αρκετές γλώσσες υποβολής ερωτημάτων που έχουν προταθεί τα τελευταία χρόνια για την διαχείριση δεδομένων γράφων. Μία από τις πρώτες προτάσεις ήταν η SPARQL - η καθιερωμένη γλώσσα ερωτημάτων για δεδομένα που ακολουθούν το RDF μοντέλο και χρησιμοποιείται από αρκετές βάσεις δεδομένων RDF γράφων όπως είναι η AllegroGraph και η RDF βάση της Oracle [19]. Όσον αφορά το property graph μοντέλο, ίσως η πιο δημοφιλής εφαρμογή του είναι η Neo4j, η οποία χρησιμοποιεί την Cypher ως γλώσσα υποβολής ερωτημάτων. Μια ακόμα γλώσσα ερωτήσεων που προορίζεται για το property

graph μοντέλο είναι η Gremlin, η οποία δημιουργήθηκε το 2009 από το Apache TinkerPop project. Οι γλώσσες αυτές παρουσιάζουν αρκετές διαφορές στο συντακτικό ή την μεθοδολογία υλοποίησης, ωστόσο μοιράζονται δύο βασικά χαρακτηριστικά τα οποία αποτελούν και τις δύο βασικές προσεγγίσεις όσον αφορά τον τρόπο επικοινωνίας του χρήστη με τον γράφο: την αναζήτηση μοτίβων σε έναν γράφο και την διάσχιση του γράφου (graph traversal).

Στην πρώτη περίπτωση, ο χρήστης θέτει ερωτήματα στην βάση για δεδομένα τα οποία ταιριάζουν σε ένα συγκεκριμένο μοτίβο. Στην Εικόνα 9 φαίνεται ένα παράδειγμα ενός απλού μοτίβου.



Εικόνα 9: Ένα απλό μοτίβο γράφου

Το μοτίβο αυτό περιγράφει τις σχέσεις φιλίας μεταξύ τριών οντοτήτων και η αντιστοίχισή του στην γλώσσα Cypher έχει ως εξής:

```
(a)-[:FRIEND_OF]->(b)-[:FRIEND_OF]->(c)-[:FRIEND_OF]->(a)
```

Σε αυτό το μοτίβο περιγράφεται ένα μονοπάτι το οποίο συνδέει τις οντότητες a, b και c μεταξύ τους. Γενικά τα μοτίβα έχουν την ίδια μορφή με τον γράφο της βάσης στην οποία θέτουν τα ερωτήματα και σχηματίζουν έναν υπογράφο της βάσης ο οποίος στις θέσεις των κορυφών ή των άκρων μπορεί να περιέχει μεταβλητές αντί για σταθερές [20]. Στο συγκεκριμένο παράδειγμα οι μεταβλητές είναι τα `a`, `b` και `c` και αποτελούν τα αναγνωριστικά τα οποία επιτρέπουν την αναφορά στην εκάστοτε κορυφή παραπάνω από μία φορά σε ένα μοτίβο.

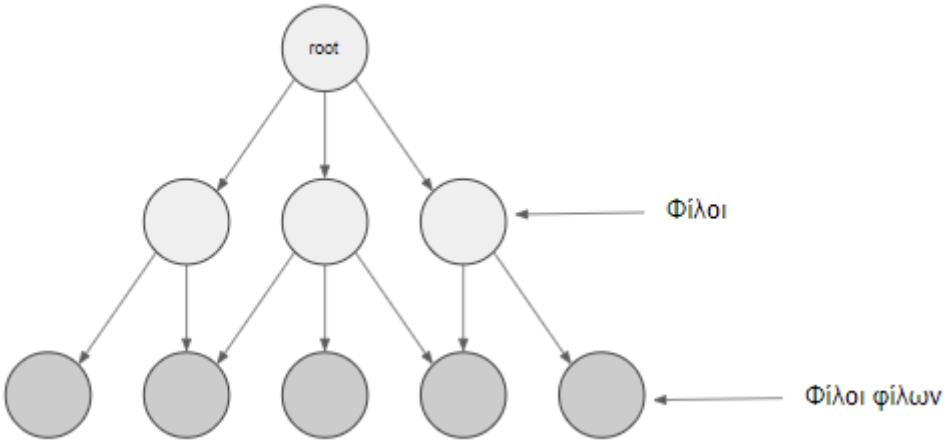
Η αναζήτηση μοτίβων μας επιτρέπει να θέτουμε ερωτήματα στην βάση με έναν οριοθετημένο τρόπο, ωστόσο είναι επίσης αναγκαίος ένας πιο ευέλικτος τρόπος επικοινωνίας με τον γράφο. Η διάσχιση του γράφου (graph traversal) είναι η σειριακή επίσκεψη στα στοιχεία του γράφου από κορυφή σε άκρη σε κορυφή κ.ο.κ. Κατά την διάρκεια της επίσκεψής αυτής, κομμάτια του γράφου μπορούν να αλλάξουν να αποθηκευτούν ή να διαγραφούν ενώ το γεγονός ότι κάθε κορυφή του γράφου διατηρεί άμεσες σχέσεις με τα γειτονικά της στοιχεία, κάνει την διάβαση αρκετά γρήγορη και αποτελεσματική. Ένα πρόβλημα το οποίο επιλύεται με την διάσχιση του γράφου είναι το πρόβλημα "friends-of-friends", ζητούμενο του οποίου είναι σε έναν γράφο όπου οντότητες ενώνονται μεταξύ τους με σχέσεις φιλίας, να βρεθούν οι φίλοι των φίλων μιας αρχικής κορυφής. Η απάντηση του προβλήματος αυτού - πάλι με την χρήση της Cypher - είναι η εξής:

```
MATCH (root)-[:FRIEND_OF*2]->(friendOfaFriend)
RETURN friendOfaFriend
```

Το παραπάνω ερώτημα δηλώνει τα εξής:

1. Ξεκίνα από την αρχική κορυφή.
2. Διέσχισε όλες τις άκρες τύπου `FRIEND_OF` που εξέρχονται από την αρχική κορυφή.
3. Για κάθε άκρη τύπου `FRIEND_OF` που διασχίστηκε, επισκέψου την κορυφή στην οποία καταλήγει.
4. Για όλους τους φίλους της αρχικής κορυφής, διέσχισε τις εξερχόμενες άκρες τύπου `FRIEND_OF`.
5. Για κάθε άκρη τύπου `FRIEND_OF` που διασχίστηκε, επισκέψου την κορυφή στην οποία καταλήγει.
6. Επέστρεψε τις κορυφές.

Οι κορυφές που θα επιστραφούν είναι και οι κορυφές που αποτελούν τους φίλους των φίλων της αρχικής κορυφής. Στην Εικόνα 10 φαίνονται τα μονοπάτια τα οποία θα ακολουθήσει ο αλγόριθμος διάβασης ενώ οι κορυφές με γκρίζο χρώμα είναι οι κορυφές που θα επιστραφούν από το ερώτημα.



Εικόνα 10: Το πρόβλημα “friend-of-a-friend”

Τα ερωτήματα αυτά ονομάζονται ερωτήματα τροχιάς (path queries) και μας επιτρέπουν να διασχίσουμε τον γράφο χρησιμοποιώντας μονοπάτια αυθαίρετου μήκους [20].

3.2 Neo4j

Η Neo4j είναι ένα σύστημα διαχείρισης βάσεων δεδομένων γράφων η οποία ακολουθεί το μοντέλο property graph. Γενικά όσον αφορά τις βάσεις δεδομένων γράφων, υπάρχουν δύο βασικές ιδιότητες που πρέπει να λάβουμε υπόψη: τον τρόπο αποθήκευσης και τον τρόπο επεξεργασίας ενός γράφου. Κάποιες υλοποιήσεις των βάσεων δεδομένων γράφων αποθηκεύουν τα δεδομένα σε σχεσιακές βάσεις ή άλλες βάσεις γενικού σκοπού. Αντίθετα η Neo4j υποστηρίζει κανονική αποθήκευση γράφων (*native graph storage*) το οποίο έχει ως αποτέλεσμα την υψηλή αποδοτικότητα στα ερωτήματα τροχιάς. Στην Neo4j τα δεδομένα αποθηκεύονται σε ξεχωριστά αρχεία. Κάθε αρχείο περιέχει πληροφορίες για συγκεκριμένα τμήματα του γράφου (κόμβους, σχέσεις και ιδιότητες). Ο διαχωρισμός της δομής του γράφου από τις ιδιότητες οδηγεί σε αποτελεσματικά και γρήγορα ερωτήματα τροχιάς, παρόλο που η όψη που έχει ένας χρήστης για έναν γράφο διαφέρει από την πραγματική όψη του γράφου στην φυσική μνήμη [21].

Κάποιες βάσεις δεδομένων γράφων χρησιμοποιούν δείκτες για την σύνδεση των κόμβων, κάτι το οποίο αυξάνει το υπολογιστικό κόστος για κάθε διάσχιση. Αντίθετα η Neo4j υποστηρίζει κανονική επεξεργασία γράφου (*native graph processing*). Μια βάση δεδομένων γράφων λέμε ότι υποστηρίζει κανονική επεξεργασία γράφου όταν έχει μία ιδιότητα που ονομάζεται γειτνίαση χωρίς δείκτη (*index-free adjacency*). Αυτό σημαίνει ότι οι κόμβοι διατηρούν άμεσες αναφορές προς τους γειτονικούς τους κόμβους στην φυσική μνήμη. Έτσι κάθε κόμβος λειτουργεί ως ένας μικρός δείκτης για τους γειτονικούς κόμβους, και οι χρόνοι των ερωτημάτων είναι ανεξάρτητοι από το συνολικό μέγεθος του γράφου [21].

3.2.1 Cypher

Η Cypher είναι μία δηλωτική γλώσσα υποβολής ερωτημάτων για το property graph μοντέλο, και είναι η βασική γλώσσα που υποστηρίζει η Neo4j. Παρέχει δυνατότητες τόσο για την υποβολή ερωτημάτων και την διαχείριση των δεδομένων σε ένα property graph, όσο και για την τροποποίηση της δομής ενός γράφου. Κάθε ερώτημα στην Cypher έχει ως είσοδο ένα property graph και ως έξοδο ένα property graph ή έναν πίνακα. Τα ερωτήματα στη Cypher είναι δομημένα σειριακά, το οποίο σημαίνει ότι οι συναρτήσεις που περιέχει ένα ερώτημα εκτελούνται με την σειρά από τα αριστερά προς τα δεξιά. Έτσι για παράδειγμα, αντί η περιγραφή των δεδομένων που θα επιστραφούν να γίνει στην αρχή του ερωτήματος όπως στην SQL με το `SELECT`, στην Cypher γίνεται στο τέλος του ερωτήματος με το `RETURN` [22]. Αυτή η γραμμικότητα επιτρέπει την ένωση πολλαπλών ερωτημάτων μέσω του όρου `WITH`, όπου η έξοδος ενός ερωτήματος δρα ως είσοδος για ένα άλλο.

Η βασική ιδέα πίσω από την Cypher είναι η αναζήτηση μοτίβων. Τα μοτίβα στην Cypher αναπαρίστανται ως εξής: `(α)-[β]->(γ)`, όπου τα α και γ στις παρενθέσεις αναπαριστούν κόμβους, το β αναπαριστά την σχέση, ενώ το βέλος είναι προαιρετικό και δηλώνει την κατεύθυνση της σχέσης. Με τη βοήθεια του όρου `MATCH` η Cypher αναζητεί τέτοια μοτίβα στον γράφο. Υπάρχουν επιπλέων όροι για την αλλαγή της δομής του γράφου οι οποίοι επίσης ακολουθούν την γραμμική ροή της Cypher. Οι βασικοί όροι αλλαγής της δομής του γράφου είναι το `CREATE` το οποίο χρησιμοποιείται για την δημιουργία νέων κόμβων ή σχέσεων, το `DELETE` για την διαγραφή κόμβων ή σχέσεων και το `SET` για την αλλαγή των ιδιοτήτων κόμβων ή σχέσεων. Επιπλέον, υπάρχει ο όρος `MERGE` ο οποίος αναζητεί μοτίβα στον γράφο όπως το `MATCH`, με την διαφορά ότι στην περίπτωση που το μοτίβο δεν υπάρχει, το δημιουργεί [22].

4 Μελέτη περίπτωσης

4.1 Ανάλυση Προβλήματος

Το πεδίο εφαρμογής στο οποίο εστιάζει η παρούσα πτυχιακή εργασία είναι ο κλάδος του τουρισμού και η ανάπτυξη μιας πιλοτικής διαδικτυακής εφαρμογής η οποία διαχειρίζεται κοινωνικά δεδομένα συναφή με τον τομέα του τουρισμού. Ειδικότερα, ο στόχος της εφαρμογής είναι η δημιουργία ενός ψηφιακού κοινωνικού δικτύου μέσω του οποίου χρήστες θα μπορούν να συνδέονται και να αλληλεπιδρούν μεταξύ τους στα πλαίσια της αναζήτησης νέων τοποθεσιών προς επίσκεψη - όπως πόλεις, χώρες ή εστιατόρια. Αναλυτικότερα οι χρήστες της εφαρμογής θα είναι σε θέση να εκτελέσουν τις εξής λειτουργίες:

- Εγγραφή και σύνδεση με όνομα χρήστη και κωδικό μέσω ενός συστήματος αυθεντικοποίησης.
- Κάθε χρήστης μπορεί να ακολουθεί άλλους χρήστες.
- Κάθε χρήστης μπορεί να δημιουργεί posts.
- Κάθε χρήστης μπορεί να αλληλεπιδρά με κάθε post δηλώνοντας αρέσκεια, κάνοντας repost ή αφήνοντας σχόλια.
- Σε κάθε post οι χρήστες μπορούν να αναφέρουν άλλους χρήστες καθώς και να δηλώσουν tags.
- Κάθε χρήστης μπορεί να αναζητά μέρη.
- Για κάθε μέρος ο χρήστης μπορεί να δηλώσει ότι το έχει επισκεφτεί, ότι ενδιαφέρεται να το επισκεφτεί ή να το βαθμολογήσει.

