ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη υπηρεσίας κοινωνικού δικτύου σε περιβάλλον Android με έμφαση στην γεωγραφική θέση του χρήστη (Geolocation)

Διπλωματική Εργασία

Καλομοίρης Νικόλαος

Επιβλέπουσα: Τσομπανοπούλου Παναγιώτα

Βόλος, Μάρτιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

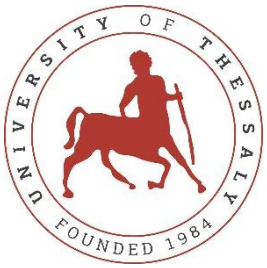
Ανάπτυξη υπηρεσίας κοινωνικού δικτύου σε περιβάλλον Android με έμφαση στην γεωγραφική θέση του χρήστη (Geolocation)

Διπλωματική Εργασία

Καλομοίρης Νικόλαος

Επιβλέπουσα: Τσομπανοπούλου Παναγιώτα

Βόλος, Μάρτιος 2021



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Development of social networking service in android environment with emphasis on the geographical location of the user (Geolocation)

Diploma Thesis

Kalomoiris Nikolaos

Supervisor: Tsompanopoulou Panagiota

Volos, March 2021

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπουσα **Τσομπανοπούλου Παναγιώτα**

Αναπληρώτρια Καθηγήτρια, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Θάνος Γεώργιος**

Μέλος Ε.ΔΙ.Π, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Μέλος **Σταμούλης Γεώργιος**

Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Ημερομηνία έγκρισης: 01-03-2021

ΕΥΧΑΡΙΣΤΙΕΣ

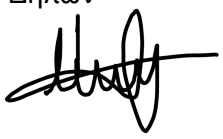
Το παρών έργο αποτελεί διπλωματική εργασία στα πλαίσια των προπτυχιακών σπουδών του τμήματος «Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών» του Πανεπιστημίου Θεσσαλίας.

Πριν την παρουσίαση της εν λόγω διπλωματικής εργασίας, θα ήθελα να ευχαριστήσω ορισμένους ανθρώπους που συνεργάστηκα μαζί τους και έπαιξαν πολύ σημαντικό ρόλο στην εκπόνηση της. Πρώτα απ' όλα θα ήθελα να ευχαριστήσω την επιβλέπουσα κ. Τσομπανοπούλου Παναγιώτα και τους επιβλέποντες κ. Θάνο Γεώργιο και κ. Σταμούλη Γεώργιο για την πολύτιμη καθοδήγησή τους και την εμπιστοσύνη που μου έδειξαν. Τέλος, ιδιαίτερες ευχαριστίες θέλω να απευθύνω στην οικογένεια μου και στους κοντινούς μου ανθρώπους, που με υπομονή και κουράγιο πρόσφεραν την απαραίτητη ηθική συμπαράσταση σε κάθε φάση της σταδιοδρομίας μου.

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών



Καλομοίρης Νικόλαος

1 Μαρτίου 2021

ΠΕΡΙΛΗΨΗ

Τα κοινωνικά δίκτυα έχουν επηρεάσει σε μεγάλο βαθμό την καθημερινότητα των ανθρώπων, αποτελώντας πλέον μια από τις μεγαλύτερες πηγές ενημέρωσης αλλά και επικοινωνίας μεταξύ τους. Η εν λόγω διπλωματική εργασία διαπραγματεύεται την υλοποίηση μίας εφαρμογής κοινωνικού δικτύου σε περιβάλλον Android. Η παρούσα εφαρμογή δίνει έμφαση στην χρήση της γεωτοποθεσίας του χρήστη, για να έχει ο ίδιος την δυνατότητα να μοιραστεί με τους ακόλουθούς του, την τοποθεσία του πάνω στον χάρτη, καθώς και να δημιουργεί σημειώσεις και εκδηλώσεις σε μορφή σημείου πάνω σε αυτόν. Αναλυτικότερα, θα έχει την δυνατότητα να καλεί τους φίλους του σε εκδηλώσεις και να επικοινωνεί μαζί τους μέσω στιγμιαίων μηνυμάτων, να τοποθετεί σημειώσεις στον χάρτη με σημαντικά γεγονότα και συμβάντα που έχουν γίνει στην εκάστοτε τοποθεσία και θα μπορεί να βρει την βέλτιστη διαδρομή για να πλοηγηθεί σε αυτά. Τέλος, θα αναλυθεί η αρχιτεκτονική της εφαρμογής, η διεπαφή του χρήστη, γιατί επιλέχθηκε η συγκεκριμένη, καθώς και ό,τι αφορά το κομμάτι των υπηρεσιών που εκτελούνται για την λειτουργία της εφαρμογής.

ABSTRACT

Social networks have greatly influenced the daily lives of people, being now one of the largest sources of information and communication between them. This diploma thesis deals with the implementation of a social networking application in an Android environment. This application focuses on user's geolocation, so that he can share his location on the map with his followers, as well as create notes and events in the form of a marker on it. In more detail, he will be able to invite his friends to the events and communicate with them via chat, place notes on the map with important events and happenings that have taken place in each location and will be able to find the best route to navigate to them. We will analyze the architecture of the application, the user interface and the reasoning behind its use as well as the services that are executed in the application.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

1.1 Περιγραφή της ιδέας

Τις τελευταίες δεκαετίες, τα κοινωνικά δίκτυα έχουν επηρεάσει την ζωή του ανθρώπου σε τέτοιο βαθμό, ώστε η χρήση τους να αποτελεί αναπόσπαστο κομμάτι της καθημερινότητας του. Η ευκολότερη επικοινωνία και διατήρηση των σχέσεων με άλλους ανθρώπους, καθώς και η πληθώρα ειδήσεων και πληροφοριών που μπορούν να βρουν σε πολύ μικρό χρονικό διάστημα, είναι κάποιοι από του λόγους που είχαν τόσο μεγάλο αντίκτυπο στην κοινωνία [1].

Ένα είδος κοινωνικού δικτύου αποτελεί το Geosocial Network [2], στο οποίο χρησιμοποιούνται γεωγραφικές υπηρεσίες και δυνατότητες, όπως το geocoding και το geotagging, που προσθέτουν περισσότερες δυνατότητες και κοινωνική δυναμική στις εφαρμογές κοινωνικής δικτύωσης. Με αυτές τις λειτουργίες, ο χρήστης μπορεί να αλληλεπιδράσει με κοινωνικά γεγονότα που σχετίζονται με την ευρύτερη τοποθεσία του, όπως η συσχέτιση του χρήστη με μαγαζιά διασκέδασης ή εστιατόρια, ο προγραμματισμός κοινωνικών εκδηλώσεων κτλ. Παραδείγματα τέτοιων εφαρμογών αποτελούν το Foursquare [3] και το Yelp [4] που επιτρέπουν στους χρήστες να μοιράζονται την τοποθεσία τους και τους προτείνουν νέα μέρη και σημεία ενδιαφέροντος.

Η διπλωματική θα βασιστεί πάνω στις εφαρμογές αυτού του είδους, προσφέροντας στον χρήστη την δυνατότητα να μοιραστεί την τοποθεσία του με φίλους, να δημιουργήσει εκδηλώσεις και να καταγράψει σημεία ενδιαφέροντος στον χάρτη, προσθέτοντας δικές του περιγραφές για την κατάλληλη καθοδήγηση των φίλων που ενδιαφέρονται ή είναι καλεσμένοι στις εκδηλώσεις. Τέλος, θα έχει την δυνατότητα πλοήγησης στον χάρτη στα συγκεκριμένα σημεία ενδιαφέροντος και στις εκδηλώσεις, αλλά και την δυνατότητα άμεσης επικοινωνίας μέσω μηνυμάτων με τους φίλους του στην εφαρμογή.

1.2 Στόχος της διπλωματικής

Η εν λόγω διπλωματική έχει ως στόχο την δημιουργία μιας εφαρμογής κοινωνικής δικτύωσης σε λογισμικό Android που θα χρησιμοποιεί την γεωτοποθεσία σαν βασικό συστατικό αλληλεπίδρασης μεταξύ των χρηστών. Η εφαρμογή εισάγει την έννοια της φιλίας (*friendship*) ως την αμοιβαία αποδεκτή σχέση μεταξύ δύο χρηστών της εφαρμογής. Δύο χρήστες που αμοιβαία αποδέχονται τα αιτήματα φιλίας του ενός προς τον άλλο, έχουν στη συνέχεια τη δυνατότητα να αλληλεπιδρούν μεταξύ τους ανταλλάσσοντας πληροφορίες για εκδηλώσεις και γεγονότα που κοινοποιούνται εντός της εφαρμογής.

Επιπρόσθετα, δίνεται μεγάλη έμφαση στη χρήση της πληροφορίας γεωτοποθεσίας των χρηστών και των επιμέρους συμβάντων. Συγκεκριμένα, μπορούν να δημιουργηθούν δύο τύποι συμβάντων: α) εκδηλώσεις και β) σημειώσεις για γεγονότα που συμβαίνουν σε πραγματικό χρόνο.

Πιο αναλυτικά, δίνεται η δυνατότητα δημιουργίας εκδηλώσεων σε συγκεκριμένη θέση πάνω στον χάρτη. Η πληροφορία που συνοδεύει κάθε εκδήλωση αποτελείται από μία σύντομη περιγραφή της , τον διοργανωτή της εκδήλωσης, την ημερομηνία και ώρα που θα γίνει και ποιοι θα παρευρεθούν.

Επιπρόσθετα, δίνεται η δυνατότητα δημιουργίας σημειώσεων τύπου “post-it” πάνω στο χάρτη , όπως για παράδειγμα η δημοσιοποίηση ότι έχει γίνει κάποιο ατύχημα σε ένα σημείο της πόλης, , η δημοσιοποίηση σε πραγματικό χρόνο ότι υπάρχει αυξημένη κίνηση οχημάτων ή ότι είναι σε εξέλιξη διαδήλωση ή άλλο αναπάντεχο γεγονός. Οι σημειώσεις διαγράφονται από τον χάρτη μετά την πάροδο συγκεκριμένου χρονικού διαστήματος που επιλέγεται από τον δημιουργό τους . Η εφαρμογή παρέχει στον χρήστη επιλογές πλοήγησης στις εκδηλώσεις και στις σημειώσεις πάνω στον χάρτη, ενώ δυνατότητα αλληλεπίδρασης με τους άλλους χρήστες της.

1.3 Οργάνωση κεφαλαίων

Στα παρακάτω κεφάλαια θα αναπτυχθούν και θα αναλυθούν οι τεχνολογίες που χρησιμοποιήθηκαν για την δημιουργία της εφαρμογής, καθώς και μία εκτενής παρουσίαση της διεπαφής χρήστη της εφαρμογής.

Στο κεφάλαιο 1, έχουμε την εισαγωγή της διπλωματικής, που εμπεριέχει την περιγραφή της ιδέας και του στόχου της, καθώς και με ποιο τρόπο οργανώνονται τα κεφάλια της.

Στο κεφάλαιο 2, έχουμε το τεχνολογικό υπόβαθρο της διπλωματικής, στον οποίο ερμηνεύονται κάποιες τεχνολογικές έννοιες και ορισμοί που θα χρησιμοποιηθούν στην συνέχεια.

Στο κεφάλαιο 3, έχουμε την τεχνολογική επισκόπηση της εφαρμογής, στην οποία αναλύονται εκτενέστερα οι τεχνολογίες, οι υπηρεσίες και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της.

Στο κεφάλαιο 4, παρουσιάζεται η αρχιτεκτονική της εφαρμογής σε ότι αφορά την βάση δεδομένων της, καθώς και τα βασικά αρχεία από τα οποία αποτελείτε.

Στο κεφάλαιο 5, περιγράφεται και αναλύεται η διεπαφή χρήστη της εφαρμογής, παρουσιάζοντας και εικόνες του γραφικού της περιβάλλοντος.

Στο κεφάλαιο 6, έχουμε την ανάλυση των υπηρεσιών της εφαρμογής που είναι υπεύθυνες για την ομαλή λειτουργία της και για την επικοινωνία της με third party services.

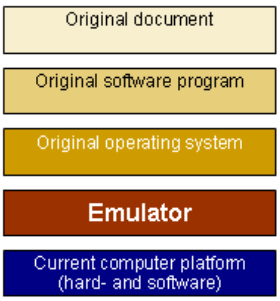
Τέλος, στο κεφάλαιο 7 έχουμε τον επίλογο και τα συμπεράσματα της διπλωματικής καθώς και τυχών επεκτάσεις που θα μπορούσαν να υλοποιηθούν στο μέλλον.

ΚΕΦΑΛΑΙΟ 2

ΤΕΧΝΙΚΟ ΥΠΟΒΑΘΡΟ

2.1 Βασικοί ορισμοί

Emulator



Εικόνα 2.1: Emulator [21]

Κάθε υπολογιστικό σύστημα αποτελείται από δύο στοιχεία, το υλικό και το λογισμικό του. Με την ενσωμάτωση αυτών των δύο στοιχείων, ένας υπολογιστής μπορεί να προσφέρει μια πληθώρα υπηρεσιών και δυνατοτήτων στον χρήστη, όπως για παράδειγμα η δημιουργία προγραμμάτων για την επίλυση καθημερινών αναγκών. Η δυνατή εξάρτιση μεταξύ υλικού και λογισμικού μπορεί να δημιουργήσει και πολλούς κινδύνους όπως η καταστροφή δεδομένων σε περίπτωση αποτυχίας λογισμικού ή υλικού του συστήματος. Η προσομοίωση είναι μια λύση για την αποφυγή τέτοιων κινδύνων και τα συστήματα που γίνεται αυτή ονομάζονται προσομοιωτές (Emulators). Emulator, είναι ένα λογισμικό ή υλικό, που επιτρέπει σε ένα υπολογιστικό σύστημα να συμπεριφέρεται και να εκτελεί προγράμματα ενός διαφορετικού υπολογιστικού συστήματος. Προσθέτει μία επιπλέον στρώση μεταξύ της υπολογιστικής πλατφόρμας που το εκτελεί με την πλατφόρμα που αναπαράγει [21]. Στην *Εικόνα 2.1* βλέπουμε την δομή του προσομοιωτή σε ένα λειτουργικό σύστημα.

API – Application Programming Interface

Μία διεπαφή προγραμματισμού εφαρμογών (API), είναι μια διεπαφή υπολογιστών που καθορίζει τις αλληλεπιδράσεις μεταξύ πολλαπλών διαμεσολαβητών λογισμικού. Είναι αυτή που καθορίζει τις κλήσεις και τα αιτήματα που μπορούν να γίνουν, πώς μπορούν να γίνουν, με ποια μορφή θα σταλούν τα δεδομένα που ζητήθηκαν από αυτά τα αιτήματα, καθώς και τις συμβάσεις που πρέπει να ακολουθούν. Επιπλέον, μπορεί να προσφέρει μηχανισμούς, ώστε να μπορεί να γίνει επέκταση της υπάρχουσας λειτουργικότητας με διαφορετικούς τρόπους και σε διαφορετικό βαθμό. Ένα API μπορεί να έχει δημιουργηθεί για μόνο μια συγκεκριμένη χρήση, ανάλογα την λειτουργία που

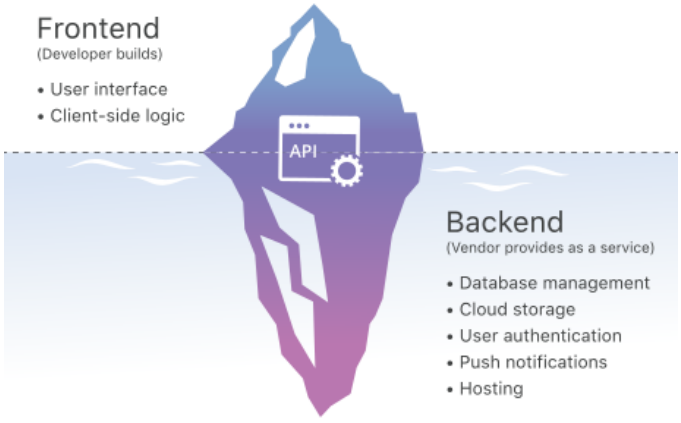
θέλει να παρέχει ή μπορεί να σχεδιαστεί με βάση κάποιο βιομηχανικό πρότυπο, ώστε να εξασφαλίζεται η διαλειτουργικότητα του [22].

Διεπαφή Χρήστη – User Interface

Διεπαφή χρήστη (User Interface), είναι το μέσο με το οποίο ένας χρήστης χειρίζεται και ελέγχει μία εφαρμογή. Ο καλός σχεδιασμός της διεπαφής χρήστη, μπορεί να οδηγήσει σε μία καλή εμπειρία χρήσης της εφαρμογής, επιτρέποντας στο άτομο να αλληλεπιδρά με την εφαρμογή με έναν φυσικό και διαισθητικό τρόπο [23].

Backend as a Service

Το Backend as a Service (BaaS) είναι ένα μοντέλο υπηρεσίας cloud, στο οποίο οι προγραμματιστές αναθέτουν σε έναν πάροχο, τις υπηρεσίες παρασκηνίου μιας εφαρμογής ιστού ή κινητού, ώστε να εστιάσουν στον σχεδιασμό και την ανάπτυξη της εφαρμογής εξυπηρετητή (Client Side app). Οι υπηρεσίες αυτές μπορεί να είναι η εξουσιοδότηση του χρήστη (User Authentication), η διαχείριση βάσης δεδομένων, καθώς και η διαχείριση ειδοποιήσεων αν έχουμε να κάνουμε με εφαρμογές κινητών [24]. Στην *Εικόνα 2.2* βλέπουμε αναλυτικά ποιες υπηρεσίες παρασκηνίου μπορεί να προσφέρει ένα πάροχος.



Εικόνα 2.2: Backend as a Service [24]

Serverless

Η Serverless είναι μια μέθοδος παροχής υπηρεσιών backend με βάση τον όγκο χρήσης τους. Ένας serverless πάροχος, δίνει την δυνατότητα στους προγραμματιστές να γράψουν και να αναπτύξουν κώδικα χωρίς να χρειάζεται να διαχειριστούν την υποκείμενη υποδομή του διακομιστή και να κάνουν αλλαγές για την επεκτασιμότητα του σε περίπτωση αυξημένου φόρου εργασίας. Η χρέωση για serveless υπηρεσίες γίνεται ανάλογα με το ποσοστό χρήσης και την υπολογιστική δύναμη που χρησιμοποιούν οι εφαρμογές [25].

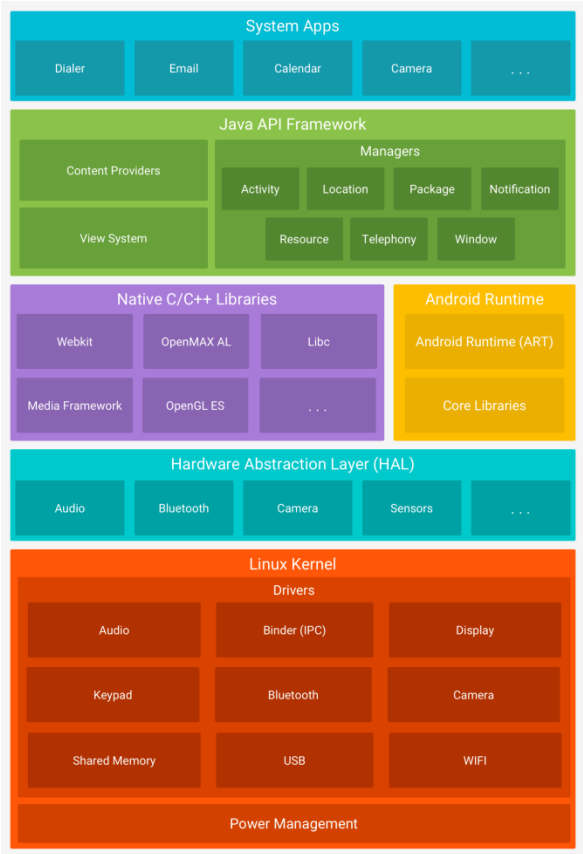
2.2 Περιβάλλον Android

Το Android [5] είναι ένα λειτουργικό σύστημα ανοιχτού κώδικα βασισμένο στην στοίβα λογισμικού Linux, που δημιουργήθηκε για ένα ευρύ φάσμα συσκευών, με έμφαση στις κινητές συσκευές με οθόνη αφής. Επιτρέπει στους κατασκευαστές λογισμικού να αναπτύξουν εφαρμογές με τις γλώσσες προγραμματισμού java [6] και Kotlin [7] που αναπτύχθηκε από την εταιρεία JetBrains.

Η αρχιτεκτονική της Android πλατφόρμας αποτελείτε από μία στοίβα λογισμικού 6 επιπέδων [8] (*Εικόνα 2.3*). Στο πρώτο επίπεδο έχουμε το Linux Kernel που αποτελεί την βάση του Android. Η χρήση του καθιστά εφικτή την εκμετάλλευση σημαντικών χαρακτηριστικών ασφαλείας που έχει το Linux, καθώς και επιτρέπει στους κατασκευαστές να δημιουργήσουν ασφαλείς και αξιόπιστους για περιφερειακές συσκευές. Στο επόμενο επίπεδο έχουμε το Hardware Abstraction Layer (HAL) . Το HAL παρέχει τις κατάλληλες διεπαφές για την επικοινωνία του υλικού της συσκευής με τα υψηλότερα επίπεδα της στοίβας. Αποτελείτε από πολλές βιβλιοθήκες, η κάθε μια από τις οποίες υλοποιεί μία διεπαφή για ένα συγκεκριμένο κομμάτι του υλικού, όπως η κάμερα ή το Bluetooth. Στο επόμενο επίπεδο της στοίβας έχουμε το Android Runtime (ART) και τις Native C/C++ βιβλιοθήκες. Το ART έχει δημιουργηθεί για να τρέχει πολλαπλά Virtual Machines (VM) σε συσκευές με περιορισμένη μνήμη. Για συσκευές Android έκδοσης 5.0 και μετά, κάθε εφαρμογή τρέχει μέσα στην δική της διαδικασία και με το δικό της ART. Οι Native C/C++ βιβλιοθήκες παρέχουν διεπαφές για την χρήση τους από τις εφαρμογές, όπως είναι η OpenGL ES που προσφέρει υποστήριξη για την διαχείριση 2D και 3D γραφικών από την εφαρμογή. Ένα επίπεδο πιο πάνω έχουμε το Java API Framework μέσω του οποίου η εφαρμογή έχει πρόσβαση στο σύνολο των λειτουργιών του Android

λειτουργικού. Αυτά τα API αποτελούν τα βασικά στοιχεία για την δημιουργία των εφαρμογών. Τέλος, έχουμε τις εφαρμογές συστήματος (System Apps) , που είναι οι βασικές εφαρμογές που υπάρχουν προ εγκατεστημένες στο λειτουργικό, όπως εφαρμογής διαχείρισης email, εφαρμογή για sms, εφαρμογή ημερολογίου και πολλές άλλες.

Μία Android εφαρμογή [5] είναι ένα λογισμικό που είναι σχεδιασμένο να τρέχει σε συσκευές ή σε προσομοιωτές Android. Επίσης ο όρος android εφαρμογή μπορεί να αναφέρεται και σε ένα αρχείο APK που σημαίνει πακέτο android (Android Package). Αποτελεί ένα συμπιεσμένο αρχείο zip που περιέχει κώδικα και πόρους της εφαρμογής, καθώς και τα μετα-δεδομένα της. Οι εφαρμογές συνήθως διανέμονται μέσω αγορών εφαρμογών ή αλλιώς app markets, όπως είναι το Google Play [9] ή μπορούν να εγκατασταθούν μέσω σύνδεσης USB από το APK αρχείο τους.



Εικόνα 2.3: Η Στοίβα λογισμικού Android. [8]

ΚΕΦΑΛΑΙΟ 3

ΤΕΧΝΟΛΟΓΙΚΗ ΕΠΙΣΚΟΠΙΣΗ

3.1 Περιβάλλον ανάπτυξης εφαρμογών Android

3.1.1 Android Studio

Το Android studio [10] (*Εικόνα 3.1*) είναι το ολοκληρωμένο προγραμματιστικό περιβάλλον (IDE) για το λειτουργικό σύστημα Android της Google, που δημιουργήθηκε από την εταιρία Jetbrains, με βάση το IntelliJ IDEA λογισμικό της. Το Android Studio εστιάζει στον προγραμματισμό εφαρμογών για Android και είναι συμβατό με τα λειτουργικά συστήματα Linux, Windows και MacOS. Προσφέρει πληθώρα λειτουργιών για την βελτιστοποίηση της παραγωγικότητας κατά την ανάπτυξη της εφαρμογής. Μερικές από τις λειτουργίες του είναι:

- Το ευέλικτο σύστημα μεταγλώττισης και εκτέλεσης εφαρμογών Gradle
- Ένα ενοποιημένο περιβάλλον ανάπτυξης για όλες τις συσκευές Android
- Συγχρονισμός αλλαγών του κώδικα με την εφαρμογή ακόμα και όταν τρέχει το πρόγραμμα
- Εκτενή εργαλεία και frameworks
- Εσωτερικά εργαλεία για την διαχείριση των πλατφόρμων Cloud της Google

Εικόνα 3.1: Logo Android Studio [10]