Επιπλέον, η εφαρμογή θα υποστηρίζει τη δημιουργία ενός προσωπικού προφίλ για κάθε χρήστη το οποίο θα περιέχει τις βασικές του πληροφορίες, τα άτομα που ακολουθεί, τα μέρη που έχει επισκεφτεί καθώς και τα post που έχει δημιουργήσει σε χρονολογική σειρά.

4.2 Τεχνολογικά εργαλεία

4.2.1 Node.js

Η Node είναι μια πλατφόρμα φτιαγμένη στο JavaScript runtime του Chrome και χρησιμοποιείται για την εύκολη δημιουργία γρήγορων, κλιμακωτών εφαρμογών δικτύων. Βασικό χαρακτηριστικό της είναι ότι αντίθετα με άλλες πλατφόρμες ανάπτυξης διαδικτυακών εφαρμογών που στηρίζονται στην πολυνηματική επεξεργασία, η Node χρησιμοποιεί ένα non-blocking I/O μοντέλο βασισμένο σε γεγονότα (events). Αυτό την καθιστά εξαιρετικά αποτελεσματική για real-time εφαρμογές μεγάλης κλίμακας οι οποίες εκτείνονται σε πολλές κατανεμημένες συσκευές. Κάθε λειτουργία στην Node εκτελείται ασύγχρονα, δηλαδή εμποδίζει την εφαρμογή από το να σταματάει κάθε φορά που περιμένει την εκτέλεση μιας λειτουργίας εισόδου/εξόδου.

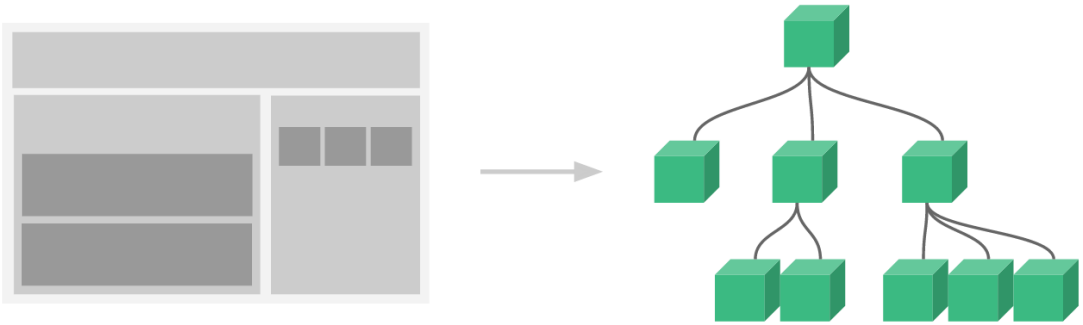
4.2.2 Vue.js

Το Vue είναι ένα framework γραμμένο σε Javascript, για την δημιουργία διεπαφών και single-page applications (*SPA*). Αποτελεί ένα JavaScript εργαλείο το οποίο επιτρέπει την δημιουργία πλούσιων και διαδραστικών front-end εφαρμογών. Εμπεριέχει αρκετές λειτουργίες οι οποίες καθιστούν εύκολη την ανάπτυξη πλήρως λειτουργικών διαδικτυακών εφαρμογών όπως ο χειρισμός πολύπλοκων δεδομένων και η εμφάνισή τους στον client, ή μηχανισμούς routing στην πλευρά του client, χωρίς να υπάρχει ανάγκη σύνδεσης με κάποιον server. Τα βασικά χαρακτηριστικά του Vue είναι τα εξής:

- **Templates (*πρότυπα*)**: Το Vue παρέχει έναν δυνατό συντακτικό βασισμένο σε templates, το οποίο μας δίνει την δυνατότητα να αποδώσουμε δηλωτικά τα δεδομένα στο DOM (*Document Object Model*) και να χειριστούμε events. Σε ένα template το απλό HTML συντακτικό εμπλουτίζεται με ειδικές ιδιότητες που ονομάζονται directives οι οποίες λένε στην εφαρμογή πώς πρέπει να διαχειριστεί ή να εμφανίσει τα δεδομένα.

- **Reactivity (*διαδραστικότητα*)**: Το βασικό στοιχείο που κάνει το Vue τόσο δυνατό εργαλείο είναι η διαδραστικότητα. Πέρα από την εμφάνιση των δεδομένων μέσω των templates, το Vue παρέχει επίσης έναν μηχανισμό δέσμευσης δεδομένων (*data binding*) ο οποίος "παρακολουθεί" τα δεδομένα για τυχόν αλλαγές και ενημερώνει το DOM ανάλογα, σε περίπτωση που αυτά αλλάξουν.

- **Components**: Τα components είναι αυτοτελή κομμάτια κώδικα που αντιπροσωπεύουν συγκεκριμένα μέρη της σελίδας (βλ. Εικόνα 11). Κάθε component επεξεργάζεται τα δικά του δεδομένα, εμπεριέχει τη δική του JavaScript και τη δική του CSS μορφοποίηση. Τα components συχνά περιέχουν άλλα υπο-components, τα οποία μπορούν να επικοινωνούν μεταξύ τους ανταλλάσοντας δεδομένα ή εκπέμποντας events. Το βασικό πλεονέκτημα του διαχωρισμού του κώδικα σε διαφορετικά components είναι το ότι ο κώδικας που είναι υπεύθυνος για ένα συγκεκριμένο κομμάτι της σελίδας είναι αποξενωμένος από τον υπόλοιπο κώδικα της εφαρμογής το οποίο κάνει την ανάπτυξη και την συντήρησή του αρκετά ευκολότερη και πιο γρήγορη.



Εικόνα 11: VueJS components

4.2.3 NPM

Το NPM (*Node Package Manager*) είναι ένα σύστημα διαχείρισης πακέτων το οποίο έρχεται προεγκατεστημένο με την Node. Το NPM θα βοηθήσει στην εύκολη εγκατάσταση πακέτων και plugins - τόσο για το front-end όσο και για το back-end - τα οποία έχουν αναπτυχθεί από τις κοινότητες προγραμματιστών της Node και του VueJS.

4.2.4 APIs

Ένα API (*Application Program Interface*) είναι ένα σύνολο από ρουτίνες, πρωτόκολλα και εργαλεία το οποίο καθορίζει το πως δύο κομμάτια λογισμικού επικοινωνούν μεταξύ τους. Πιο συγκεκριμένα, ένα δημόσιο API είναι ένα API το οποίο επιτρέπει την πρόσβαση των προγραμματιστών στα δεδομένα μιας διαδικτυακής υπηρεσίας ή εφαρμογής. Δεν είναι η βάση δεδομένων ή ο server, αλλά ο κώδικας που καθορίζει τα σημεία πρόσβασης των χρηστών στα δεδομένα της εφαρμογής. Για τις ανάγκες της πτυχιακής εργασίας, τα δημόσια APIs που χρησιμοποιήθηκαν είναι τα εξής:

- **Google Places API**: Το Places API της Google είναι μια υπηρεσία η οποία επιστρέφει πληροφορίες σχετικά με μέρη μέσω αιτημάτων HTTP. Σε αυτο το API τα μέρη ορίζονται ως γεωγραφικές τοποθεσίες (χώρες, πόλεις κλπ.), εγκαταστάσεις (ξενοδοχεία, εστιατόρια, υπηρεσίες, κλπ.), και διάφορα σημεία ενδιαφέροντος (αρχαιολογικοί χώροι, μουσεία, κλπ.).

- **Google Maps Static API**: Το Maps Static API επιτρέπει την ενσωμάτωση στατικών εικόνων Google Maps σε διαδικτυακές εφαρμογές. Το API δημιουργεί έναν χάρτη μέσω URL παραμέτρων οι οποίες στέλνονται μέσω ενός τυπικού αιτήματος HTTP και επιστρέφει τον χάρτη ως εικόνα.

4.2.5 Git/Github

Το Git είναι ένα κατανεμημένο σύστημα ελέγχου εκδόσεων (version-control system) σχεδιασμένο για την παρακολούθηση των αλλαγών του κώδικα κατά την διάρκεια ανάπτυξης μιας εφαρμογής. Έχει σχεδιαστεί για το συντονισμό των εργασιών μεταξύ προγραμματιστών, αλλά μπορεί να χρησιμοποιηθεί για την παρακολούθηση αλλαγών σε οποιοδήποτε σύνολο αρχείων.

Το Github είναι μια κερδοσκοπική εταιρεία που προσφέρει υπηρεσίες φιλοξενίας κώδικα χρησιμοποιώντας το version-control σύστημα git και καθιστά την εύκολη συνεργασία μεταξύ ατόμων ή ομάδων για την ανάπτυξη λογισμικού. Το Github χρησιμοποιήθηκε ως ένα αποθετήριο για την φιλοξενία του κώδικα της εφαρμογής που δημιουργήθηκε στα πλαίσια της πτυχιακής εργασίας [23].

4.2.6 Πακέτα και plugins - Front-End

- **Vuetify**: Το Vuetify είναι ένα σημασιολογικό framework για το VueJS. Παρέχει έτοιμα, σημασιολογικά και επαναχρησιμοποιούμενα Vue components βασισμένα στο Material Design της Google, με στόχο την εύκολη και γρήγορη ανάπτυξη όμορφων διεπαφών.

- **Vuex**: Το Vuex αποτελεί ένα σχεδιαστικό πρότυπο και μία βιβλιοθήκη για VueJS εφαρμογές. Λειτουργεί ως ένας κεντρικός χώρος αποθήκευσης όλων των components

μιας VueJS εφαρμογής και περιέχει κανόνες που εξασφαλίζουν ότι η κατάσταση της εφαρμογής μπορεί να μεταλλαχθεί μόνο με έναν προβλέψιμο τρόπο.

- **Axios**: Το Axios είναι μια βιβλιοθήκη η οποία επιτρέπει την δημιουργία ασύγχρονων αιτημάτων HTTP σε ένα API για την εκτέλεση CRUD (Create, Read, Update, Delete) λειτουργιών.

4.2.7 Πακέτα και plugins - Backend

- **Express.js**: Ισχυρό πλεονέκτημα της Node είναι ότι δεν έχει έναν προκαθορισμένο τρόπο χρήσης για την δημιουργία ενός server παρέχοντας έτσι την ευελιξία στον χρήστη στο πως θα δημιουργήσει έναν server. Ωστόσο στην δημιουργία ιστοσελίδων και διαδικτυακών εφαρμογών υπάρχουν κάποια βασικά καθήκοντα που πρέπει να εκτελούνται κάθε φορά. Το Express είναι ένα ευέλικτο Node framework σχεδιασμένο να εκτελεί τα καθήκοντα αυτά με έναν καλά δοκιμασμένο και επαναλαμβανόμενο τρόπο.

- **node-Neo4j**: Το node-Neo4j είναι ένας JavaScript οδηγός ο οποίος αποτελεί τον βασικό τρόπο επικοινωνίας του Node server με την Neo4j βάση.

- **Passport.js**: Το Passport είναι μια ευέλικτη Node βιβλιοθήκη η οποία μπορεί να χρησιμοποιηθεί σε οποιαδήποτε Express εφαρμογη με στόχο την πιστοποίηση χρηστών είτε μέσω όνομα και κωδικό χρήστη, είτε μέσω διαδικτυακών πλατφόρμων όπως το Facebook ή το Twitter.

4.3 Σχεδιασμός υλοποίησης

Αξιοποιώντας τα παραπάνω εργαλεία δημιουργήθηκε μία σύγχρονη διαδικτυακή εφαρμογή η οποία χωρίζεται σε τρία βασικά κομμάτια: (α) την Neo4j βάση δεδομένων στην οποία αποθηκεύονται όλα τα δεδομένα, (β) το back-end (server) κομμάτι το οποίο είναι υπεύθυνο για την επικοινωνία με την βάση και την δημιουργία ενός API για την επικοινωνία με τον client, και (γ) το front-end (client) κομμάτι το οποίο είναι υπεύθυνο για την ανάκτηση των δεδομένων της Neo4j βάσης μέσω του back-end API και την παρουσίασή τους στον χρήστη.