3.1.2 Android SDK

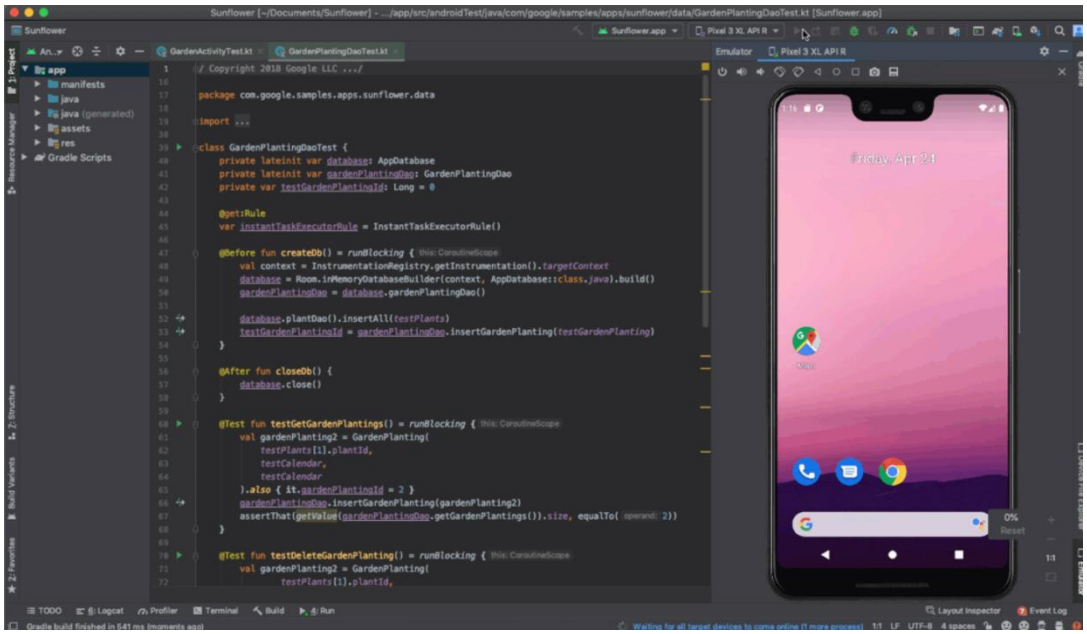
Το κιτ ανάπτυξης λογισμικού Android (SDK) [11] αποτελείτε από ένα σετ εργαλείων ανάπτυξης για εφαρμογές Android. Η συλλογή αυτών των εργαλείων αποτελείται από έναν αποσφαλματωτή (Debugger), βιβλιοθήκες, emulators καθώς και μία πληθώρα δειγμάτων κώδικα και βοηθητικές πληροφορίες για την ανάπτυξη εφαρμογών. Τα λειτουργικά που το υποστηρίζουν είναι τα Linux, Windows και MacOS.

Το Android SDK αναβαθμίζει τις εκδόσεις του μαζί με την γενική αναβάθμιση του Android. Τέλος, υποστηρίζει και παλαιότερες εκδώσεις του Android για παλαιότερες συσκευές.

3.1.3 Android Emulators

Ένας Android Emulator [12, 30] (*Εικόνα 3.2*) μπορεί να προσομοιώσει το περιβάλλον και την λειτουργία μιας Android συσκευής στον υπολογιστή. Ο Emulator παρέχει σχεδόν όλες τις λειτουργίες που έχει μία Android συσκευή. Τέτοιες λειτουργίες είναι η προσομοίωση εισερχόμενων κλήσεων και μηνυμάτων (SMS), δυνατότητα χρήσης και καθορισμού της γεωγραφικής τοποθεσίας της συσκευής, προσομοίωση διαφορετικών ταχυτήτων δικτύου καθώς και προσομοίωση αισθητήρα γυροσκοπίου, κάμερας και άλλων λειτουργιών. Τέλος, ο χρήστης μπορεί να χρησιμοποιήσει το Google play και να κατεβάσει τις εφαρμογές του όπως σε μία κανονική συσκευή Android.

Η χρήση του Emulator συνιστάται για την δοκιμή των εφαρμογών που αναπτύσσονται σε Android, καθώς τις περισσότερες φορές είναι πιο γρήγορη και πιο εύκολη η διαδικασία, από το να γίνει η δοκιμή σε φυσικές συσκευές. Για παράδειγμα, καθιστά πιο γρήγορη την μεταφορά δεδομένων κατά την ανάπτυξη και την δοκιμή, σε σύγκριση με μία φυσική συσκευή που είναι συνδεδεμένη με USB στον υπολογιστή. Επίσης, παρέχουν προκαθορισμένες ρυθμίσεις για μια πληθώρα συσκευών, όπως κινητά, tablet, έξυπνες τηλεοράσεις και γενικά έξυπνες συσκευές που κάνουν χρήση του λειτουργικού Android.



Εικόνα 3.2: Android phone emulator στο Android Studio [12]

Κάθε Emulator που χρησιμοποιείται στο περιβάλλον ανάπτυξης του Android Studio χρησιμοποιεί μια εικονική συσκευή Android (AVD – Android Virtual Device). Με την AVD καθορίζονται τα χαρακτηριστικά του συγκεκριμένου Emulator, όπως ποιά έκδοση Android έχει, τις δυνατότητες του υλικού της συσκευή που προσομοιώνεται όπως RAM, χώρο αποθήκευσης, πυρήνες επεξεργαστή και άλλες. Μπορούμε να καθορίσουμε πολλαπλές AVD με διαφορετικά χαρακτηριστικά για την καλύτερη δοκιμή της εφαρμογής σε διαφορετικές συσκευές. Κάθε AVD αποτελεί ανεξάρτητη συσκευή, με τον δικό της προσωπικό χώρο αποθήκευσης για τα δεδομένα του χρήστη, την δική της κάρτα SD για αποθηκευτικό χώρο καθώς και πολλά άλλα. Όλες οι AVD δημιουργούνται και σώζονται στον AVD Manager. Εκεί ο προγραμματιστής μπορεί να επιλέξει να ξεκινήσει την συσκευή ή συσκευές που έχει δημιουργήσει για δοκιμές, καθώς και να αλλάξει ή να διαγράψει τις AVD που έχει δημιουργήσει. Στην *Εικόνα 3.3* μπορούμε να δούμε την διεπαφή χρήστη που προσφέρει ο AVD Manager στο Android Studio.

Εικόνα 3.3: Android Virtual Device Manager [12]

3.2 Backend as a service - Firebase

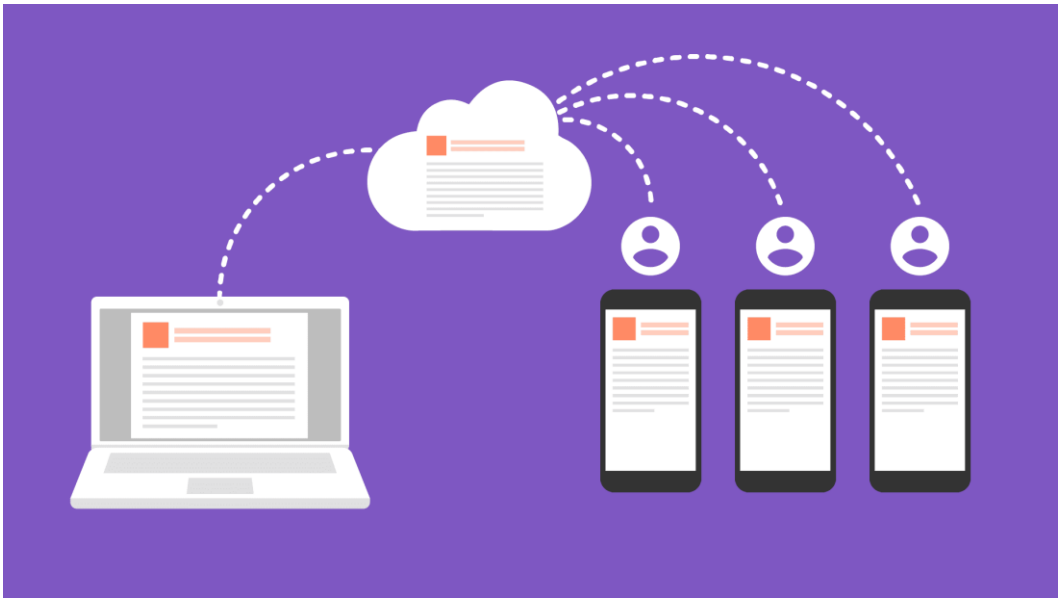
Εικόνα 3.4: Firebase Logo [13]

Το Firebase [13] (*Εικόνα 3.4*) είναι μία πλατφόρμα που αναπτύχθηκε από την Google για την δημιουργία εφαρμογών κινητού και ιστού. Αρχικά ήταν ανεξάρτητη εταιρία που ξεκίνησε το 2011 και αγοράστηκε από την Google το 2014. Προσφέρει μία πληθώρα λειτουργιών που βοηθούν στην γρηγορότερη ανάπτυξη εφαρμογών και στην αυτοματοποιημένη κλιμάκωση του διακομιστή της εφαρμογής με την αύξηση των χρηστών. Παρέχει λειτουργίες όπως εξουσιοδότηση χρηστών στην εφαρμογή, βάσεις δεδομένων πραγματικού χρόνου, συναρτήσεις Cloud για την δημιουργία της λογικής στην πλευρά του server, χωρίς να χρειάζεται ο προγραμματιστής να διαχειρίζεται τον server, καθώς και πολλές άλλες λειτουργίες που στοχεύουν στην serverless ανάπτυξη της εφαρμογής.

3.2.1 Firebase Realtime Database

Η Firebase Realtime database [14] είναι μία NoSQL βάση δεδομένων στο υπολογιστικό νέφος (Cloud). Τα δεδομένα έχουν την δυνατότητα να συγχρονιστούν σε όλους τους εξυπηρετητές σε πραγματικό χρόνο (*Εικόνα 3.5*), ενώ ο συγχρονισμός μπορεί να γίνει ακόμα και αν η εφαρμογή είναι εκτός σύνδεσης. Τα δεδομένα σώζονται σε μορφή JSON (JavaScript Object Notation) και συγχρονίζονται αυτόματα σε κάθε συνδεδεμένο

εξυπηρετητή. Η βάση μπορεί κατευθείαν να συνδεθεί με τους εξυπηρετητές της εφαρμογής μέσω API χωρίς να χρειάζεται η διαχείριση διακομιστή εφαρμογής, χρησιμοποιώντας το Serverless μοντέλο. Η ασφάλεια και η επικύρωση των δεδομένων γίνεται μέσω των κανόνων ασφάλειας της Realtime Database, που μπορούν να γραφτούν στον πίνακα ελέγχου της βάσης στην πλατφόρμα Firebase. Η σύνταξη τους θυμίζει την γλώσσα JavaScript και εκτελούνται κάθε φορά που κάτι γράφεται ή διαβάζεται στην βάση.



Εικόνα 3.5: Realtime database diagram [14]

3.2.2 Firebase Authentication

Το Firebase Authentication [15] προσφέρει υπηρεσίες σε επίπεδο διακομιστή για την εξουσιοδότηση των χρηστών στην εφαρμογή. Παρέχει SDK και βιβλιοθήκες διεπαφής χρήστη για την καλύτερη ανάπτυξη μηχανισμού εξουσιοδότησης χρηστών. Υποστηρίζει εξουσιοδότηση με την χρήση ηλεκτρονικού ταχυδρομείου και κωδικού πρόσβασης, μέσω αριθμού τηλεφώνου καθώς και μέσω third party authentication της Google, Facebook, Twitter και Github (*Εικόνα 3.6*).

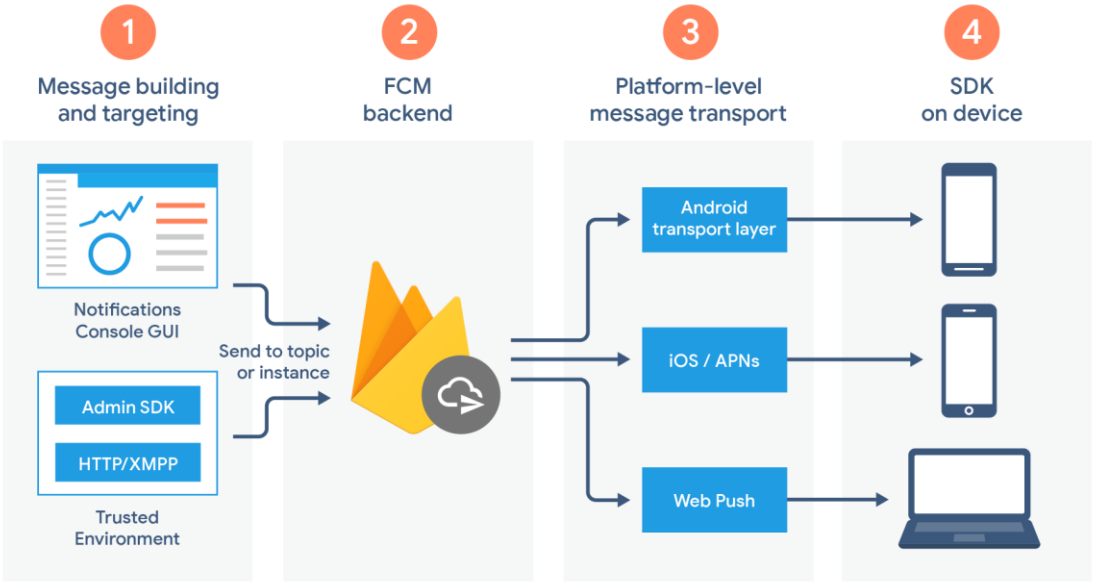
Το Firebase Authentication μπορεί να ενσωματωθεί και με τις υπόλοιπες υπηρεσίες του Firebase, όπως για παράδειγμα την βάση δεδομένων για την αποθήκευση των

στοιχείων του χρήστη της εφαρμογής. Επιπροσθέτως, παρέχει την δυνατότητα σύνδεσης συστήματος εξουσιοδότησης που έχει φτιαχτεί από τον εκάστοτε προγραμματιστή και να μπορεί να συνδεθεί και με τις υπόλοιπες υπηρεσίες του Firebase. Τέλος, δίνει την επιλογή ανώνυμης σύνδεσης του χρήστη στην εφαρμογή, καθώς και να μπορεί να συνδεθεί με διαπιστευτήρια και να συνεχίσει από εκεί που άφησε την εφαρμογή κατά την χρήση στην ανώνυμη κατάσταση.