Πρωταρχικός στόχος είναι η δημιουργία της βάσης δεδομένων κατά το πρότυπο του property graph. Με άλλα λόγια θα πρέπει να προσδιοριστούν ποιες οντότητες είναι αναγκαίο να υπάρχουν στο property graph, ποιες θα είναι οι σχέσεις που τις συνδέουν, και τέλος να δημιουργηθούν τα κατάλληλα ερωτήματα στην βάση έτσι ώστε να καλυφθούν οι ανάγκες της εφαρμογής που αναφέρθηκαν στην ενότητα 4.1. Στην συνέχεια θα υλοποιηθεί το back-end κομμάτι σε περιβάλλον Node.js. Με την βοήθεια της βιβλιοθήκης Express.js θα δημιουργηθεί ένα API το οποίο θα εκτελεί CRUD (Create, Read, Update, Delete) λειτουργίες στην Neo4j βάση. Η ιδέα είναι ότι όταν ο client κάνει ένα HTTP request σε ένα API endpoint του server, εκτελείται ένα συγκεκριμένο ερώτημα στην βάση και επιστρέφονται στον client τα κατάλληλα δεδομένα. Για παράδειγμα, αν ο client κάνει ένα POST request στο endpoint `http://[SERVER]/user`, δημιουργείται ένας νέος χρήστης στην βάση. Τέλος θα υλοποιηθεί το κομμάτι του front-end με την χρήση του MVC framework Vue.js. Το front-end

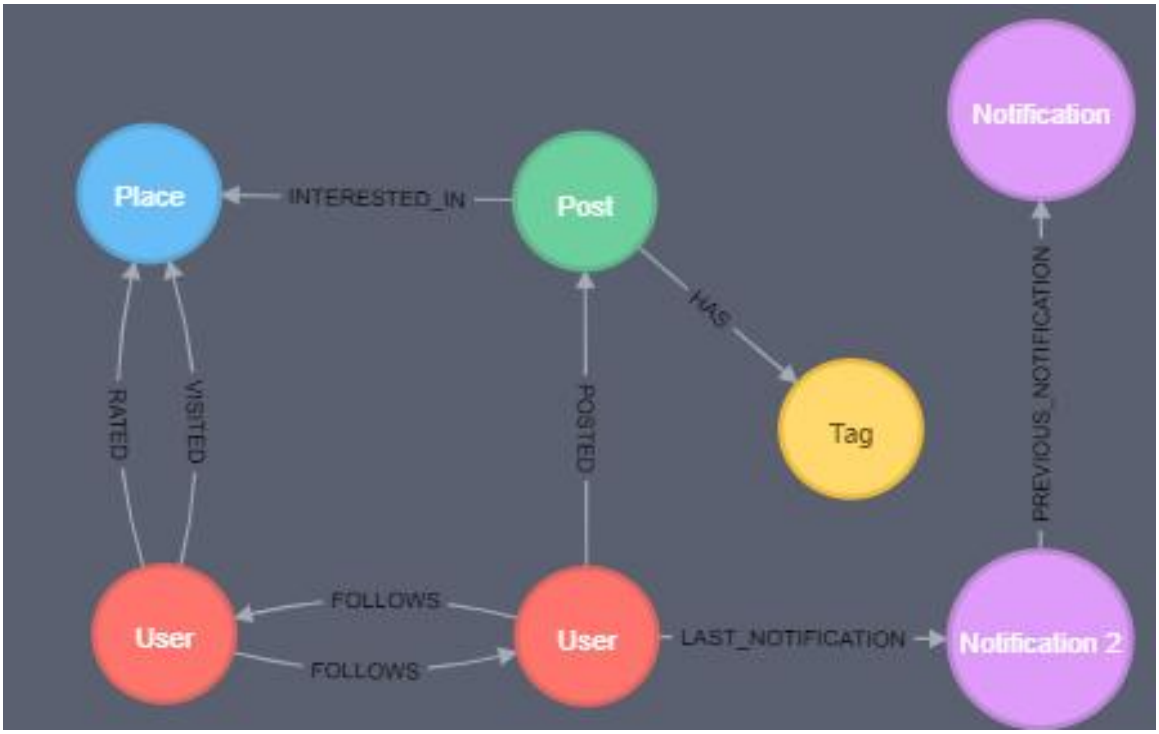
κομμάτι θα χρησιμοποιεί το back-end API που δημιουργήθηκε για την ανάκτηση δεδομένων από την Neo4j βάση και θα τα παρουσιάζει στον χρήστη μέσω component σχεδιασμένα με βάση το material design της Google.

4.4 Το property graph με τα ενδεικτικά δεδομένα

Για την υλοποίηση του μοντέλου δεδομένων στην Neo4j, το πρώτο πράγμα που πρέπει να προσδιοριστεί είναι οι κόμβοι. Οι κόμβοι συνήθως αντιπροσωπεύουν οντότητες, αλλά μπορούν επίσης να αντιπροσωπεύουν και άλλα στοιχεία, ανάλογα με την περίπτωση χρήσης. Για το συγκεκριμένο σενάριο το property graph που υλοποιήθηκε περιέχει τους εξής κόμβους:

- User: Οι χρήστες της εφαρμογής
- Post: Τα post των χρηστών
- Place: Τα μέρη που μας δίνει το Google Places API
- Tag: Τα tags που δημιουργούν οι χρήστες
- Notification: Οι ενημερώσεις που λαμβάνει ο χρήστης

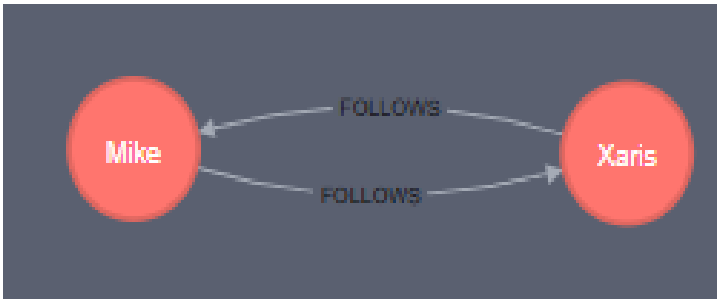
Το τελικό μοντέλο δεδομένων με όλους τους κόμβους και τις σχέσεις που τους συνδέουν φαίνεται στην Εικόνα 12.



Εικόνα 12 : Μοντέλο δεδομένων

4.3.1 User domain model

Η βασική οντότητα της εφαρμογής είναι οι χρήστες (βλ. Εικόνα 13). Η πρώτη λειτουργία που μπορεί να εκτελέσει ο κάθε χρήστης είναι η δημιουργία νέου λογαριασμού στην εφαρμογή. Κατά την δημιουργία λογαριασμού εκτελείται το ερώτημα στην Εικόνα 14, το οποίο δημιουργεί έναν νέο κόμβο τύπου `User`.



Εικόνα 13: Users domain model

```
1 CREATE (user:User {data})
2 RETURN user
```

Εικόνα 14: Δημιουργία χρήστη

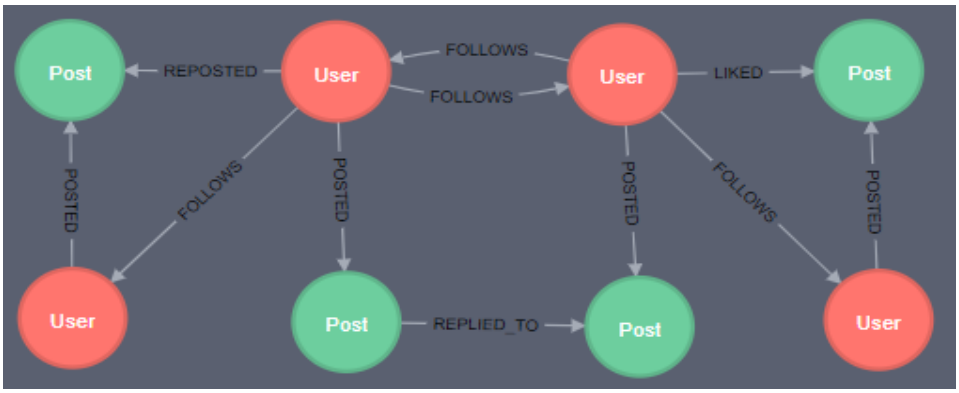
Η μεταβλητή data, είναι ένα JSON ερχείο το οποίο περιέχει τις βασικές ιδιότητες του χρήστη όπως είναι το όνομα, το επίθετο, το όνομα χρήστη και ο κωδικός. Το όνομα χρήστη λειτουργεί και ως μοναδικό αναγνωριστικό για την μελλοντική ανάκτηση των δεδομένων του. Μετά την εγγραφή του, ο χρήστης μπορεί να ακολουθήσει άλλους χρήστες της εφαρμογής. Όταν ο χρήστης θέλει να ακολουθήσει έναν άλλο χρήστη εκτελείται το ερώτημα στην Εικόνα 15 το οποίο δημιουργεί την κατευθυνόμενη σχέση `:FOLLOWS` μεταξύ δύο χρηστών.

```
1 MATCH (user:User {username: {user}}), (other:User {username: {otherUser}})
2 MERGE (user)-[r:FOLLOWS]->(other)
3 ON CREATE SET r.timestamp = timestamp()
4 RETURN r
```

Εικόνα 15: Δημιουργία σχέσης :FOLLOWS μεταξύ δύο χρηστών

4.3.2 Posts domain model

Τα post αποτελούν τον βασικό μηχανισμό με τον οποίο οι χρήστες της εφαρμογής μπορούν να αλληλεπιδράσουν μεταξύ τους (βλ. Εικόνα 16). Ο χρήστης έχει την δυνατότητα να δημιουργήσει ένα νέο post και να αλληλεπιδράσει με post χρηστών που ακολουθεί δηλώνοντας αρέσκεια, σχολιάζοντας ή κοινοποιώντας το post στο προσωπικό του προφίλ. Η δημιουργία του κάθε post επιτυγχάνεται με το ερώτημα της Εικόνα 17.



Εικόνα 16: Posts domain model

```
1 MATCH (poster:User {username: {username}})
2 CREATE (poster)-[:POSTED {timestamp: timestamp()}]->(post:Post {content: {content}})
```

Εικόνα 17: Δημιουργία νέου post

Αυτό το ερώτημα δημιουργεί έναν νέο κόμβο τύπου `Post` και μία νέα σχέση που συνδέει το post με έναν χρήστη. Η μεταβλητή `username` περιέχει το όνομα του χρήστη που δημιουργεί το post, και η μεταβλητή `content` το περιεχόμενο του post. Για την αποτελεσματική ανάκτηση των δεδομένων των post από την εφαρμογή, είναι επίσης αναγκαίο και ένα μοναδικό αναγνωριστικό. Η Neo4j παρέχει ένα μοναδικό ID για κάθε κόμβο και σχέση του γράφου, ωστόσο τα IDs αυτά επαναχρησιμοποιούνται από την Neo4j σε περίπτωση που κάποιος κόμβος ή κάποια σχέση διαγραφούν. Αυτό σημαίνει ότι οι εφαρμογές που χρησιμοποιούν και βασίζονται σε εσωτερικά IDs της Neo4j είναι εύθραυστες και κινδυνεύουν να κάνουν λάθη. Συνεπώς, συνιστάται η χρήση IDs που έχουν δημιουργηθεί από τον χρήστη.

```
1 MERGE (id:UniqueId {name: "Post"})
2 ON CREATE SET id.count = 1
3 ON MATCH SET id.count = id.count + 1
```

Εικόνα 18: Δημιουργία μοναδικού αναγνωριστικού

Το ερώτημα στην Εικόνα 18 δημιουργεί με την βοήθεια του όρου `MERGE` έναν βοηθητικό κόμβο τύπου `UniqueId` ο οποίος περιέχει ως χαρακτηριστικό έναν μετρητή που αυξάνεται κάθε φορά που εκτελείται το ερώτημα. Ο συνδυασμός των ερωτημάτων των εικόνων 17 και 18 με τον όρο `WITH`, έχει ως αποτέλεσμα την δημιουργία του ερωτήματος στην **Εικόνα *19*** το οποίο αποτελεί και το τελικό ερώτημα δημιουργίας ενός post με μοναδικό αναγνωριστικό.

```
1 MERGE (id:UniqueId {name: "Post"})
2 ON CREATE SET id.count = 1
3 ON MATCH SET id.count = id.count + 1
4 WITH id.count AS uid
5 CREATE (poster)-[:POSTED {timestamp: timestamp()}]->(post:Post {id: uid, content: {content}})
6 RETURN post, poster
```

Εικόνα 19: Δημιουργία post με μοναδικό αναγνωριστικό

Η επόμενη ενέργεια που σχετίζεται με τα post είναι η δημιουργία σχολίων. Κάθε σχόλιο θεωρείται ως ένα post το οποίο συνδέεται με την σχέση :REPLIED_TO με ένα άλλο post όπως φαίνεται στην Εικόνα 18. Το ερώτημα δημιουργίας σχολίων είναι παρόμοιο με το ερώτημα δημιουργίας post με την διαφορά ότι είναι αναγκαία η δημιουργία περισσότερων σχέσεων (βλ. Εικόνα 20).

```
1  MERGE (id:UniqueId {name: "Post"})
2  ON CREATE SET id.count = 1
3  ON MATCH SET id.count = id.count + 1
4  WITH id.count AS uid
5
6  MATCH (user:User {username: {username}})
7  MATCH (post:Post {id: {postId}})<-[:POSTED]-(poster)
8  CREATE (reply:Post {id: uid, content: {reply}})
9  CREATE (user)-[:POSTED {timestamp: timestamp()}]->(reply)
10 CREATE (reply)-[:REPLIED_TO {timestamp: timestamp()}]->(post)
```

Εικόνα 20: Δημιουργία σχολίου με μοναδικό αναγνωριστικό

Όσον αφορά τα ερωτήματα δήλωσης αρέσκειας και την κοινοποίηση των post, τα ερωτήματα είναι σχετικά απλά και αποτελούνται από την δημιουργία μιας σχέσης μεταξύ ενός χρήστη και ενός post, όμοια με το ερώτημα δημιουργίας νέου post (βλ. Εικόνα 17).