Εικόνα 3.6: Τρόποι εξουσιοδότησης των χρηστών [15]

3.2.3 Firebase Cloud Messaging

Το Firebase Cloud Messaging (FCM) [16], είναι μια λειτουργία που επιτρέπει την ανταλλαγή μηνυμάτων μεταξύ πολλαπλών πλατφόρμων με μηδενικό κόστος (*Εικόνα 3.7*). Με την χρήση του FCM μπορούν να σταλούν μηνύματα ανακοινώσεων στους εξυπηρετητές για την συνεχή αλληλεπίδραση και ενημέρωση σε ότι αφορά την εφαρμογή, ενημερώσεις σε περίπτωση αλλαγής δεδομένων στην βάση, όπως για παράδειγμα ενημέρωση ότι εστάλη νέο μήνυμα σε μία εφαρμογή συνομιλιών και πολλές άλλες λειτουργίες. Τα μηνύματα μπορούν να μεταφέρουν δεδομένα χωρητικότητας έως και 4 Kb στις εφαρμογές των χρηστών. Τα μηνύματα μπορούν να σταλούν μέσω του Firebase Admin SDK ή του πρωτόκολλου διακομιστή FCM και έχει πλήρη συνδεσιμότητα με τις υπόλοιπες υπηρεσίες του Firebase.



Εικόνα 3.7: Διάγραμμα αρχιτεκτονικής FCM. [31]

3.2.4 Firebase Cloud Functions

το Firebase Cloud Functions [17] είναι ένα serverless framework που δίνει την δυνατότητα στον χρήστη να εκτελεί κώδικα διακομιστή ως απάντηση σε γεγονότα που πυροδοτούνται από τις λειτουργίες του Firebase ή από αιτήματα HTTP. Η γλώσσα προγραμματισμού που χρησιμοποιείται στο συγκεκριμένο framework είναι η JavaScript ή το συντακτικό υπερσύνολο της, η TypeScript. Ο κώδικας αποθηκεύεται στο Cloud της Google και εκτελείται σε ένα διαχειριζόμενο περιβάλλον, χωρίς να χρειάζεται την ανάγκη επίβλεψης η κλιμάκωσης σε μεγαλύτερο διακομιστή αν υπάρξει μεγαλύτερο φόρτο εργασίας. Ο τρόπος που λειτουργεί είναι, όταν τελειώσει η ανάπτυξη του κώδικα, γίνεται deploy στους διακομιστές της Google και είναι έτοιμος για χρήση.

3.3 Version Control

3.3.1 Git

Το Git [18] είναι ένα κατανεμημένο σύστημα ελέγχου εκδόσεων που παρέχεται δωρεάν και είναι ανοιχτού κώδικα. Σχεδιάστηκε για τον έλεγχο των αλλαγών του κώδικα ενός προγράμματος κατά την ανάπτυξη του. Προσφέρει την δυνατότητα εύκολης και γρήγορης διαχείρισης του κώδικα και των εκδόσεων του από πολλαπλούς

προγραμματιστές. Εστιάζει κυρίως στην ταχύτητα, στην ακεραιότητα των δεδομένων καθώς και στην υποστήριξη για κατανεμημένες ροές.

Η ανάπτυξη του Git ξεκίνησε το 2005, όταν μια ομάδα προγραμματιστών που δούλευαν στον πυρήνα του Linux, σταμάτησαν την χρήση του συστήματος ελέγχου εκδόσεων Bitkeeper, θέλοντας ένα πιο γρήγορο και κατανεμημένο σύστημα. Αυτή η ομάδα προγραμματιστών, κυρίως με τον Linus Torvals, δημιουργό του λειτουργικού Linux, δημιούργησαν το Git για τις ανάγκες διαχείρισης του πυρήνα των Linux από πολλαπλούς ανθρώπους με πολύ μικρή χρονική καθυστέρηση αλλαγών.

3.3.2 Github

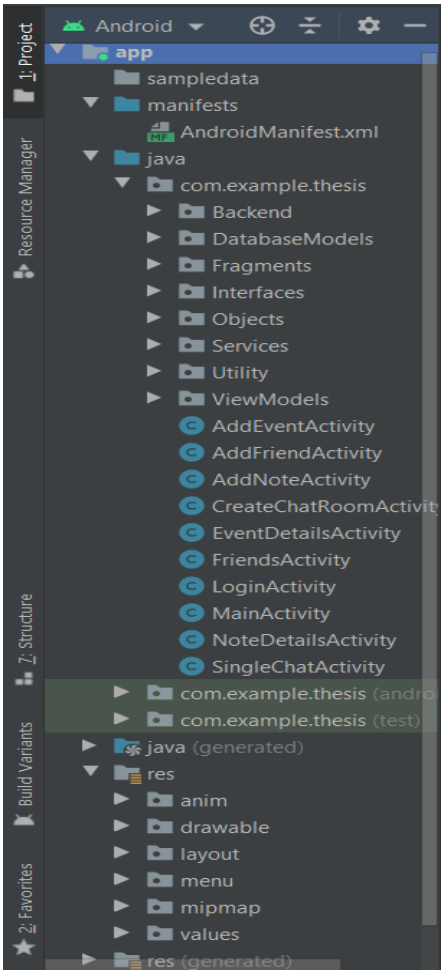
Το Github [19] είναι πάροχος φιλοξενίας για ανάπτυξη λογισμικού και εκδόσεων ελέγχου χρησιμοποιώντας Git. Προσφέρει τις λειτουργίες κατανεμημένου ελέγχου εκδόσεων και διαχείριση πηγαίου κώδικα που έχει το Git, με επιπλέον δικές του λειτουργίες όπως παρακολούθηση σφαλμάτων, διαχείριση εργασιών, συνεχή ενσωμάτωση και πολλά άλλα. Δημιουργήθηκε το 2008 από τους Chris Wanstrath, P. J. Hyett, Tom Preston-Werner και Scott Chacon με την γλώσσα προγραμματισμού Ruby και το framework Ruby on Rails. Το 2018 αγοράστηκε από την Εταιρία Microsoft. Στην πλατφόρμα του Github ο χρήστης μπορεί να έχει πρόσβαση και να διαχειριστεί projects με την διεπαφή τερματικού του Git. Ακόμα δίνει την δυνατότητα στους χρήστες να βρουν δημόσια repositories άλλων προγραμματιστών και εταιριών. Επιπλέον, προσφέρει και κάποιες λειτουργίες κοινωνικής δικτύωσης όπως ακόλουθους, κοινωνικά feeds και γραφήματα που δείχνουν πως δουλεύουν οι προγραμματιστές στα repositories τους.

ΚΕΦΑΛΑΙΟ 4

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

4.1 Δομή της εφαρμογής στο Android Studio

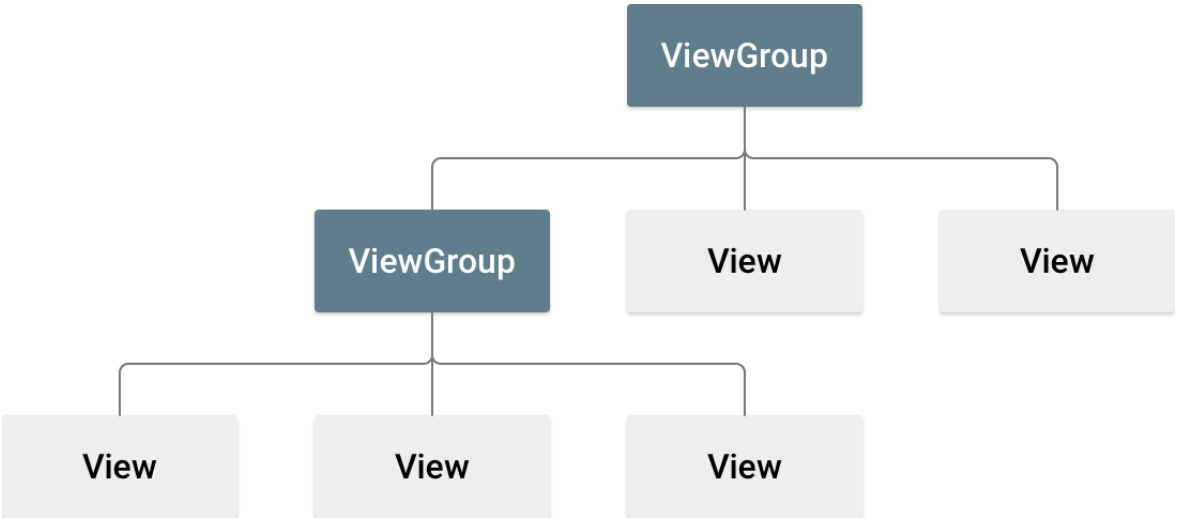
Κάθε project στο Android Studio [10], περιέχει όλα όσα καθορίζουν τον χώρο εργασίας του προγραμματιστή όταν δημιουργεί μια εφαρμογή. Μπορεί να περιέχει τον πηγαίο κώδικα της, τα πολυμέσα που χρησιμοποιούνται για την ανάπτυξη της, τον κώδικα δοκιμών, ακόμα και τα αρχεία διαμόρφωσης της τελικής εφαρμογής πριν την έκδοσή της. Όταν ξεκινάει ένα καινούριο project, το Android Studio δημιουργεί την απαραίτητη δομή για όλα τα αρχεία της εφαρμογής. Στην *Εικόνα 4.1* βλέπουμε την δομή των αρχείων της εφαρμογής που δημιουργήθηκε για την συγκεκριμένη διπλωματική.



Εικόνα 4.1: Δομή αρχείων εφαρμογής

4.1.1 Αρχεία XML

Η γλώσσα XML (Extensible Markup Language) [20], είναι μια γλώσσα σήμανσης, που περιέχει ένα σύνολο κανόνων για την ηλεκτρονική κωδικοποίηση κειμένων. Στις εφαρμογές Android, η XML χρησιμοποιείται για τον σχεδιασμό της διεπαφής με τον τελικό χρήστη. Τα XML αρχεία που χρησιμοποιούνται για την απεικόνιση της διεπαφής χρήστη, ονομάζονται αρχεία διάταξης (Layout Files). Όλα τα στοιχεία σε ένα αρχείο διάταξης, έχουν δημιουργηθεί χρησιμοποιώντας μια ιεραρχία από αντικείμενα View και View Group (*Εικόνα 4.2*). Ένα αντικείμενο View, συνήθως απεικονίζει μία οντότητα με την οποία ο χρήστης μπορεί να αλληλεπιδράσει. Παραδείγματα τέτοιων αντικειμένων είναι τα κουμπιά, τα πλαίσια εικόνων, τα πλαίσια εισαγωγής κειμένων και πολλά άλλα. Επιπρόσθετα, ένα αντικείμενο ViewGroup είναι ένα αντικείμενο που περιέχει αντικείμενα τύπου View και καθορίζει την δομή της διάταξης των αντικειμένων αυτών. Τα αντικείμενα ViewGroup, συνήθως ονομάζονται διατάξεις (Layouts) και μπορεί να είναι πολλών διαφορετικών ειδών και να προσφέρουν διαφορετικούς τρόπους διάταξης των αντικειμένων που περιέχουν. Μερικά παραδείγματα είναι το Linear Layout και το Constraint Layout [26].



Εικόνα 4.2: Ενδεικτική ιεραρχική δομή της διεπαφής με τον χρήστη [26]

Κομβικό ρόλο σε οποιαδήποτε εφαρμογή Android έχει το αρχείο Manifest. Το αρχείο αυτό έχει κωδικοποίηση XML και περιέχει τα δομικά συστατικά από τα οποία αποτελείται η εφαρμογή [27]. Ορίζει το όνομα του πακέτου της εφαρμογής, τα δομικά

στοιχεία της, όπως τα Activities και τα Fragments που περιέχει, τις άδειες που είναι απαραίτητες για την πρόσβαση σε προστατευμένα κομμάτια του συστήματος, καθώς και τις δυνατότητες του υλικού και του λογισμικού που χρειάζεται η εφαρμογή για να εγκατασταθεί στις συσκευές.

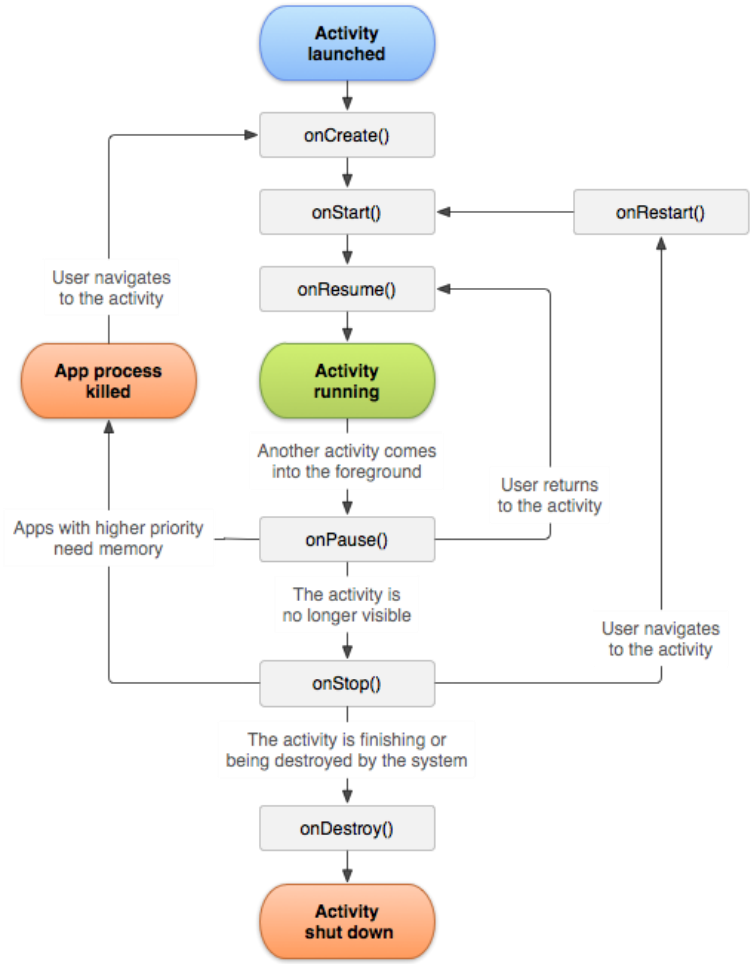
Τέλος, ένα άλλο είδος αρχείων που κωδικοποιούνται σε γλώσσα XML , είναι τα αρχεία τιμών. Αυτά τα αρχεία, περιέχουν τις αλφαριθμητικές τιμές όλων των χρωμάτων, των διαστάσεων αλλά και των κειμένων που χρησιμοποιούνται μέσα στην εφαρμογή. Η χρήση τους καθιστά πιο εύκολη την μαζική αλλαγή των εκάστοτε τιμών, ειδικά αν χρησιμοποιούνται σε πολλαπλά κομμάτια της εφαρμογής.

4.1.2 Αρχεία Java

Στην συγκεκριμένη εφαρμογή της διπλωματικής, επιλέχτηκε η γλώσσα Java σαν κύρια γλώσσα προγραμματισμού. Η Java [6] είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού, η οποία σχεδιάστηκε από την εταιρία πληροφορικής Sun Microsystems. Ο πηγαίος κώδικας της μεταγλωττίζεται σε κώδικα εικονικής μηχανής (Virtual machine code) ή σε bytecode. Αυτό την καθιστά ανεξάρτητη από το λειτουργικό εκτέλεσης, με αποτέλεσμα να μπορεί να μεταγλωττιστεί μία φορά και να εκτελείται σε πολλές πλατφόρμες. Στο Android, χρησιμοποιείται για την ανάπτυξη των δομικών στοιχεία της εφαρμογής. Τα κύρια δομικά στοιχεία μιας εφαρμογή Android είναι:

Activity

Είναι το κύριο δομικό στοιχείο μιας εφαρμογής, καθώς και υπεύθυνο για την δημιουργία ενός παραθύρου που θα φιλοξενήσει μια διεπαφή χρήστη της εφαρμογής και θα εκτελέσει κάποια από την λογική της. Ένα Activity μπορεί να βρεθεί σε τέσσερις καταστάσεις (*Εικόνα 4.3*). Η πρώτη κατάσταση είναι η ενεργή, στην οποία βρίσκεται συνήθως όταν ο χρήστης αλληλεπιδρά με αυτή. Η δεύτερη κατάσταση είναι η ορατή, στην οποία το Activity βρίσκεται συνήθως όταν δεν υπάρχει κάποια αλληλεπίδραση με τον χρήστη αλλά είναι διαθέσιμη σε αυτόν. Η τρίτη είναι η κρυφή ή σταματημένη κατάσταση. Εκεί το Activity έχει συγκαλυφθεί από ένα άλλο και δεν είναι πλέον ορατό στον χρήστη, αλλά κρατάει όλα του τα δεδομένα. Τέλος, έχουμε την κατάσταση καταστροφής, που πλέον το Activity απελευθερώνεται εντελώς από την μνήμη [28].



Εικόνα 4.3: Activity lifecycle [28]

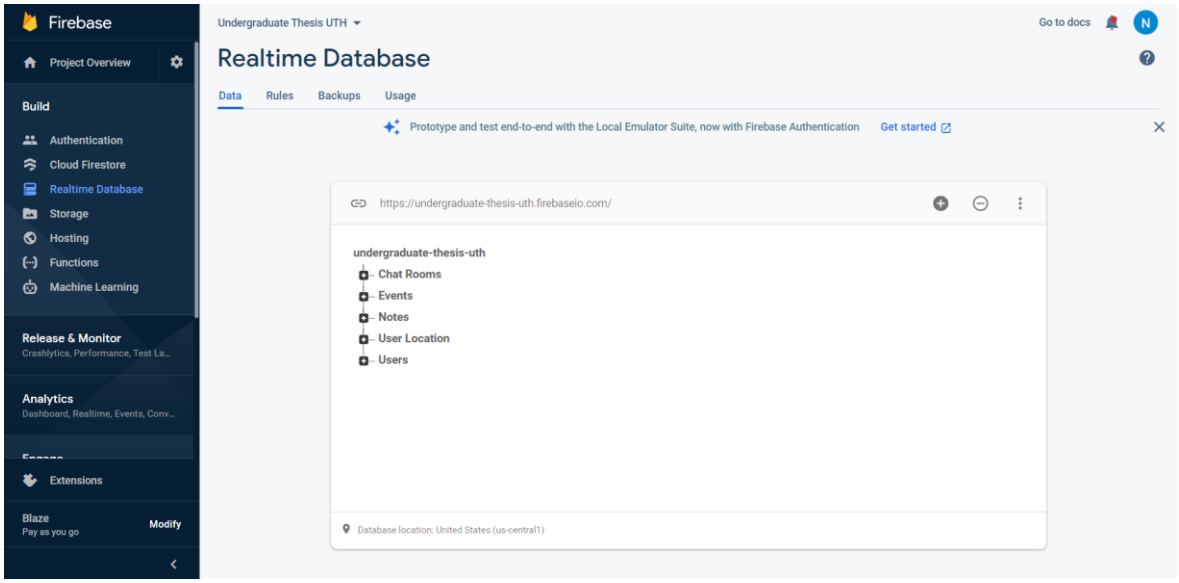
Fragment

Το Fragment αντιπροσωπεύει ένα τμήμα της διεπαφής χρήστη της εφαρμογής, που έχει την δυνατότητα να επαναχρησιμοποιείται. Κάθε Fragment καθορίζει και διαχειρίζεται την δική του διάταξη (Layout), τον δικό του κύκλο ζωής (Lifecycle) και χειρίζεται τα δικά του συμβάντα εισόδου (Input Events). Τα Fragments δεν μπορούν να σταθούν μόνα τους σε μια εφαρμογή, αλλά είναι αναγκαίο να φιλοξενούνται μέσα σε ένα Activity ή σε ένα άλλο Fragment [29].

4.2 Αρχιτεκτονική Realtime Database

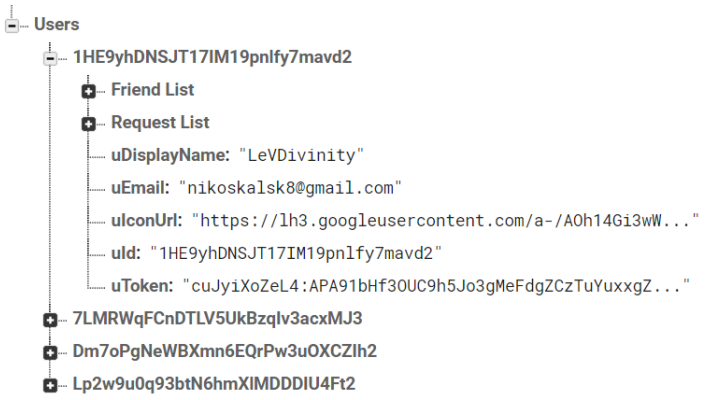
Όπως αναφερθήκαμε και πιο πάνω, για την εφαρμογή χρησιμοποιείται σαν βάση δεδομένων η Firebase Realtime Database, η οποία είναι μία NoSQL βάση. Τα δεδομένα στην βάση σώζονται σε μορφή JSON αντικειμένων. Στην σελίδα διαχείρισης της Realtime database, έχουμε ένα dashboard, όπου εκεί μπορούμε να δούμε και να διαχειριστούμε

τα δεδομένα που σώζονται στην βάση. Στην *Εικόνα 4.4* απεικονίζεται η οθόνη διαχείρισης της βάσης δεδομένων που μπορεί να δει ο προγραμματιστής στην πλατφόρμα Firebase.

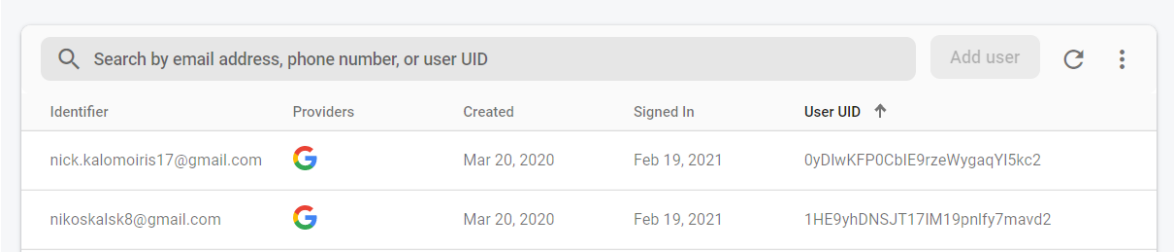


Εικόνα 4.4: Realtime database dashboard

Όταν ένας χρήστης συνδέεται για πρώτη φορά στην εφαρμογή, τα στοιχεία σύνδεσής του σώζονται στην συλλογή Users της βάση δεδομένων (*Εικόνα 4.5*) και στην υπηρεσία Authentication του Firebase (*Εικόνα 4.6*). Μόλις γίνει η σύνδεση, δίνεται σε κάθε χρήστη ένα μοναδικό id, το οποίο αποτελεί και το αναγνωριστικό του στην βάση. Εκτός απο το όνομα, το email και την εικόνα προφίλ του χρήστη, σώζεται και ένα μοδανικό token που αντιστοιχεί στην συσκευή του χρήστη. Αυτό το token, χρησιμεύει στην στοχευμένη αποστολή ειδοποιήσεων στην συσκευή του χρήστη, κατά την αποστολή μηνυμάτων σε μία συνομιλία. Τέλος, αποθηκεύονται και δύο επιπλέον λίστες στα δεδομένα του χρήστη. Η λίστα φίλων περιέχει τα id των φίλων, τα οποία έχουν προστεθεί από τον χρήστη, καθώς και η λίστα αιτημάτων φιλίας, η οποία περιέχει τα id των χρηστών που έχουν στείλει αίτημα φιλίας. Οι δύο αυτές λίστες είναι άδειες την πρώτη φορά που συνδέεται ο χρήστης.