4.3.3 Tags domain model

Τα tags λειτουργούν ως ένας μηχανισμός κατηγοριοποίησης των post (βλ. Εικόνα 21). Κάθε φορά που ο χρήστης δημιουργεί ένα post, επιλέγει αν θα προσδιορίσει ένα ή περισσότερα tags τα οποία συσχετίζονται με το post που δημιουργείται. Είναι λοιπόν προφανές ότι η δημιουργία των tags είναι απαραίτητο να εκτελεστεί ταυτόχρονα με την δημιουργία του post. Επεκτείνοντας το ερώτημα δημιουργίας νέου post στην Εικόνα 19, έχουμε ως αποτέλεσμα το ερώτημα δημιουργίας post με tags στην Εικόνα 22. Στο συγκεκριμένο ερώτημα η μεταβλητή `tags` αποτελείται από έναν πίνακα ο οποίος περιέχει τα ονόματα των tags που σχετίζονται με το συγκεκριμένο post. Με την χρήση όρου `FOREACH` που παρέχει η Neo4j, ο οποίος χρησιμοποιείται για την ενημέρωση ή την δημιουργία δεδομένων που βρίσκονται σε μορφή πίνακα, μπορούμε να ανατρέξουμε σε όλα τα στοιχεία του πίνακα `tags` και να δημιουργήσουμε τους απαραίτητους κόμβους για κάθε ένα από αυτά.



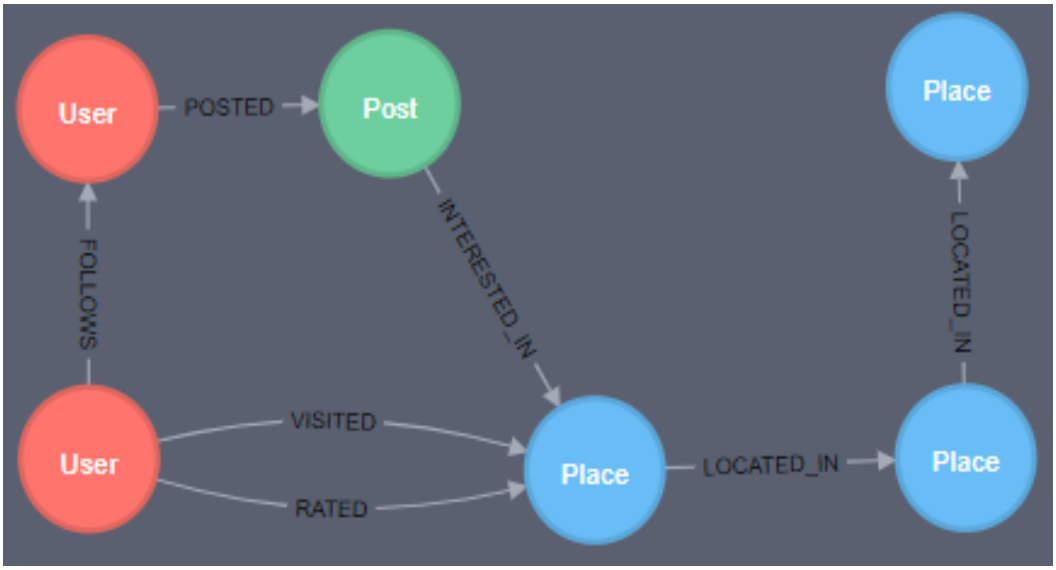
Εικόνα 21: Tags domain model

```
MERGE (id:UniqueId {name: "Post"})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
WITH id.count AS uid
MATCH (poster:User {username: {username}})
CREATE (post:Post {id: uid, content: {content}})
CREATE (poster)-[:POSTED {timestamp: timestamp()}]->(post)
WITH post, poster
UNWIND {tags} AS tags
FOREACH (b IN CASE WHEN tags IS NULL THEN [] ELSE [tags] END |
            MERGE (tag:Tag {name: tags})
            CREATE (post)-[:HAS]->(tag))
RETURN mentions, post, poster
```

Εικόνα 22: Δημιουργία post με tags

4.3.4 Places domain model

Τα δεδομένα των τοποθεσιών προέρχονται από το Places API της Google, και εισέρχονται στην Neo4j βάση εφόσον ο χρήστης διαδράσει με αυτά. Η δομή των τοποθεσιών φαίνεται στην Εικόνα 23 όπου ένα μέρος μπορεί να βρίσκεται σε ένα άλλο μέρος. Για παράδειγμα ένα ξενοδοχείο βρίσκεται σε μία πόλη η οποία με την σειρά της βρίσκεται σε μία χώρα. Ο χρήστης μπορεί να αλληλεπιδράσει με τα μέρη με τρεις τρόπους, μπορεί να δηλώσει ότι έχει επισκεφτεί ένα μέρος, μπορεί να το βαθμολογήσει με έναν αύξον αριθμό, ή μπορεί να δηλώσει ενδιαφέρον προς επίσκεψη για ένα μέρος.



Εικόνα 23: Places domain model

```
MERGE (place:Place {place_id: {placeId}})
WITH place
MATCH (user:User {username: {username}})
MERGE (user)-[:VISITED]->(place)
RETURN place, user
```

Εικόνα 24: Δημιουργία σχέσης :VISITED

Όταν ο χρήστης θέλει να δηλώσει ότι έχει επισκεφτεί ένα μέρος εκτελείται το ερώτημα στην Εικόνα 24 το οποίο δημιουργεί την σχέση :VISITED μεταξύ ενός χρήστη και μιας τοποθεσίας. Όταν ο χρήστης θέλει να βαθμολογήσει ένα μέρος εκτελείται ένα παρόμοιο ερώτημα με την διαφορά ότι η σχέση που δημιουργείται έχει το χαρακτηριστικό rating το οποίο δηλώνει την βαθμολογία που έθεσε ο χρήστης στο μέρος (βλ. Εικόνα 25).

```
MERGE (place:Place {place_id: {placeId}})
WITH place
MATCH (user:User {username: {username}})
MERGE (user)-[:RATED {rating: {rating}}]->(place)
RETURN place, user
```

Εικόνα 25: Δημιουργία σχέσης :RATED

Για την εκδήλωση ενδιαφέροντος ο χρήστης δημιουργεί ένα post με το οποίο δηλώνει ότι ενδιαφέρεται να επισκεφτεί ένα συγκεκριμένο μέρος. Η δημιουργία του post γίνεται με τον τρόπο που αναλύσαμε στην ενότητα 4.3.2 με την διαφορά ότι δημιουργείται μία επιπλέον σχέση που ενώνει το post με το μέρος (βλ. Εικόνα 26).

```
MERGE (id:UniqueId {name: "Post"})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
WITH id.count AS uid
MERGE (place:Place {place_id: {placeId}})
WITH place, uid
MATCH (user:User {username: {username}})
CREATE (post:Post {id: uid, content: {postContent}})
CREATE (user)-[:POSTED {timestamp: timestamp()}]->(post)-[:INTERESTED_IN]->(place)
RETURN place, user, post
```

Εικόνα 26: Δημιουργία ενδιαφέροντος

4.3.5 Notifications domain model

Όταν ο χρήστης δηλώσει ενδιαφέρον για ένα μέρος, τότε εκτελούνται διαδοχικά τα ερωτήματα στην Εικόνα 28 και Εικόνα 29 αντίστοιχα. Το ερώτημα της Εικόνα 28 επιστρέφει μία λίστα από φίλους του χρήστη που έχουν επισκεφθεί ένα συγκεκριμένο μέρος. Για κάθε χρήστη που επιστράφηκε, εκτελείται το ερώτημα της Εικόνα 29 το οποίο δημιουργεί ένα νέο notification για τον εκάστοτε χρήστη.



Εικόνα 27: Notifications domain model

```
MATCH (user:User {username: {username}})
MATCH (user)-[:FOLLOWS]->(followee:User)
WITH collect(followee) AS followeeList, user
MATCH (follower:User)-[:FOLLOWS]->(user)
WITH followeeList, collect(follower) AS followerList
WITH [x IN followeeList WHERE x IN followerList] AS friendsList
UNWIND friendsList AS friends
MATCH (place:Place {place_id: {placeId}})
MATCH (friends)-[:VISITED]->(place)
RETURN collect(friends.username) AS friendVisitors
```

Εικόνα 28: Αναζήτηση φίλων που έχουν επισκεφθεί ένα μέρος

```
MATCH (user:User {username: {username}})
OPTIONAL MATCH (user)-[r:LAST_NOTIFICATION]->(old:Notification)
DELETE r
CREATE (new:Notification {content: {notification}})
CREATE (user)-[:LAST_NOTIFICATION]->(new)
WITH collect(old) AS oldList, new
FOREACH (x IN oldList | CREATE (new)-[:PREVIOUS_NOTIFICATION]->(x))
RETURN new
```

Εικόνα 29: Δημιουργία notification

4.3.6 Προτάσεις φιλίας βάσει κοινών γνωστών

Τα συστήματα προτάσεων (recommendation systems) αποτελούν μία από τις βασικές περιπτώσεις χρήσης των βάσεων δεδομένων γράφων. Στο συγκεκριμένο σενάριο ένα σύστημα προτάσεων που μπορεί να δημιουργηθεί είναι οι προτάσεις φιλίας βάση κοινών γνωστών. Οι προτάσεις φιλίας γίνονται κάνοντας την υπόθεση ότι εάν δύο χρήστες ενός κοινωνικού δικτύου έχουν έναν κοινό γνωστό, τότε υπάρχει μια αυξημένη πιθανότητα αυτοί οι χρήστες να γίνουν φίλοι κάποια στιγμή στο μέλλον. Το ερώτημα της Εικόνα 30 επιστρέφει μία λίστα τριών ατόμων ταξινομημένη σύμφωνα με το πόσους κοινούς γνωστούς έχουν με έναν αρχικό χρήστη.

```
MATCH (me:User {username: {username}})-[:FOLLOWS]->(followee:User)-[:FOLLOWS]->(sug)
WHERE me <> sug
AND NOT (me)-[:FOLLOWS]->(sug)
RETURN sug.username AS username,
       sug.fname AS fname,
       sug.lname AS lname,
       count(sug) AS common
ORDER BY common DESC
LIMIT 3
```

Εικόνα 30: Σύστημα προτάσεων φιλίας βάσει κοινών γνωστών

4.3.7 Χρονολόγιο χρήστη

Ένα από τα βασικά κομμάτια των κοινωνικών δικτύων είναι το χρονολόγιο το οποίο επιτρέπει σε έναν χρήστη να βλέπει την δραστηριότητα των ατόμων που ακολουθεί σε χρονολογική σειρά. Στο συγκεκριμένο σενάριο το χρονολόγιο αποτελείται από τα post, τις κοινοποιήσεις και τα σχόλια που έχουν κάνει τα άτομα που ακολουθεί ένας χρήστης. Το ερώτημα δημιουργίας του χρονολογίου πρέπει να επιστρέφει μία λίστα από post σε φθίνουσα χρονολογική σειρά η οποία θα περιέχει τις εξής πληροφορίες για κάθε ένα από αυτά:

- Το id του post.
- Το περιεχόμενο του post.
- Τα στοιχεία του χρήστη που το δημιούργησε.
- Πόσα likes και πόσες κοινοποιήσεις έχει.
- Αν προέρχεται από κοινοποίηση άλλου post και ποιος χρήστης την έκανε.
- Αν πρόκειται για σχόλιο σε άλλο post και το όνομα του χρήστη που το δημιούργησε.
- Αν ο χρήστης του χρονολογίου έχει δηλώσει αρέσκεια ή έχει κοινοποιήσει το post.

Το ερώτημα που συντάσσει το χρονολόγιο του χρήστη φαίνεται στην Εικόνα 31.

```
MATCH (loggedIn:User {username: {username}})-[:FOLLOWS]-(followee:User)-[r:POSTED|REPOSTED]->(post:Post)
MATCH (post)<-[:POSTED]-(poster:User)
OPTIONAL MATCH (post)<-[:LIKED]-(liker:User) WHERE liker.username = loggedIn.username
OPTIONAL MATCH (post)<-[l:LIKED]-()
OPTIONAL MATCH (post)<-[:REPOSTED]-(reposter:User) WHERE reposter.username = loggedIn.username
OPTIONAL MATCH (post)<-[re:REPOSTED]-()
OPTIONAL MATCH (post)-[:REPLIED_TO]->()<-[:POSTED]-(user)
RETURN post.id AS id,
       post.content AS content,
       poster.username AS username,
       poster.fname AS fname,
       poster.lname AS lname,
       COUNT(DISTINCT l) AS likes,
       COUNT(DISTINCT re) AS reposts,
       CASE WHEN type(r) = "REPOSTED" THEN followee.username ELSE false END AS reposter,
       CASE WHEN liker.username = loggedIn.username THEN true ELSE false END AS liked,
       CASE WHEN reposter.username = loggedIn.username THEN true ELSE false END AS reposted,
       user.username AS reply,
       r.timestamp AS timestamp
ORDER BY timestamp DESC
```

Εικόνα 31: Δημιουργία χρονολογίου

4.3.8 Προφίλ Χρήστη

Κάθε χρήστης έχει ένα προσωπικό προφίλ το οποίο περιέχει τα προσωπικά του στοιχεία, τους χρήστες που ακολουθεί, τους χρήστες που τον ακολουθούν, τα μέρη που έχει επισκεφθεί και ένα χρονολόγιο με όλα τα post που έχει δημιουργήσει ή έχει κοινοποιήσει.

```
MATCH (u:User {username: {username}})<-[:FOLLOWS]-(user:User)
RETURN collect(user) AS users
```

Εικόνα 32: Άτομα που ακολουθούν έναν χρήστη

Στην Εικόνα 32 φαίνεται το ερώτημα που επιστρέφει τα άτομα που ακολουθούν τον χρήστη. Το ερώτημα το οποίο επιστρέφει τα άτομα που ακολουθεί ο χρήστης είναι παρόμοιο με την διαφορά ότι η σχέση `:FOLLOWS` έχει αντίθετη κατεύθυνση στο μοτίβο. Με μικρές αλλαγές παίρνουμε επίσης το ερώτημα που επιστρέφει τα μέρη που έχει επισκεφθεί ο χρήστης (βλ. Εικόνα 33).