Εικόνα 4.5: Users



Εικόνα 4.6: Users in Firebase Authentication Service

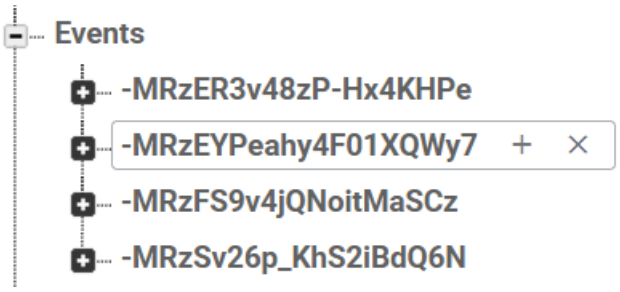
Εφόσον ολοκληρωθεί η σύνδεση, ζητείται από τον χρήστη να παραχωρήσει άδεια στην εφαρμογή, έτσι ώστε αυτή να μπορεί να χρησιμοποιήσει την τοποθεσία του. Στην συνέχεια, με την αποδοχή του χρήστη, η εφαρμογή λαμβάνει τη γεωγραφική τοποθεσία του και τη σώζει στην συλλογή User Location της βάσης δεδομένων. Όσον αφορά την αναγνώριση της τοποθεσίας που σώσαμε, χρησιμοποιείται το ίδιο id με αυτό που σώζονται τα δεδομένα του χρήστη στην συλλογή Users (*Εικόνα 4.7*). Επειδή η τοποθεσία του χρήστη θα αλλάζει ανά τακτά χρονικά διαστήματα, σώζεται και ένα αντίγραφο των στοιχείων του χρήστη μαζί με την τοποθεσία του, για την γρηγορότερη προσπέλαση των πληροφοριών κατά την επεξεργασία τους στην εφαρμογή.

Εικόνα 4.7: User Location

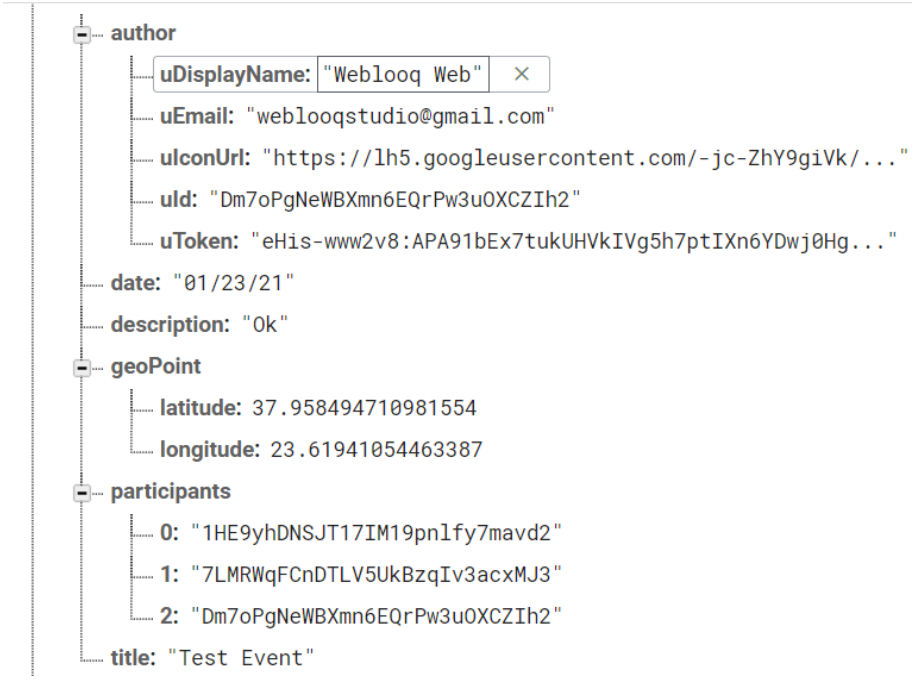
Μία από τις δυνατότητες που έχει ο χρήστης στην εφαρμογή, είναι να δημιουργήσει μία εκδήλωση ή μία σημείωση πάνω στον χάρτη, προσθέτοντας τις πληροφορίες, οι οποίες θα είναι ορατές στους φίλους που έχει στην εφαρμογή. Κατά την δημιουργία μίας εκδήλωσης ή σημείωσης, ζητείται από τον χρήστη να προσθέσει κάποια χαρακτηριστικά, όπως τίτλο και κείμενο πληροφοριών. Όταν τελειώσει αυτή την διαδικασία, διαλέγει ένα σημείο στο χάρτη, για να μετατρέψει την εκδήλωση ή την σημείωση σε marker και να είναι πλέον ορατή στους φίλους του. Εφόσον ο χρήστης επιλέξει το σημείο και την προσθήκη, τότε τα δεδομένα της εκδήλωσης ή της σημείωσης σώζονται στη συλλογή Events (*Εικόνα 4.8*) ή Notes (*Εικόνα 4.10*) αντίστοιχα. Κάθε εκδήλωση ή σημείωση, σώζεται με ένα τυχαίο id, που δημιουργείται κατά την προσθήκη της στην βάση δεδομένων και είναι μοναδικό.

Κάθε εκδήλωση που σώζεται στην βάση έχει τα εξής δεδομένα: τον τίτλο της εκδήλωσης, την περιγραφή της, την ημερομηνία που θα πραγματοποιηθεί, τα στοιχεία του δημιουργού της, μια λίστα με τα id των καλεσμένων καθώς και την γεωγραφική τοποθεσία που έχει καρφιτσωθεί πάνω στον χάρτη (*Εικόνα 4.9*).

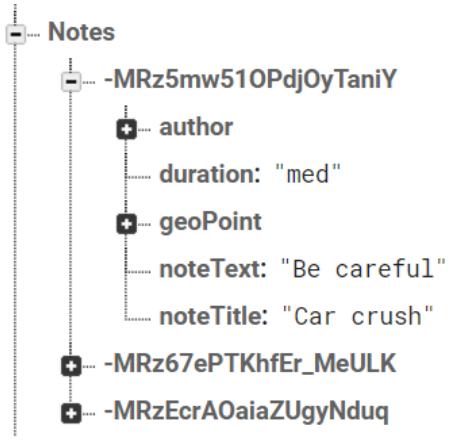
Με παρόμοια λογική σώζονται και τα δεδομένα των σημειώσεων στην βάση δεδομένων. Κάθε σημείωση σώζεται με τον τίτλο της, το κείμενό της, τα στοιχεία του δημιουργού της, την γεωγραφική της τοποθεσία πάνω στο χάρτη, καθώς και τον χρόνο που θα διαρκέσει πάνω σε αυτόν από την στιγμή της δημιουργία της (*Εικόνα 4.10*).



Εικόνα 4.8: Συλλογή Events

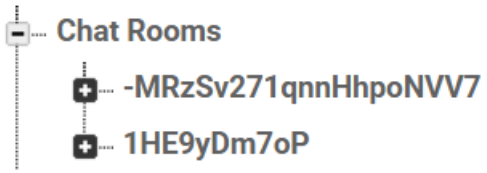


Εικόνα 4.9: Δεδομένα εκδήλωσης



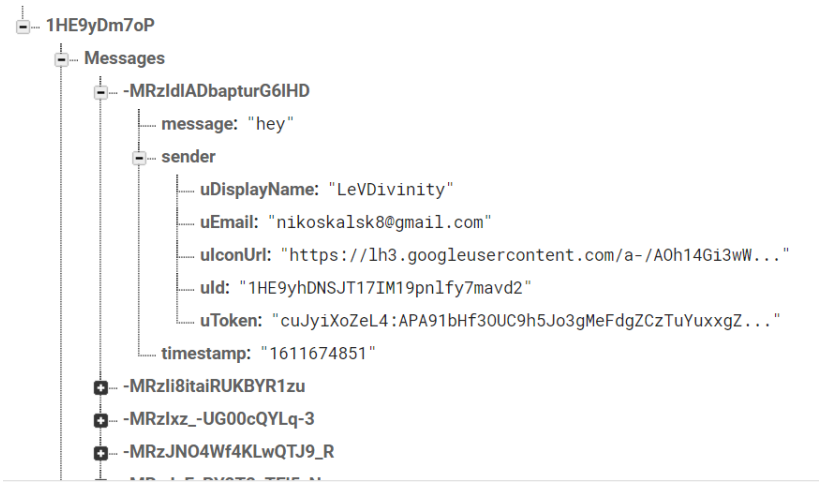
Εικόνα 4.10: Συλλογή Notes - Δεδομένα σημείωσης

Τέλος, ο χρήστης έχει την δυνατότητα να δημιουργήσει συνομιλίες και να ανταλλάξει μηνύματα με φίλους του μέσω αυτών. Κατά την δημιουργία μιας συνομιλίας, ο χρήστης επιλέγει ποιοι φίλοι του θα αποτελέσουν μέλη της και στην συνέχεια την δημιουργεί. Αυτή σώζεται στην συλλογή Chat Rooms της βάσης δεδομένων (*Εικόνα 4.11*). Κάθε chat room έχει το δικό του id, το οποίο δημιουργείται ανάλογα με το είδος της συνομιλίας. Αν υποθέσουμε πως έχουμε συνομιλία μεταξύ δύο ατόμων, θεωρείται ατομική και το id της παράγεται από την ένωση των 5 πρώτων γραμμάτων του id του κάθε μέλους της. Με αυτό τον τρόπο εξασφαλίζεται ένα ξεχωριστό id και δίνεται η δυνατότητα να ελεγχθεί αν ήδη υπάρχει η συνομιλία, ώστε να μην χρειαστεί να δημιουργηθεί πάλι. Στην περίπτωση μιας συνομιλίας τριών ή περισσοτέρων ατόμων, δεν χρειάζεται να εξασφαλιστεί ότι είναι μοναδική. Άρα η παραγωγή του id γίνεται τυχαία και σώζεται και αυτή στην βάση. Τα δεδομένα κάθε συνομιλίας, αποτελούνται από τα εξής στοιχεία: μία λίστα με τα μηνύματα της συνομιλίας, το όνομά της και μία λίστα που περιέχει τα στοιχεία όλων των μελών της (*Εικόνα 4.12*). Στην λίστα μηνυμάτων, κάθε μήνυμα σώζεται με ένα μοναδικό id και περιέχει το κείμενο του μηνύματος, τα στοιχεία του αποστολέα του και ένα timestamp που καθορίζει την ώρα και ημερομηνία αποστολής του (*Εικόνα 4.13*).



Εικόνα 4.11: Συλλογή Chat Rooms

Εικόνα 4.12: δεδομένα συνομιλίας



Εικόνα 4.13: Λίστα μηνυμάτων - δεδομένα μηνύματος

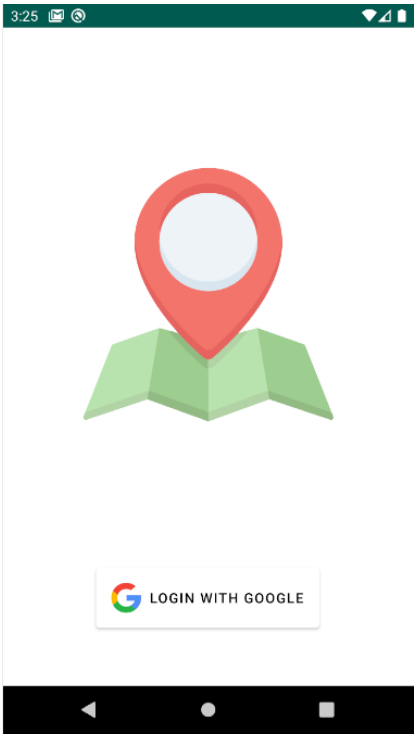
ΚΕΦΑΛΑΙΟ 5

ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ

Ο προσεκτικός σχεδιασμός της διεπαφής με τον τελικό χρήστη, αποτελεί σημαντικό παράγοντα για την επιτυχία κάθε εφαρμογής μέσα από την προσφορά της καλύτερης δυνατής εμπειρίας στο χρήστη. Μια απλή και κατανοητή αρχιτεκτονική της διεπαφής μπορεί να αναδείξει τις λειτουργίες της εφαρμογής και να ελκύσει το ενδιαφέρον των χρηστών.

5.1 Οθόνη σύνδεσης

Ανοίγοντας για πρώτη φορά την εφαρμογή, βλέπουμε την οθόνη σύνδεσής της. Εκεί, έχουμε το λογότυπο της εφαρμογής και ένα κουμπί, που πατώντας το, ο χρήστης μπορεί να συνδεθεί με τον Google λογαριασμό του στην εφαρμογή (*Εικόνα 5.1*). Μετά την σύνδεση του χρήστη, κατευθύνεται στην αρχική οθόνη της εφαρμογής.



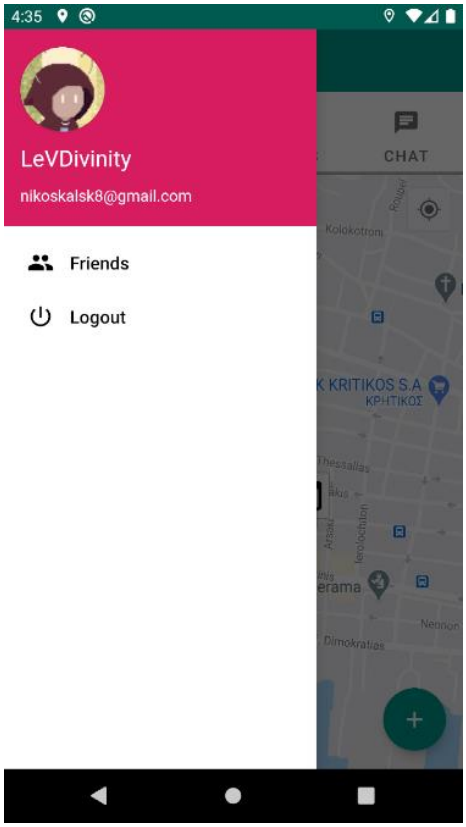
Εικόνα 5.1: Οθόνη σύνδεσης

5.2 Αρχική Οθόνη

Όταν τελειώσει η εξουσιοδότηση του χρήστη και πραγματοποιηθεί σύνδεση με τον λογαριασμό του, η εφαρμογή εισέρχεται στην αρχική οθόνη. Εκεί βλέπουμε ένα Activity με tab layout τεσσάρων καρτελών και την γραμμή εργαλείων στο πάνω μέρος της εφαρμογής. Σε αυτήν, υπάρχει το όνομα της εφαρμογής και ένα κουμπί hamburger που μας οδηγεί στο προφίλ χρήστη.