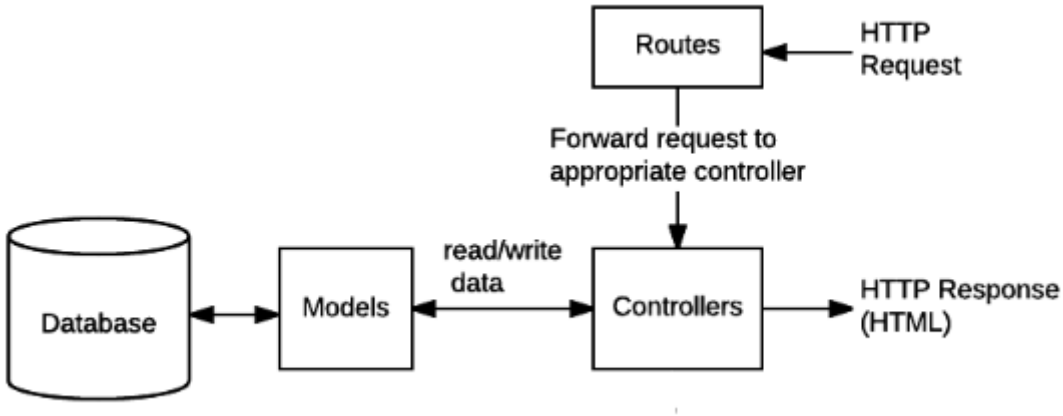
```
1 MATCH (user:User {username: {username}})-[:VISITED]->(place:Place)
2 RETURN place.name as name, place.place_id AS place_id
```

Εικόνα 33: Μέρη που έχει επισκεφθεί ο χρήστης

Το ερώτημα που δημιουργεί το προσωπικό χρονολόγιο του χρήστη είναι παρόμοιο με το ερώτημα στην Εικόνα 31, με την μόνη διαφορά ότι γίνεται αναζήτηση post που έχει δημιουργήσει ο ίδιος ο χρήστης και όχι τα άτομα που ακολουθεί.

4.4 Υλοποίηση back-end

Η υλοποίηση του back-end κομματιού έγινε σε περιβάλλον Node.js με την βοήθεια της βιβλιοθήκης Express.js. Στο διάγραμμα της Εικόνα 34 φαίνεται μια οπτική αναπαράσταση της ροής των δεδομένων μιας express/node εφαρμογής και των κομματιών που πρέπει να υλοποιηθούν έτσι ώστε να επιτευχθεί ο χειρισμός των HTTP αιτημάτων του client.



Εικόνα 34: Αρχιτεκτονική server

4.4.1 Επικοινωνία με την βάση δεδομένων

Για την επικοινωνία του server με την βάση δεδομένων χρησιμοποιήθηκε η βιβλιοθήκη `node-Neo4j` η οποία παρέχει μηχανισμούς που επιτρέπουν την ενσωμάτωση Cypher ερωτημάτων μέσα στον JavaScript κώδικα. Στην Εικόνα 35 φαίνεται ο κώδικας που εκτελείται για την σύνδεση της Neo4j βάσης με τον Node server.



Εικόνα 35: Σύνδεση με την βάση δεδομένων

Η τοποθέτηση αυτού του κώδικα σε ένα ξεχωριστό και αυτόνομο αρχείο επιτρέπει την επαναχρησιμοποίησή του και σε άλλα αρχεία του server χωρίς να χρειάζεται η εκ νέου σύνδεση με την βάση δεδομένων.

4.4.2 Models

Τα μοντέλα αντιπροσωπεύουν δεδομένα και είναι υπεύθυνα για όλες τις αλληλεπιδράσεις με την βάση δεδομένων μέσω της βιβλιοθήκης `node-Neo4j`. Για κάθε οντότητα που υπάρχει στην Neo4j βάση δεδομένων δηλώνεται και ένα ξεχωριστό μοντέλο το οποίο περιέχει τα απαραίτητα ερωτήματα προς την βάση που σχετίζονται με το μοντέλο αυτό. Στο παράδειγμα της Εικόνα 36 φαίνεται το μοντέλο Tag το οποίο αντιπροσωπεύει την οντότητα :Tag της βάσης δεδομένων και περιέχει τα απαραίτητα ερωτήματα που σχετίζονται με αυτό. Στην γραμμή 4 της εικόνας, γίνεται η αρχικοποίηση του μοντέλου ενώ στην γραμμή 9 φαίνεται μία από τις μεθόδους (`getPopularTags`) η οποία εκτελεί ένα συγκεκριμένο ερώτημα στην βάση δεδομένων που επιστρέφει μία λίστα με τα Tags που χρησιμοποιήθηκαν τις περισσότερες φορές.

```
1   const db = require('../config/database').db
2
3   // Private constructor
4   let Tag = module.exports = function Tag (_node) {
5     this._node = _node
6   }
7
8   // Get popular tags
9   Tag.getPopularTags = (callback) => {
10    const qp = {
11      query: [
12        'MATCH (tag:Tag)<-[t:HAS]-(:Post)',
13        'RETURN tag.name AS name, COUNT(t) AS count',
14        'ORDER BY count DESC'
15      ].join('\n')
16    }
17
18    db.cypher(qp, (err, result) => {
19      if (err) callback(err)
20      callback(null, result)
21    })
22  }
```

Εικόνα 36: Παράδειγμα μοντέλου Tag

4.4.3 Controllers

Οι controllers είναι υπεύθυνοι για τον διαχωρισμο του κώδικα των routes από τον κώδικα που χειρίζεται τα HTTP αιτήματα του client. Για κάθε μοντέλο που υπάρχει στον server υπάρχει και ένας controller ο οποίος καλεί τις μεθόδους των μοντέλων και επιστρέφει τα αποτελέσματα στον client. Στο παράδειγμα της Εικόνα 37, ο `TagController` καλεί την μέθοδο `getPopularTags` του μοντέλου `Tag` και στην γραμμή 10 στέλνει τα αποτελέσματα στον client με την εντολή `res.send(result)`

```
1   const Tag = require('../models/Tag')
2
3   module.exports = {
4     // Get popular tags
5     async getPopularTags (req, res) {
6       Tag.getPopularTags((err, result) => {
7         if (err) {
8           res.status(400).send(err)
9         } else {
10          res.send(result)
11        }
12      })
13    },
```

Εικόνα 37: Παράδειγμα controller TagController

4.4.4 Routes

Τα routes στην Express.js είναι υπεύθυνα να παραπέμπουν τα HTTP αιτήματα του client στους controllers. Στο παράδειγμα της Εικόνα 38 δηλώνεται ένα HTTP αίτημα με την μέθοδο GET στο URL μονοπάτι '`/tags/popular`'. Όταν λοιπόν ο client στείλει ένα GET αίτημα στο URL '`http://SERVER/tags/popular`' θα εκτελεστεί η μέθοδος `getPopularTags` του `TagController` η οποία επιστρέφει στον client τα πιο δημοφιλή tag.

```
156   app.get('/tags/popular',
157     TagController.getPopularTags)
```

Εικόνα 38: Παράδειγμα route

4.5 Υλοποίηση front-end

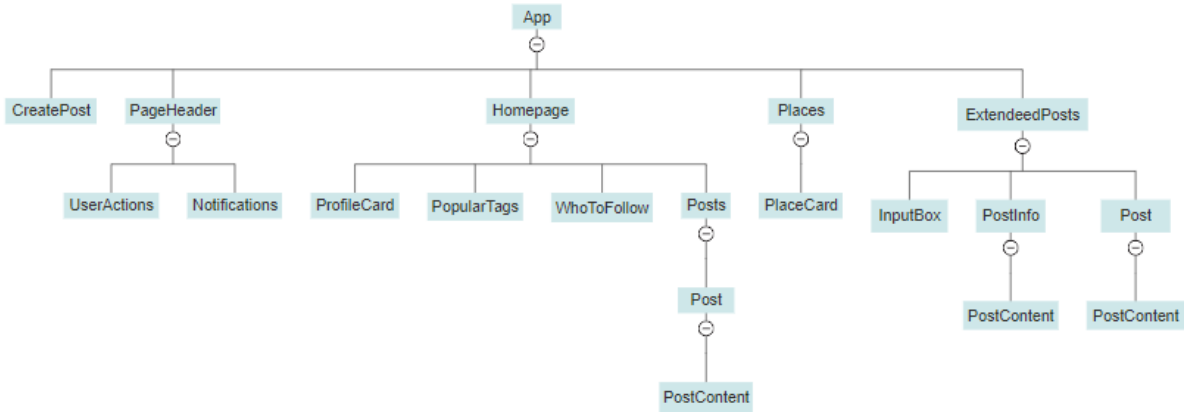
Ο παραδοσιακός τρόπος ανάπτυξης διαδικτυακών εφαρμογών βασίζεται στον server για την επεξεργασία των δεδομένων και την λογική της εφαρμογής ο οποίος εν τέλει στέλνει τα τελικά HTML αρχεία στον client. Τα front-end frameworks μας δίνουν την δυνατότητα να μεταφέρουμε όλη ή μέρος της επεξεργασίας και λογικής στον browser αφήνοντας έτσι τον server να διοχετεύει απλά τα δεδομένα από την βάση δεδομένων στον browser. Ένα από τα δυνατά χαρακτηριστικά του Vue είναι ότι παρέχει την δυνατότητα δημιουργίας component ενός αρχείου (single-file components) τα οποία είναι components που όλος ο κώδικάς τους περιέχεται σε ένα αυτοτελή αρχείο επιτρέποντας την καλύτερη οργάνωση του κώδικα. Όπως φαίνεται στην Εικόνα 39, ένα single-file component αποτελείται από τρία βασικά στοιχεία:

- `<template>`: Περιέχει το HTML κομμάτι του component
- `<script>`: Περιέχει τον JavaScript κώδικα και την λογική του component
- `<style>`: Περιέχει τα CSS styles του component

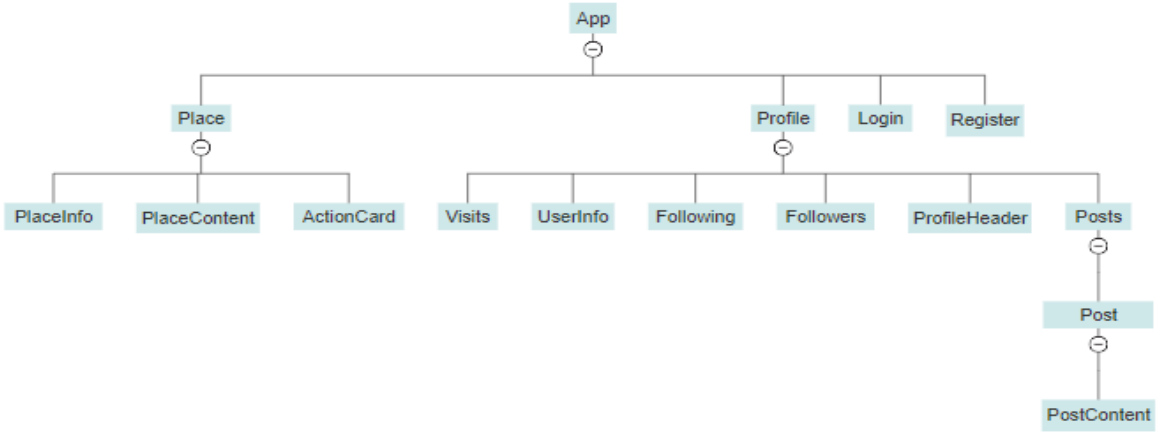
```
1   <template>
2     <div>
3       <!-- HTML Markup -->
4     </div>
5   </template>
6
7   <script>
8   import TagService from '@/services/TagService'
9
10  export default {
11    data () {
12      return {
13        tags: []
14      }
15    },
16    async created () {
17      this.tags = (await TagService.getPopularTags()).data
18    }
19  }
20  </script>
21
22  <style lang="scss" scoped>
23  // CSS Styles
24  </style>
```

Εικόνα 39: Παράδειγμα single-file component

Στόχος στην ανάπτυξη Vue εφαρμογών είναι ο διαχωρισμός των διαφορετικών λειτουργιών της εφαρμογής σε μια ιεραρχία από διαφορετικά, αυτοτελή και επαναχρησιμοποιούμενα single-file components. Η ιεραρχία της συγκεκριμένης Vue εφαρμογής φαίνεται στην Εικόνα 40 και την Εικόνα 41.



Εικόνα 40: Ιεραρχία Vue components (Μέρος 1)



Εικόνα 41: Ιεραρχία Vue components (Μέρος 2)

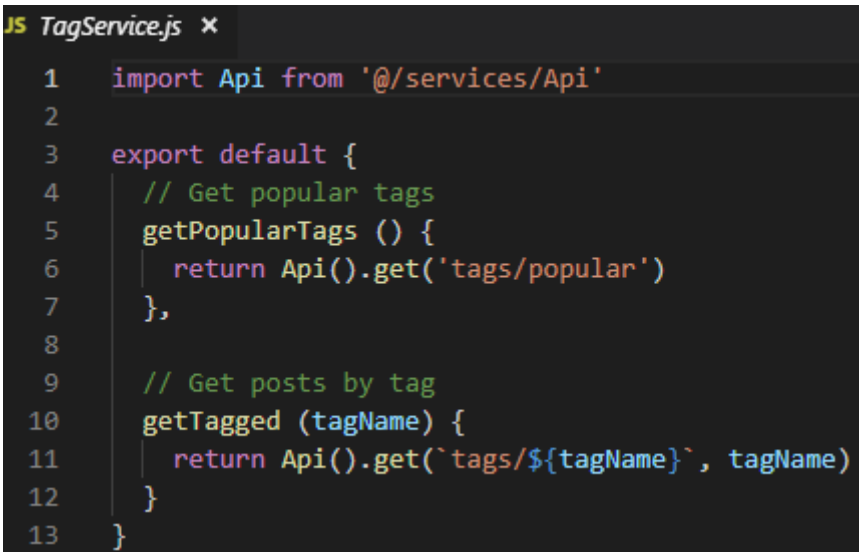
4.5.1 Χρησιμοποιώντας το back-end API

Για τη χρήση του back-end API χρησιμοποιήθηκε η βιβλιοθήκη Axios η οποία επιτρέπει στον client να στέλνει HTTP αιτήματα στον server. Ο κώδικας της Εικόνα 42 χρησιμοποιεί το Axios για την σύνδεση του client με τον server. Ο κώδικας αυτός βρίσκεται στο αρχείο `Api.js` και με την εντολή `export default` μπορεί να χρησιμοποιηθεί σε οποιοδήποτε άλλο αρχείο της εφαρμογής.



Εικόνα 42: Επικοινωνία client-server μέσω του Axios

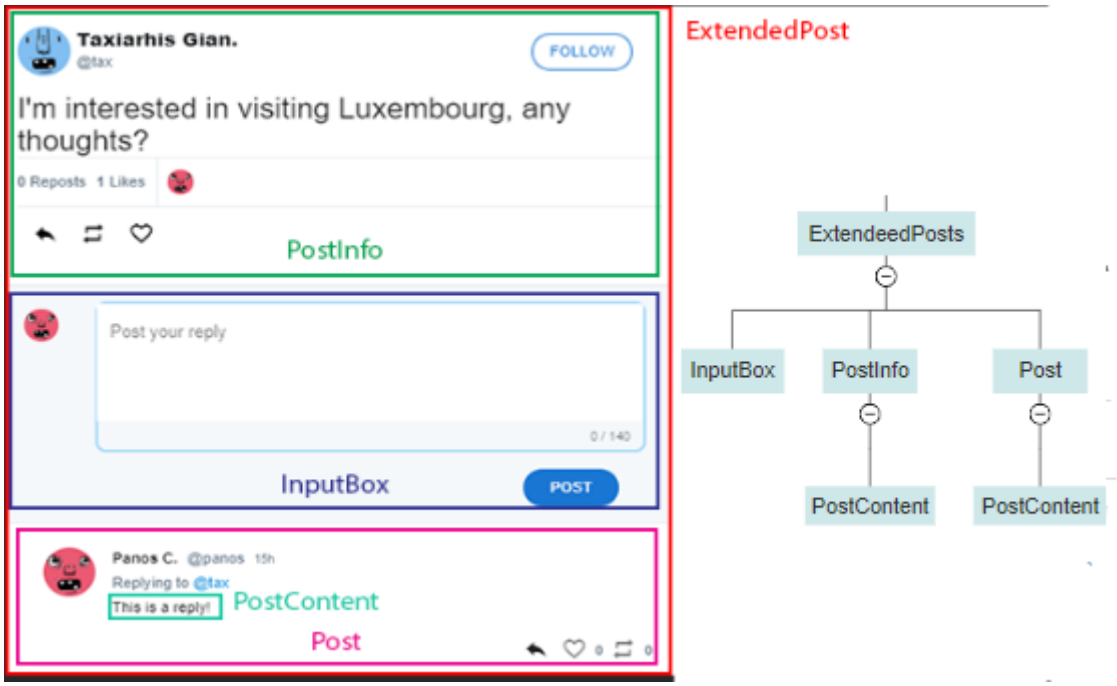
Η επικοινωνία με το back-end API γίνεται με την χρήση των services τα οποία είναι αρχεία που περιέχουν συγκεκριμένες μεθόδους που επικοινωνούν με συγκεκριμένα endpoints του API. Στο παράδειγμα της Εικόνα 43 φαίνεται το αρχείο `TagService.js` το οποίο περιέχει τις μεθόδους επικοινωνίας του client με τα API endpoints που σχετίζονται με τα Tags. Όμοιως, έχουν δημιουργηθεί διαφορετικά services για κάθε οντότητα που υπάρχει στη βάσης δεδομένων.



Εικόνα 43: Παράδειγμα TagService.js

4.5.2 Ροή δεδομένων

Τα components του Vue πρέπει να είναι όσο το δυνατόν πιο αυτόνομα και απομονωμένα το ένα από το άλλο. Ωστόσο είναι προφανές ότι στις περισσότερες περιπτώσεις υπάρχει η ανάγκη ενός επιπέδου επικοινωνίας μεταξύ των components μέσω της οποίας μπορούν να ανταλλάξουν δεδομένα. Για την καλύτερη κατανόηση της ροής των δεδομένων ανάμεσα στα Vue components θα χρησιμοποιήσουμε ως παράδειγμα το component `ExtendedPost` της Εικόνα 44 το οποίο έχει τρία components παιδιά.



Εικόνα 44: Παράδειγμα ExtendedPost

Δεδομένου ότι κάθε component είναι απομονωμένο από τα υπόλοιπα, τα components παιδία δεν μπορούν ποτέ να έχουν άμεση πρόσβαση στα δεδομένα του γονέα. Για να έχει πρόσβαση ένα component παιδί στα δεδομένα του γονέα τα δεδομένα πρέπει να ρέουν από τον γονέα στο παιδί με την βοήθεια του όρου `props` που παρέχει το Vue. Τα `props` αποτελούν χαρακτηριστικά των components και πρέπει να δηλώνονται αποκλειστικά στο component παιδί ενώ πρέπει να απολάβουν μία τιμή στο component γονέα. Ως αποτέλεσμα τα δεδομένα μέσω των props μπορούν να ρέουν μόνο προς μία κατεύθυνση από τον γονέα στο παιδί και όχι αντίθετα.

ExtendedPost.vue

```
<template>
  <v-card class="card--flex-toolbar" width="640px">
    <post-info :post="post" v-if="dataLoaded"></post-info>
    <input-box :poster="post.poster" @reply="reply"></input-box>
    <v-layout wrap>
      <v-flex xs12>
        <ol class="comments">
          <li class="comment" v-for="reply in replies" :key="reply.id">
            <post :post="reply"></post>
          </li>
        </ol>
      </v-flex>
    </v-layout>
  </v-card>
</template>
```

PostInfo.vue

```
<script>
export default {
  props: {
    post: {
      type: Object,
      required: true
    }
  }
}
</script>
```

Εικόνα 45: Παράδειγμα ροής δεδομένων μέσω props

Στο συγκεκριμένο παράδειγμα το component `ExtendedPost` περιέχει τα δεδομένα ενός post στα οποία πρέπει να έχει πρόσβαση το component παιδί `PostInfo`. Αυτό σημαίνει ότι πρέπει να δηλωθεί ένα prop ονόματος post στο `PostInfo` όπως φαίνεται στο δεξί σκέλος της Εικόνα 45. Στο αριστερό σκέλος φαίνεται το HTML κομμάτι του component

`ExtendedPost` το οποίο καλεί το `PostInfo` περνώντας του τα δεδομένα του post (βλ. γραμμή 3).

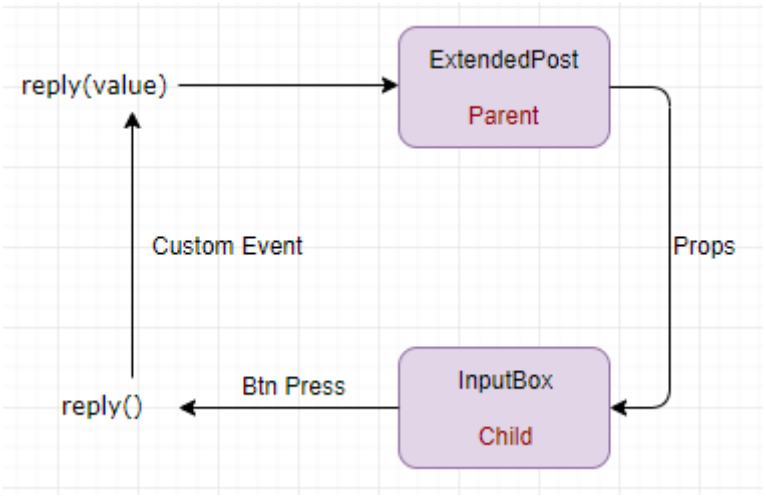
Δεδομένου ότι τα δεδομένα μέσω των props μπορούν να ρέουν μόνο προς μία κατεύθυνση από τον γονέα στο παιδί, τα παιδιά μπορούν να επικοινωνούν απευθείας με έναν γονέα με την χρήση των events. Τα events του Vue λειτουργούν με την δημιουργία συμβάντων σε ένα component, `$emit(customEvent)`, και την ακρόαση των συμβάντων αυτών από ένα άλλο component, `@customEvent = (doSomething)`.

```
1  <template>
2    <v-card class="card--flex-toolbar" width="640px">
3      <post-info :post="post" v-if="dataLoaded"></post-info>
4      <input-box :poster="post.poster" @reply="reply"></input-box>
5      <v-layout wrap>
6        <v-flex xs12>
7          <ol class="comments">
8            <li class="comment" v-for="reply in replies" :key="reply.id"
9              <post :post="reply"></post>
10           </li>
11         </ol>
12       </v-flex>
13     </v-layout>
14   </v-card>
15 </template>
```

```
1  <script>
2  export default {
3    // ...
4    methods: {
5      reply () {
6        // ...
7        this.$emit('reply', reply)
8      }
9    }
10 }
11 </script>
```

Εικόνα 46: Παράδειγμα ροής δεδομένων μέσω events

Στο παράδειγμά μας όταν ο χρήστης θέλει να αφήσει ένα σχόλιο στο post, εκτελείται η μέθοδος `reply()` του component `InputBox` (βλ. δεξί σκέλος Εικόνα 46), η οποία δημιουργεί το event `'reply'`. Στο αριστερό σκέλος της εικόνας, το component `ExtendedPost` περιέχει το component `InputBox` (γραμμή 4) και ακούει για το event `'reply'` (`@reply`). Στην συγκεκριμένη περίπτωση, όταν ενεργοποιηθεί το event `'reply'`, εκτελείται η μέθοδος `reply()` του `ExtendedPost` η οποία επικοινωνεί με τον server και δημιουργεί ένα νέο σχόλιο στη βάση. Στην Εικόνα 47 υπάρχει μία οπτική αναπαράσταση της ροής των δεδομένων του παραδείγματος που χρησιμοποιήθηκε.