5.2.1 Οθόνη προφίλ χρήστη

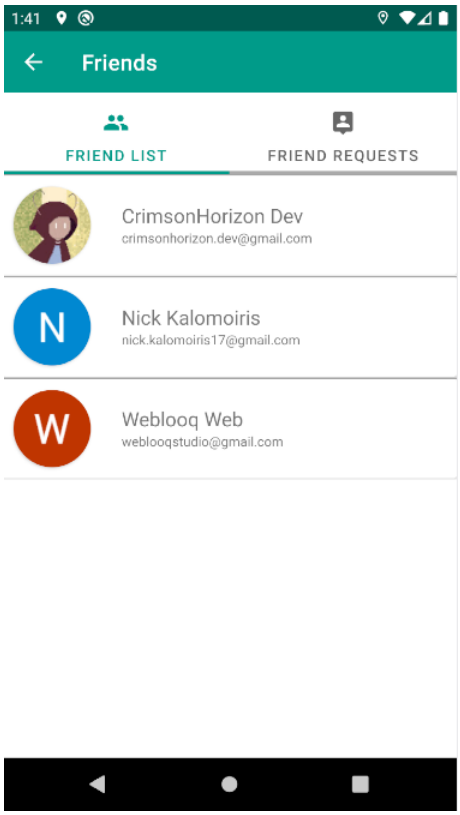
Το προφίλ του χρήστη εισέρχεται συρταρωτά από τα αριστερά προς τα δεξιά της οθόνης, έχοντας πλάτος όσο τα δύο τρίτα της. Η διάταξή του χωρίζεται σε δύο τμήματα. Στο πάνω μέρος έχουμε την κεφαλίδα (Header), με πληροφορίες του χρήστη, όπως η εικόνα προφίλ, το όνομα και το email του. Κάτω από την κεφαλίδα υπάρχει ένα μενού με δύο επιλογές. Στην πρώτη επιλογή έχουμε το κουμπί Friends που μας οδηγεί στην οθόνη των φίλων και στην δεύτερη το κουμπί Logout, για την αποσύνδεση από το προφίλ (*Εικόνα 5.2*).



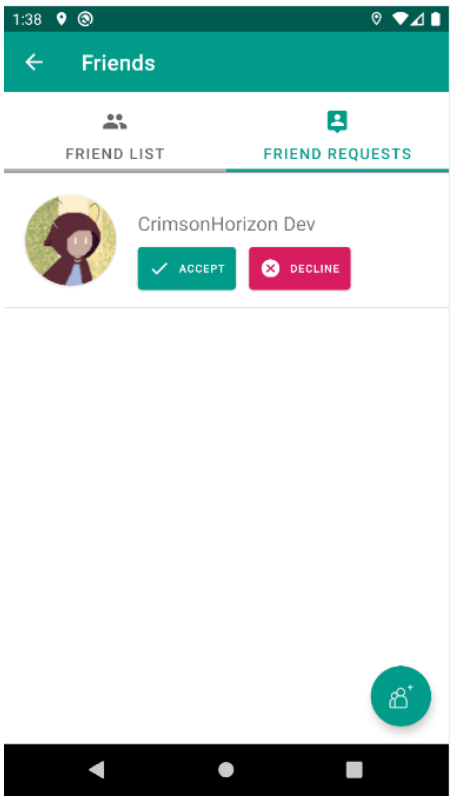
Εικόνα 5.2: Προφίλ χρήστη

5.2.2 Οθόνη φίλων

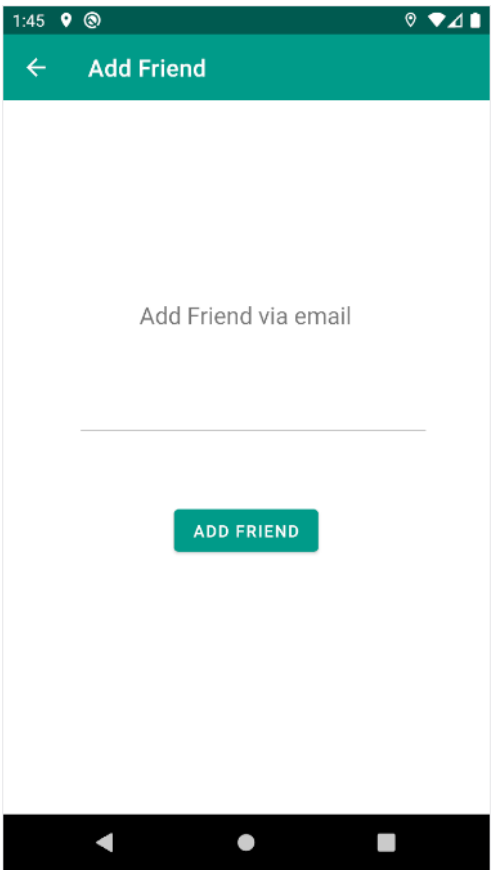
Επιλέγοντας το κουμπί Friends στο προφίλ χρήστη, μεταφερόμαστε στην οθόνη φίλων. Εκεί έχουμε ένα tab layout με δύο καρτέλες. Στην πρώτη καρτέλα βρίσκεται η λίστα των φίλων και στην δεύτερη, η λίστα με τα αιτήματα φιλίας που μας έχουν σταλεί. Κάθε στοιχείο της λίστας φίλων, έχει την εικόνα προφίλ του εκάστοτε φίλου και πληροφορίες όπως το όνομα και το email του (*Εικόνα 5.3*). Στην λίστα αιτημάτων, εμφανίζονται όλα τα αιτήματα φιλίας που δέχεται ο χρήστης. Έχει την επιλογή να αποδεχτεί το αίτημα φιλίας ή να το απορρίψει (*Εικόνα 5.4*). Στην οθόνη της λίστας αιτημάτων, κάτω δεξιά, είναι τοποθετημένο το κουμπί προσθήκης φίλων. Πατώντας το, εμφανίζεται η οθόνη προσθήκης φίλων, όπου πληκτρολογώντας το email του χρήστη που θέλουμε να προσθέσουμε, μπορούμε να του στείλουμε αίτημα φιλίας. Σε περίπτωση που δεν υπάρχει αυτός ο χρήστης στην εφαρμογή ή γράψαμε ένα κείμενο που δεν παραπέμπει σε email, μας βγάζει μήνυμα λάθους (*Εικόνα 5.5*).



Εικόνα 5.3: Λίστα φίλων



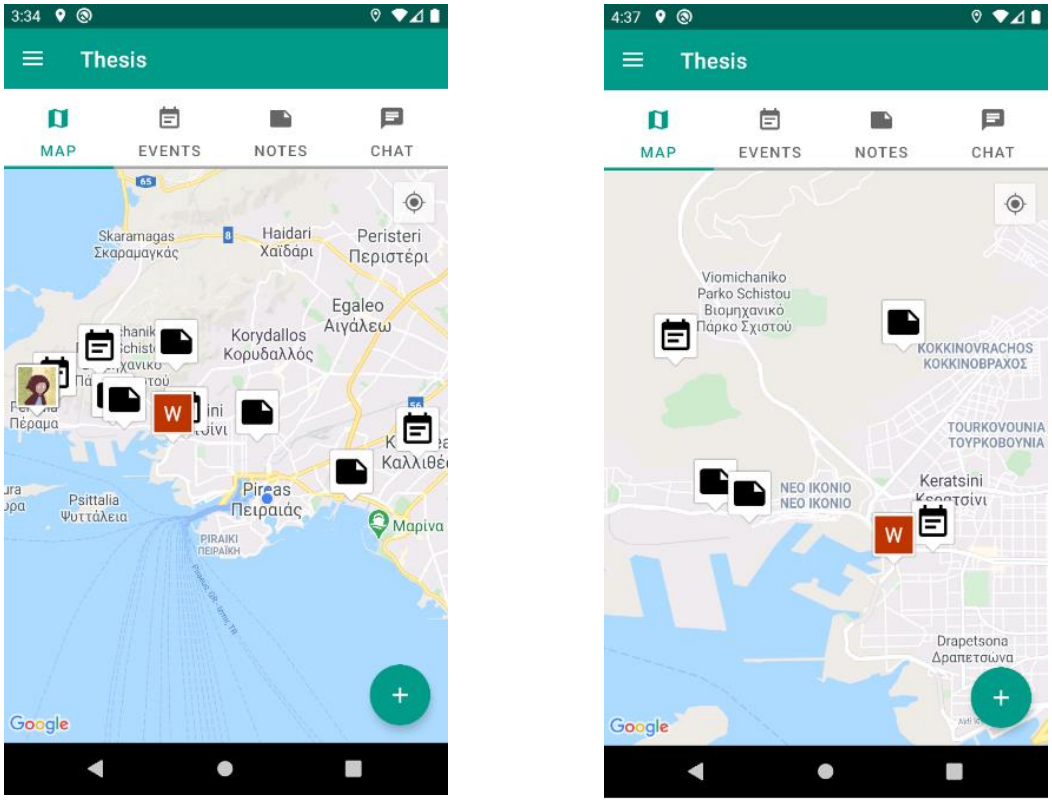
Εικόνα 5.4: Λίστα αιτημάτων φιλίας



Εικόνα 5.5: Οθόνη προσθήκης φίλου

5.2.3 Καρτέλα χάρτη

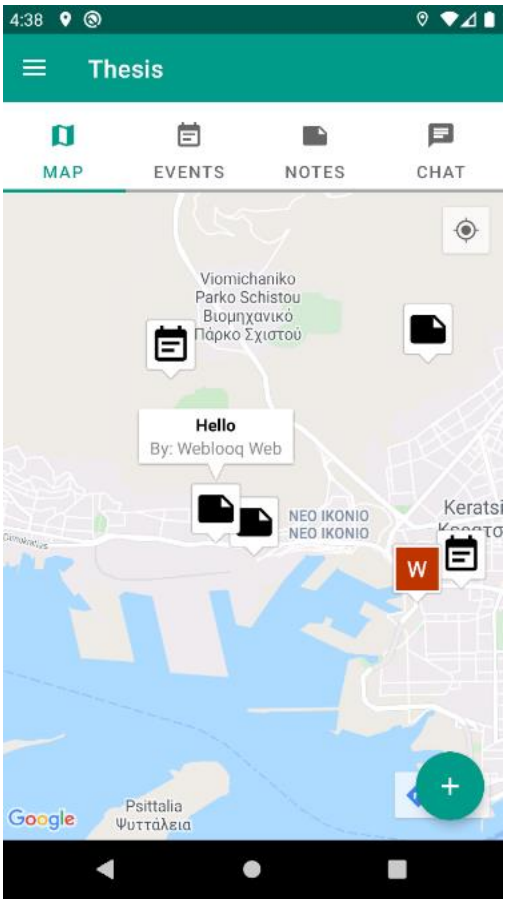
Η πρώτη καρτέλα που βλέπουμε όταν μπαίνουμε στην αρχική οθόνη, είναι η καρτέλα του χάρτη. Εκεί ο χρήστης μπορεί να δει τα markers των φίλων του πάνω στον χάρτη, καθώς και τα markers των εκδηλώσεων και των σημειώσεων που έχουν δημιουργηθεί. Η τοποθεσία του χρήστη απεικονίζεται με μία μπλε κουκίδα στον χάρτη. Τα markers των εκδηλώσεων έχουν σαν εικονίδιο ένα ημερολόγιο και τα markers των σημειώσεων ένα χαρτί τύπου “post-it”, έτσι ώστε ο χρήστης να είναι σε θέση να διακρίνει τις διαφορές. Τα markers των φίλων, έχουν σαν εικονίδιο την εικόνα προφίλ του Google λογαριασμού τους (*Εικόνα 5.6*).