Εικόνα 47: Οπτική αναπαράσταση ροής δεδομένων

4.6 Πιλοτική επίδειξη εφαρμογής

4.6.1 Χρονολόγιο προφίλ χρήστη

Η πρώτη εικόνα του χρήστη μετά την σύνδεσή του στην εφαρμογή είναι το προσωπικό του προφίλ όπου παρουσιάζονται τα τελευταία post που έχει δημιουργήσει καθώς και post άλλων χρηστών που έχει κοινοποιήσει. Για την εμφάνιση των post αυτών εκτελείται το ερώτημα τροχιάς της Εικόνα 48 το οποίο αναζητεί για κόμβους τύπου `:Post` που συνδέονται με τον χρήστη είτε με την σχέση `:POSTED` είτε με την σχέση `:REPOSTED`. Για κάθε ένα από αυτά τα post το ερώτημα αναζητεί και επιστρέφει τις εξής πληροφορίες:

- Το id του post.
- Το περιεχόμενο του post.
- Τα στοιχεία του χρήστη που δημιούργησε το post.
- Τον αριθμό των κοινοποιήσεων και likes του post.
- Το αν ο χρήστης έχει δηλώσει αρέσκεια ή έχει κοινοποιήσει το post.
- Το αν πρόκειται για σχόλιο σε άλλο post

```
MATCH (loggedIn:User {username: 'panos'})-[r:POSTED|REPOSTED]->(post:Post)
MATCH (post)<-[:POSTED]-(poster:User)
OPTIONAL MATCH (post)<-[:LIKED]-(liker:User) WHERE liker.username = loggedIn.username
OPTIONAL MATCH (post)<-[liked:LIKED]-()
OPTIONAL MATCH (post)<-[:REPOSTED]-(reposter:User) WHERE reposter.username = loggedIn.username
OPTIONAL MATCH (post)<-[reposted:REPOSTED]-()
OPTIONAL MATCH (post)-[:REPLIED_TO]->()<-[:POSTED]-(user)
RETURN post.id AS id,
       post.content AS content,
       poster.username AS username,
       poster.fname AS fname,
       poster.lname AS lname,
       COUNT(DISTINCT liked AS likes,
       COUNT(DISTINCT reposted) AS reposts,
       CASE WHEN liker.username = loggedIn.username THEN true ELSE false END AS liked,
       CASE WHEN reposter.username = loggedIn.username THEN true ELSE false END AS reposted,
       CASE WHEN exists(user.username) THEN user.username ELSE false END AS reply,
       r.timestamp AS timestamp
ORDER BY timestamp DESC
```

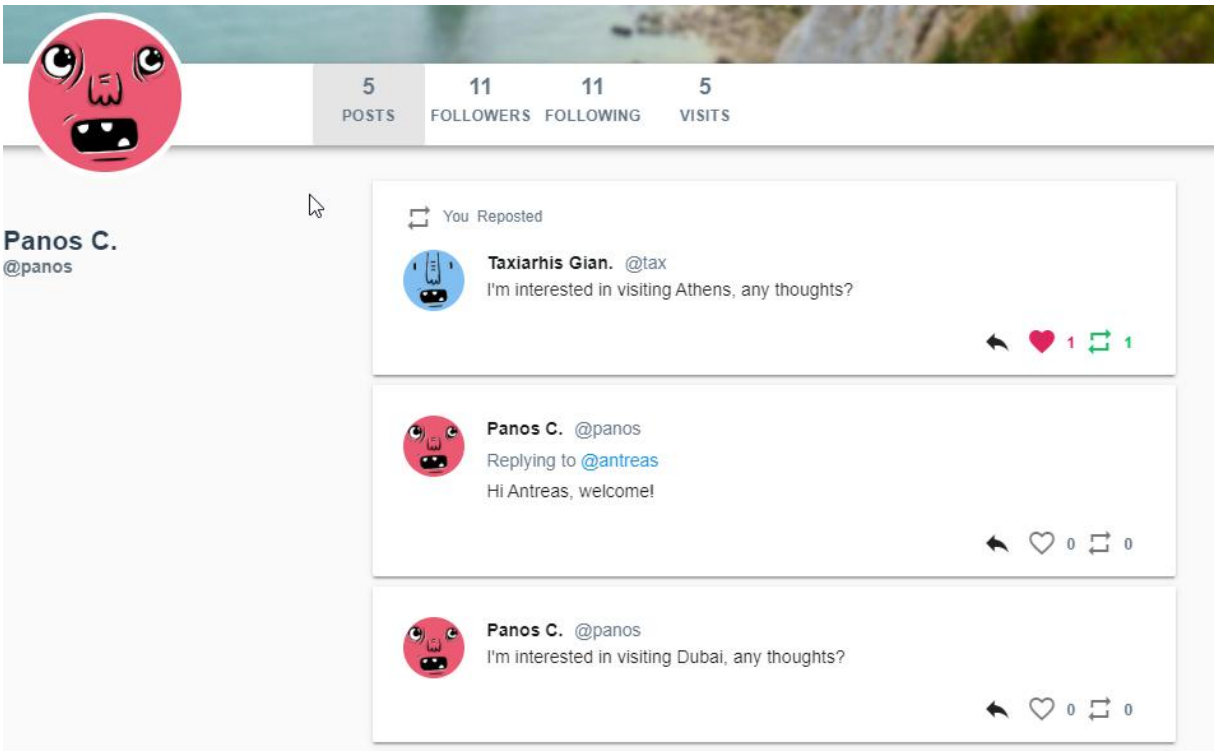
Εικόνα 48: Χρονολόγιο προφίλ χρήστη - Ερώτημα

Τα post επιστρέφονται σε φθίνουσα χρονολογική σειρά σύμφωνα με την ιδιότητα timestamp των σχέσεων `:POSTED` και `:REPOSTED`. Το αποτέλεσμα του ερωτήματος φαίνεται στην Εικόνα 49.

id	content	username	fname	lname	likes	reposts	liked	reposted	reply2	reply	timestamp
7	"I'm interested in visiting Athens, any thoughts?"	"tax"	"Taxiarhis"	"Gian."	1	1	true	true	false	null	1569233368846
9	"Hi Antreas, welcome!"	"panos"	"Panos"	"C."	0	0	false	false	"antreas"	"antreas"	1569229858950
6	"I'm interested in visiting Dubai, any thoughts?"	"panos"	"Panos"	"C."	0	0	false	false	false	null	1568991061104
5	"I like #vue too!"	"panos"	"Panos"	"C."	0	0	false	false	"tax"	"tax"	1568985835243
2	"This is a reply!"	"panos"	"Panos"	"C."	1	0	false	false	"tax"	"tax"	1566232525749

Εικόνα 49: Χρονολόγιο προφίλ χρήστη – Αποτέλεσμα

Τέλος, στην Εικόνα 50 παρουσιάζεται η τελική διεπαφή με όλα τα δεδομένα που προήλθαν από το property graph.



Εικόνα 50: Προσωπικό προφίλ χρήστη

Ας σημειωθεί ότι το ερώτημα δημιουργίας του χρονολόγιου του προφίλ του χρήστη διαφέρει από το ερώτημα δημιουργίας του γενικού χρονολόγιου του χρήστη που αναλύσαμε στην ενότητα 4.3.8 (βλ. Εικόνα 31).

4.6.2 Αρχική σελίδα εφαρμογής

Η αρχική σελίδα της εφαρμογής χωρίζεται σε 4 μέρη (βλ. Εικόνα 51) τα βασικά στοιχεία του χρήστη (περίγραμμα 1), τα πιο δημοφιλή tags (περίγραμμα 2), τις προτάσεις φιλίας (περίγραμμα 4) και το γενικό χρονολόγιο του χρήστη (περίγραμμα 3). Κάθε μέρος έχει σημειωθεί με έναν αριθμό για την εύκολη αναφορά του κατά τη διάρκεια της συγκεκριμένης ενότητας.



Εικόνα 51: Αρχική σελίδα εφαρμογής

Αρχικά, τα βασικά στοιχεία του χρήστη προέρχονται από την εκτέλεση του ερωτήματος της Εικόνα 52 το οποίο για έναν χρήστη επιστρέφει το όνομά του, των αριθμό των post που έχει δημιουργήσει ή έχει κοινοποιήσει, τα άτομα που ακολουθεί και τα άτομα που τον ακολουθούν.

```
MATCH (user:User {username: 'akp'})
OPTIONAL MATCH (user)-[following:FOLLOWS]->(:User)
OPTIONAL MATCH (user)<-[followers:FOLLOWS]-(:User)
OPTIONAL MATCH (user)-[r:POSTED|REPOSTED]->(:Post)
RETURN user.username AS username,
       user.fname AS fname,
       user.lname AS lname,
       COUNT(DISTINCT following) AS following,
       COUNT(DISTINCT followers) AS followers,
       COUNT(DISTINCT r) AS posts
```

Εικόνα 52: Βασικές πληροφορίες χρήστη - Ερώτημα

Στην Εικόνα 53 φαίνεται το αποτέλεσμα του ερωτήματος το οποίο εν τέλει αποτυπώνεται στο περίγραμμα 1 της Εικόνα 51.

username	fname	lname	following	followers	posts
"akp"	"Alex"	"Eagle"	7	6	1

Εικόνα 53: Βασικές πληροφορίες χρήστη - Αποτέλεσμα

Το ερώτημα της Εικόνα 54 αναζητά κόμβους τύπου `:Tag` που συνδέονται με την σχέση `:HAS` με κόμβους τύπου `:Post` και στη συνέχεια επιστρέφει το όνομα των tags που βρήκε καθώς και έναν μετρητή που δηλώνει πόσες φορές χρησιμοποιήθηκε το καθένα.

```
MATCH (tag:Tag)<-[t:HAS]-(:Post)
RETURN tag.name AS name, COUNT(t) AS count
ORDER BY count DESC
```

Εικόνα 54: Τα πιο δημοφιλή tags - Ερώτημα

Τα tags επιστρέφονται σε φθίνουσα σειρά σύμφωνα με το πόσες φορές έχει χρησιμοποιηθεί το καθένα (βλ. Εικόνα 55). Η διεπαφή που παρουσιάζει τα αποτελέσματα του συγκεκριμένου ερωτήματος φαίνεται στο περίγραμμα 2 της Εικόνα 51.

name	count
"#hello"	4
"#travel"	1
"#photography"	1
"#vue"	1

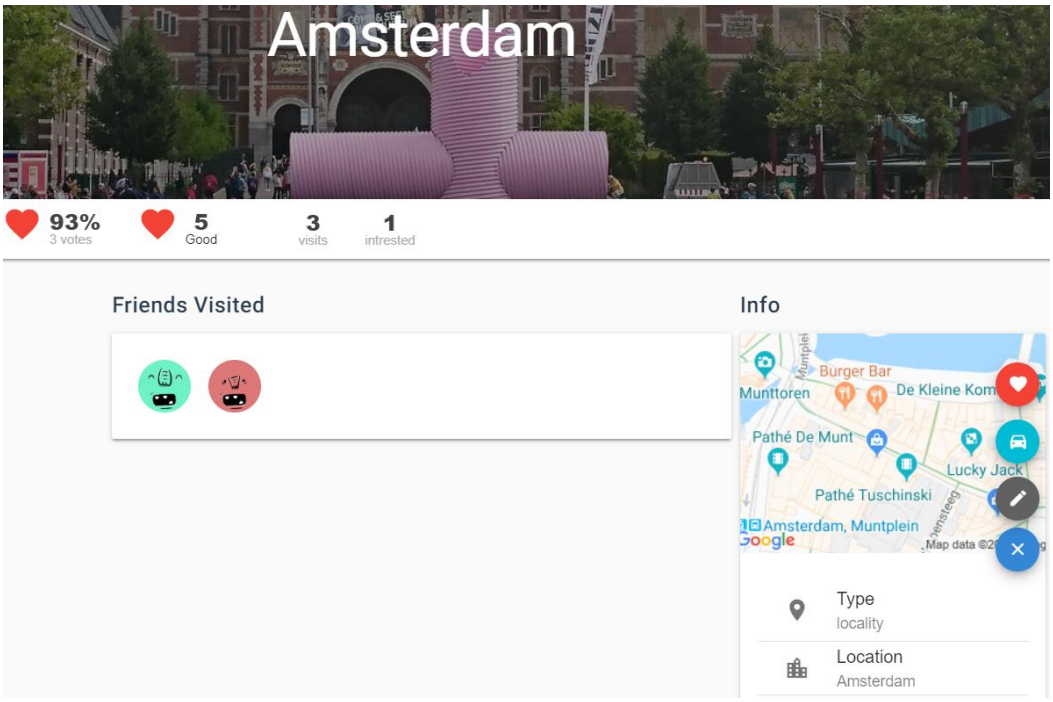
Εικόνα 55: Τα πιο δημοφιλή tags - Αποτέλεσμα

Στο περίγραμμα 4 της Εικόνα 51 οι προτάσεις φιλίας για τον χρήστη που είναι συνδεδεμένος στην εφαρμογή. Αυτό επιτυγχάνεται με την εκτέλεση του ερωτήματος προτάσεων φιλίας βάσει κοινών γνωστών (βλ. Εικόνα 30) που αναλύσαμε στην ενότητα 4.3.6.

Τέλος η υλοποίηση του γενικού χρονολόγιου στο περίγραμμα 3 της Εικόνα 51 γίνεται όμοια με την υλοποίηση του προσωπικού χρονολόγιου στο προφίλ του χρήστη που είδαμε στην ενότητα 4.6.2.

4.6.3 Τοποθεσίες

Στη σελίδα Places της εφαρμογής ο χρήστης μπορεί να αναζητήσει τοποθεσίες που τον ενδιαφέρουν. Επιλέγοντας μία τοποθεσία μεταφέρεται στην σελίδα της, όπου μπορεί να εκτελέσει λειτουργίες όπως βαθμολογία, επίσκεψη ή εκδήλωση ενδιαφέροντος.



Εικόνα 56: Σελίδα τοποθεσίας

Στο δεξί μέρος της Εικόνα 56 φαίνονται τα τρία action buttons τα οποία αντιστοιχούν στις τρεις λειτουργίες που αναφέρθηκαν. Όταν ο χρήστης θέλει να βαθμολογήσει μία τοποθεσία δημιουργείται μία σχέση τύπου `:RATED` μεταξύ ενός κόμβου τύπου `:User` και ενός κόμβου τύπου `:Place`. Αυτή η σχέση έχει μια ΄μοναδική ιδιότητα που ονομάζεται rating στην οποία αποθηκεύεται η βαθμολογία που έδωσε ο χρήστης. Το ερώτημα βαθμολογίας φαίνεται στην Εικόνα 25 (βλ. Ενότητα 4.3.4).

Ο χρήστης μπορεί επιπλέον να δει ποιοι από τους φίλους του έχουν επισκεφτεί το συγκεκριμένο μέρος. Αυτό γίνεται με την εκτέλεση του ερωτήματος της Εικόνα 57 το οποίο αρχικά αναζητά τους φίλους του χρήστη. Ως φίλους του χρήστη ορίζουμε τα άτομα που ακολουθεί και τον ακολουθούν ταυτόχρονα. Στη συνέχεια βρίσκει κάθε φίλο που έχει επισκεφτεί το μέρος και αποστρέφει μία λίστα με τα ονόματά τους.

```
MATCH (user:User {username: 'akp'})
MATCH (user)-[:FOLLOWS]->(followee:User)
WITH collect(followee) AS followeeList, user
MATCH (follower:User)-[:FOLLOWS]->(user)
WITH followeeList, collect(follower) AS followerList
WITH [x IN followeeList WHERE x IN followerList] AS friendsList
UNWIND friendsList AS friends
MATCH (place:Place {place_id: 'ChIJVXealLU_xkcRja_At0z9AGY'})
MATCH (friends)-[:VISITED]->(place)
RETURN collect(friends.username) AS friendVisitors
```

Εικόνα 57: Φίλοι που έχουν επισκεφτεί ένα μέρος

Τέλος, ο χρήστης μπορεί να δει στατιστικά που σχετίζονται με την τοποθεσία όπως την συνολική βαθμολογία του, πόσα άτομα το έχουν βαθμολογήσει, πόσα άτομα το έχουν επισκεφτεί και πόσα άτομα ενδιαφέρονται να το επισκεφτούν. Μπορεί επίσης να δει αν ο ίδιος εκτελέσει κάποιες από αυτές τις λειτουργίες.

```
MATCH (place:Place {place_id: 'ChIJVXealLU_xkcRja_At0z9AGY'})
MATCH (user:User {username: 'akp'})
OPTIONAL MATCH (user)-[r1:RATED]->(place)
WITH place, user, r1
OPTIONAL MATCH (place)<-[r:RATED]-(rateUsers)
WITH place,
     user,
     r1,
     count(DISTINCT rateUsers) AS rateCount,
     avg(r.rating) AS ratingAvg,
     collect(rateUsers.username) AS ratings
OPTIONAL MATCH (place)<-[:VISITED]-(visitUsers)
WITH place,
     user,
     r1,
     rateCount,
     ratingAvg,
     ratings,
     count(DISTINCT visitUsers) AS visitCount,
     collect(visitUsers.username) AS visits
OPTIONAL MATCH (place)<-[:INTERESTED_IN]-(interestPost)<-[:POSTED]-(interestUsers)
WITH user,
     r1,
     rateCount,
     visitCount,
     ratingAvg,
     count(DISTINCT interestUsers) AS interestCount,
     collect(interestUsers.username) AS interests,
     ratings,
     visits
RETURN rateCount,
       visitCount,
       interestCount,
       ratingAvg,
       CASE WHEN user.username IN ratings THEN r1.rating ELSE false END AS rated,
       CASE WHEN user.username IN visits THEN true ELSE false END AS visited,
       CASE WHEN user.username IN interests THEN true ELSE false END AS interested
```

Εικόνα 58: Στατιστικά τοποθεσίας - Ερώτημα

Στην Εικόνα 58 φαίνεται το ερώτημα που επιστρέφει τα στατιστικά μιας συγκεκριμένης τοποθεσίας και στην Εικόνα 59 φαίνονται τα αποτελέσματα.

rateCount	visitCount	interestCount	ratingAvg	rated	visited	interested
3	3	1	4.666666666666667	5	true	false

Εικόνα 59: Στατιστικά τοποθεσίας - Αποτέλεσμα

Στην συγκεκριμένη περίπτωση ο χρήστης έχει επισκεφτεί και βαθμολογήσει το μέρος αλλά δεν έχει δηλώσει ενδιαφέρον. Αυτό αποτυπώνεται στην διεπαφή της Εικόνα 56 με τον χρωματισμό των αντίστοιχων action buttons.

5 Επίλογος

Στο κεφάλαιο αυτό θα γίνει μία συνολική αποτίμηση της πτυχιακής εργασίας. Θα αναφερθούν οι νέες γνώσεις και δεξιότητες που αποκτήθηκαν από τον φοιτητή καθώς και πιθανές επεκτάσεις οι οποίες μπορούν να χρησιμοποιηθούν ως θέματα μελλοντικών πτυχιακών εργασιών.

5.1 Γνώσεις που αποκτήθηκαν

Στα πλαίσια της πτυχιακής εργασίας αναπτύχθηκε ένα πρωτότυπο μίας διαδικτυακής εφαρμογής κοινωνικής δικτύωσης με την χρήση του συστήματος Neo4j που βασίζεται στη διαχείριση βάσεων δεδομένων τύπου γράφος. Οι ειδικοί στόχοι της εργασίας ήταν τρεις:

- η γενική μελέτη και επισκόπηση των διαφόρων NoSQL μοντέλων δεδομένων που έχουν αναπτυχθεί τα τελευταία χρόνια
- η αναλυτική επισκόπηση του συστήματος Neo4j, του μοντέλου δεδομένων που υιοθετεί (δηλ. το property graph) και της γλώσσας υποβολής ερωτημάτων cypher
- η ανάπτυξη μίας εφαρμογής με τη χρήση του Neo4j συστήματος, με σκοπό την επίδειξη αληθινών περιπτώσεων χρήσης τις οποίες θα συναντούσε κανείς σε πραγματικές εφαρμογές κοινωνικής δικτύωσης, και το πως αυτές επιλύονται με τη χρήση μίας βάσης δεδομένων γράφων.

Αρχικά, η αναλυτική θεωρητική μελέτη του Neo4j συστήματος καθώς και η πρακτική υλοποίηση μίας Neo4j βάσης δεδομένων είχε ως αποτέλεσμα την επέκταση των πρακτικών δεξιοτήτων του φοιτητή όσον αφορά την διαχείριση δεδομένων με την χρήση ενός property graph. Επιπλέον, η ανάπτυξη μίας διαδικτυακής εφαρμογής έφερε σε επαφή τον φοιτητή με τεχνολογίες της τρέχουσας τεχνολογικής τροχιάς. Αποκτήθηκαν νέες γνώσεις και δεξιότητες όσον αφορά συγκεκριμένες σύγχρονες βιβλιοθήκες και frameworks καθώς και το πως αυτά αλληλοεπιδρούν μεταξύ τους για να συνθέσουν μια σύγχρονη διαδικτυακή full-stack εφαρμογή. Τέλος, η γενική μελέτη των διαφόρων NoSQL τεχνολογιών οδήγησε στο συμπέρασμα ότι κάθε διαφορετικό NoSQL μοντέλο δεδομένων ευδοκιμεί σε διαφορετικές περιπτώσεις χρήσης. Αυτό έδωσε στον φοιτητή την κριτική ικανότητα να επιλέξει το κατάλληλο μοντέλο δεδομένων σε μελλοντικά προβλήματα διαχείρισης δεδομένων που τυχών του παρουσιαστούν.

5.2 Μελλοντική Εργασία και Επεκτάσεις

Λόγω του περιορισμένου χρόνου υλοποίησης της πτυχιακής εργασίας, είναι προφανές ότι υπάρχουν αρκετά περιθώρια βελτίωσης ή επέκτασής της. Οι βελτιώσει αφορούν κυρίως την ανάπτυξη επιπλέον λειτουργιών στην εφαρμογή οι οποίες θα εξετάζουν νέα σενάρια που απαιτούν την δημιουργία επιπλέον σύνθετων ερωτημάτων στην βάση ή την αλλαγή του γράφου. Τέτοια σενάρια μπορεί να είναι η αναζήτηση των post και των tags από τους χρήστες, η δημιουργία επιπλέον τρόπων επικοινωνίας μεταξύ των χρηστών (π.χ. chatting), ή η επέκταση του τουριστικού κομματιού με τη δημιουργία ενός μηχανισμού σχεδίασης διακοπών.

Για την υλοποίηση των διαφόρων σεναρίων η εφαρμογή χρησιμοποίησε κυρίως υποθετικά σύνολα δεδομένων μικρής κλίμακας. Αυτό επιτρέπει σε μελλοντικές εργασίες να ασχοληθούν με την χρήση του Neo4j συστήματος βάσεων δεδομένων για την διαχείριση μεγάλου όγκου δεδομένων. Ερωτήματα που μπορεί να προκύψουν είναι το πως πρέπει να κατασκευαστεί ένα property graph το οποίο πιθανόν διατηρεί εκατομμύρια σχέσεις μεταξύ δύο κόμβων, έτσι ώστε να διαχειρίζεται αποτελεσματικά τα δεδομένα. Είναι εφικτός ο διαχωρισμός ενός γράφου έτσι ώστε να λειτουργήσει σε ένα κατανεμημένο περιβάλλον και πως αυτό θα επηρέαζε τα ερωτήματα τροχιάς.

Τέλος, όπως προαναφέρθηκε, κάθε NoSQL μοντέλο δεδομένων υπερτερεί σε συγκεκριμένες περιπτώσεις χρήσης. Σύνθετες εφαρμογές συνδυάζουν διάφορους τύπους προβλημάτων διαχείρισης δεδομένων. Είναι προφανές ότι η επιλογή της κατάλληλης τεχνολογίας αποθήκευσης δεδομένων μπορεί να αποδειχθεί πιο παραγωγική από την προσπάθεια επίλυσης όλων των πτυχών του προβλήματος με την ίδια τεχνολογία. Μελλοντικές εργασίες λοιπόν, θα μπορούσαν να ασχοληθούν με την ανάπτυξη μίας εφαρμογής η οποία θα χρησιμοποιεί διαφορετικές τεχνολογίες NoSQL βάσεων δεδομένων για την αντιμετώπιση των διαφορετικών αναγκών αποθήκευσης δεδομένων που τυχόν προκύψουν.

Βιβλιογραφία

[1] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks”, IBM Research Lab, San Jose, CA, vol. 13, no. 6, pp. 377-387, Feb. 1970

[2] D. Sullivan, “Different Databases for Different Requirements” in NoSQL for Mere Mortals, Boston, MA, Addison-Wesley, 2015

[3] R. Carvelli “SQL & NoSQL: A Brief History”, 2015. [Online]. Available: http://blog.grio.com/2015/11/sql-nosql-a-brief-history.html

[4] Microsoft.com, “ACID properties”, 2006. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/office/developer/server-technologies/aa480356(v=msdn.10)

[5] S. Tiwari, “Distributed ACID Systems” in Professional NoSQL, Indianapolis, IN, John Wiley & Sons, 2011

[6] MongoDB.com “What is NoSQL”. [Online]. Available: https://www.mongodb.com/nosql-inline

[7] N. Leavitt, “Will NoSQL Databases Live Up to Their Promise?”, Computer, vol. 43, No. 2, pp. 12-14, Feb. 2010

[8] R. Hecht, S. Jablonski, “NoSQL evaluation: A use case oriented survey”, Proceedings 2011 International Conference on Cloud and Service Computing, CSC 2011, no. 6138544, 2011

[9] V. Manoj, “Comparative Study of NoSQL Document, Column Store Databases and Evaluation of Cassandra”, IJDMS, India, vol. 6, no. 6, Aug 2014

[10] MongoDB.com, “Use Cases”. [Online]. Available: https://docs.mongodb.com/ecosystem/use-cases/

[11] D. Featherston, “Cassandra: Principles and Application”, Department of Computer Science University of Illinois at Urbana-Champaign

[12] A. Lakshman, P. Malik, “. Cassandra: a decentralized structured storage system”, ACM SIGOPS Operating Systems Review, vol. 44, no. 2, pp. 35-40, April 2010

[13] I. Robinson, J. Webber, E. Eifrem, “Options for Storing Connected Data” in Graph Databases, O’Reilly Media, 2015

[14] M. A. Rodriguez, P. Neubauer, “The Graph Traversal Pattern”, arXiv:1004.1001, Apr. 2010

[15] M. A. Rodrigues, P. Neubauer, “Constructions from Dots and Lines”, Bulletin of the American Society for Information Science and Technology, vol. 36, no. 6, pp 35-41, Sep. 2010

[16] Javatpoint.com, “Types of Graphs”, [Online]. Available:

https://www.javatpoint.com/graph-theory-types-of-graphs
[17] E. Miller, “An Introduction to the Resource Description Framework”, Bulletin of the American Society for Information Science and Technology, vol 25, no. 1, Jan. 2005
[18] O. van Rast, S. Hong, J. Kim, X. Meng, H. Chafi, “PGQL: a property graph query language”, Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, no. 7, CA, Jun. 2016
[19] SPARQL Query Language for RDF, [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/
[20] R. Angles, M. Arenas, P. Barcelo, A. Hogan, J. Reutter, D. Vrgoc “Foundations of Modern Query Languages for Graph Databases”, ACM Computing Surveys (CSUR), Vol. 50, no. 68, Nov. 2017
[21] I. Robinson, J. Webber, E. Eifrem, “Graph Database Internals” in Graph Databases, O’Reilly Media, 2015
[22] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, “Cypher: An Evolving Query Language for Property Graphs”, In ACM SIGMOD International Conference on Management of Data, TX, 1433-1445, 2018
[23] https://github.com/Panos-JC/orbit-project