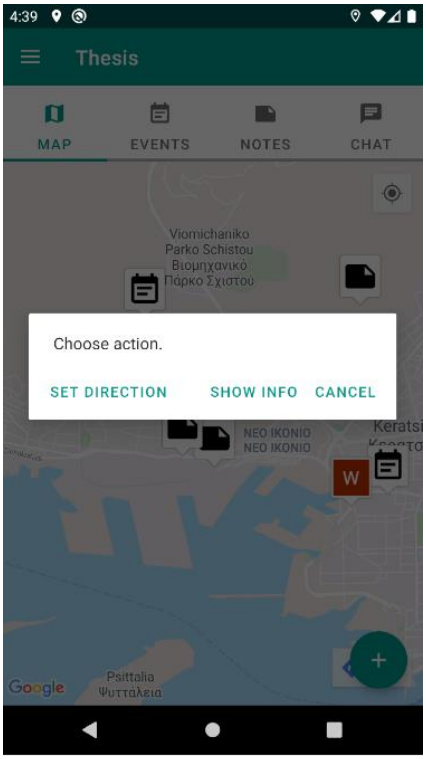
Εικόνα 5.6: Καρτέλα χάρτη

Ο χρήστης έχει την δυνατότητα να πλοηγηθεί και να μεγεθύνει ή ακόμα και να μικρύνει τον χάρτη, καθώς και να αλληλεπιδράσει με τα markers των φίλων, των εκδηλώσεων αλλά και των σημειώσεων. Εάν ο χρήστης επιλέξει τα markers των φίλων, ανοίγει ένα παράθυρο πληροφοριών, το οποίο περιέχει το όνομα και το email του εκάστοτε φίλου. Στο σημείο αυτό, αξίζει να διευκρινιστεί ότι τα markers των εκδηλώσεων και των σημειώσεων έχουν και μία επιπλέον λειτουργία από αυτά των φίλων. Πιο αναλυτικά, όταν ο χρήστης επιλέξει ένα marker εκδήλωσης ή σημείωσης, ανοίγει ένα παράθυρο πληροφοριών, παρόμοιο με αυτό του marker των φίλων, στο οποίο αναγράφεται τόσο ο τίτλος της εκάστοτε εκδήλωσης ή σημείωσης, όσο και το όνομα του χρήστη που την δημιούργησε (*Εικόνα 5.7*). Ωστόσο, αν επιλεχθεί το παράθυρο πληροφοριών των δύο αυτών marker, παρουσιάζεται στον χρήστη ένα Dialog Box με τρείς επιλογές: την επιλογή “Set Direction”, την επιλογή “Show Info” και την επιλογή “Cancel” (*Εικόνα 5.9*). Όταν ο χρήστης επιλέξει την “Set Direction”, του βγάζει άλλο ένα παράθυρο Dialog για να επιλέξει τον τρόπο μετακίνησης (περπατώντας ή με αυτοκίνητο) (*Εικόνα 5.8*). Αφού επιλέξει τον τρόπο μετακίνησης, σχεδιάζονται πάνω στον χάρτη οι τρείς γρηγορότερες

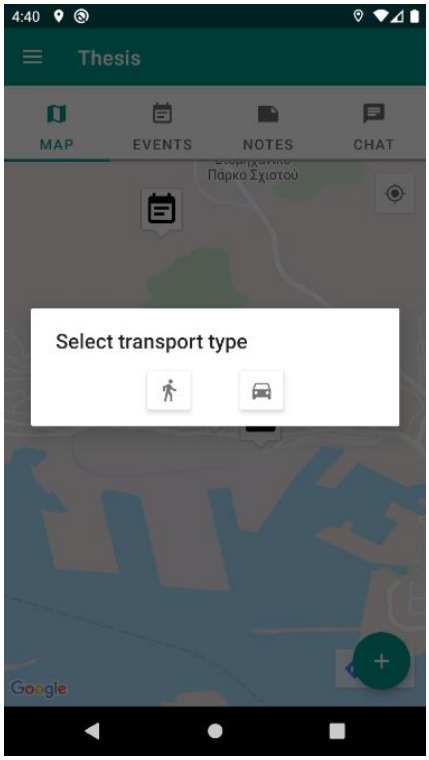
διαδρομές, από την τοποθεσία του χρήστη, στην τοποθεσία της εκδήλωσης ή σημείωσης που έγινε η επιλογή (*Εικόνα 5.10*).



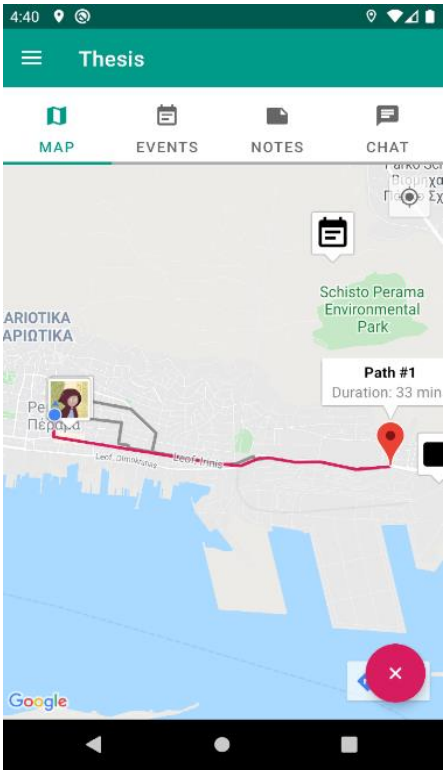
Εικόνα 5.7: Παράθυρο πληροφοριών Note



Εικόνα 5.8: Παράθυρο επιλογών marker

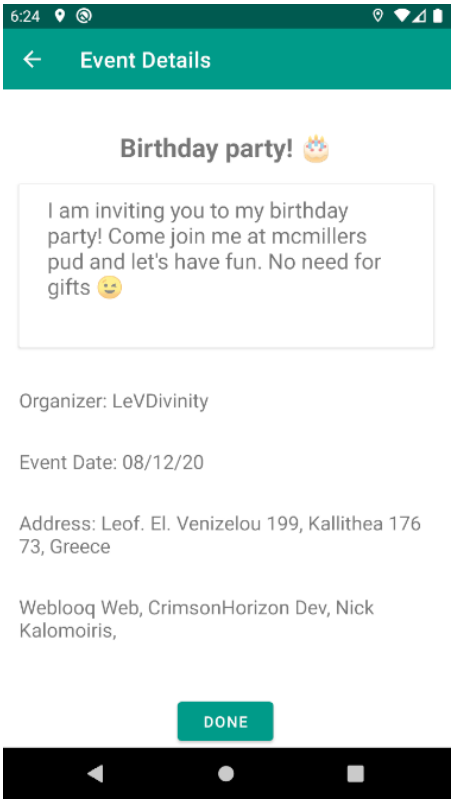


Εικόνα 5.9: Παράθυρο επιλογών τρόπου μεταφοράς

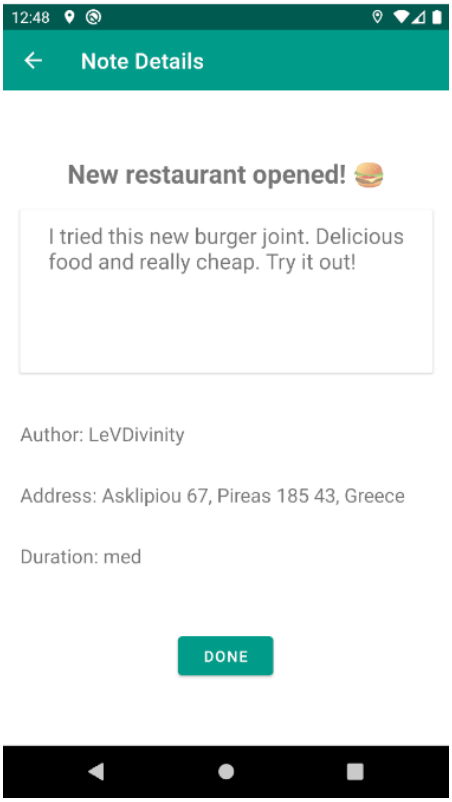


Εικόνα 5.10: Διαδρομή πλοήγησης με περπάτημα

Με την επιλογή “Show Info”, μεταφερόμαστε σε μια καινούρια οθόνη - activity, που περιέχει τα στοιχεία της συγκεκριμένης εκδήλωσης ή σημείωσης που επιλέχθηκε. Στην περίπτωση που ο χρήστης επιλέξει σημείωση, παρουσιάζεται στην οθόνη ο τίτλος της, ένα κείμενο με την περιγραφή που έχει γραφτεί, η γεωγραφική τοποθεσία της, καθώς και η διάρκεια που θα παραμείνει πάνω στον χάρτη (*Εικόνα 5.12*). Στην περίπτωση που ο χρήστης επιλέξει εκδήλωση, εμφανίζεται ο τίτλος, το κείμενο περιγραφής της εκδήλωσης, το άτομο που την έχει διοργανώσει, η ημερομηνία και η διεύθυνση που θα πραγματοποιηθεί, καθώς και οι καλεσμένοι της (*Εικόνα 5.11*). Τέλος, με την επιλογή “Cancel”, ο χρήστης εξέρχεται από το παράθυρο επιλογών και συνεχίζει στην οθόνη το



Εικόνα 5.11: Λεπτομέρειες Note

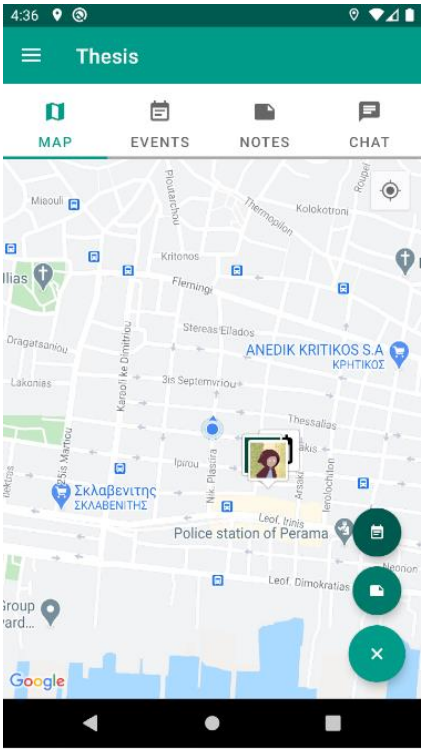


Εικόνα 5.12: Λεπτομέρειες Event

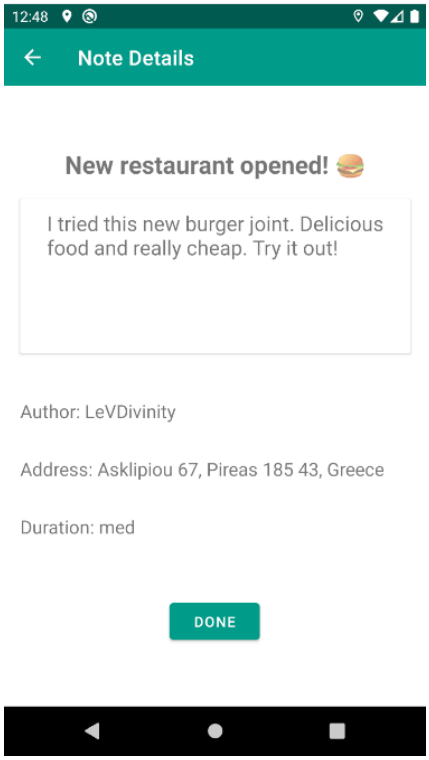
Στο κάτω δεξιά μέρος του χάρτη, υπάρχει ένα Floating Action Button (*Εικόνα 5.13*). Επιλέγοντας το, ο χρήστης μπορεί να δημιουργήσει μια καινούρια σημείωση ή εκδήλωση. Με την επιλογή δημιουργίας σημείωσης, ανοίγει μια καινούρια οθόνη. Εκεί, ζητείται από τον χρήστη να προσθέσει τις απαραίτητες πληροφορίες που χρειάζονται για την δημιουργία της σημείωσης. Στην πρώτη περιοχή κειμένου, ο χρήστης καλείται να δώσει τον τίτλο της σημείωσης. Το μέγεθος του τίτλου δεν πρέπει να ξεπερνά τους 35 χαρακτήρες, όπως αναγράφεται και στο πάνω δεξιά μέρος της περιοχής κειμένου. Επιπλέον, ο χρήστης πρέπει να προσθέσει τις επιπλέων πληροφορίες της σημείωσης, που δεν πρέπει να ξεπεράσουν τους 144 χαρακτήρες. Τέλος, ο χρήστης επιλέγει την χρονική διάρκεια που θα παραμείνει η σημείωση πάνω στον χάρτη (μισή ώρα, μία ώρα ή τρείς ώρες). Για να τελειώσει η δημιουργία της σημείωσης, πρέπει να συμπληρωθούν όλες οι πληροφορίες, αλλιώς δεν μπορεί να προχωρήσει στην περάτωση της διαδικασίας. Με το κουμπί Add Note, τελειώνει η εισαγωγή πληροφοριών και επιστρέφει στην οθόνη του χάρτη για να την προσθέσει (*Εικόνα 5.14*).

Παρόμοια διαδικασία έχουμε και στην εισαγωγή της εκδήλωσης. Αρχικά, έχουμε δύο περιοχές κειμένου όπως και στη σημείωση (*Εικόνα 5.15*). Η πρώτη είναι για να

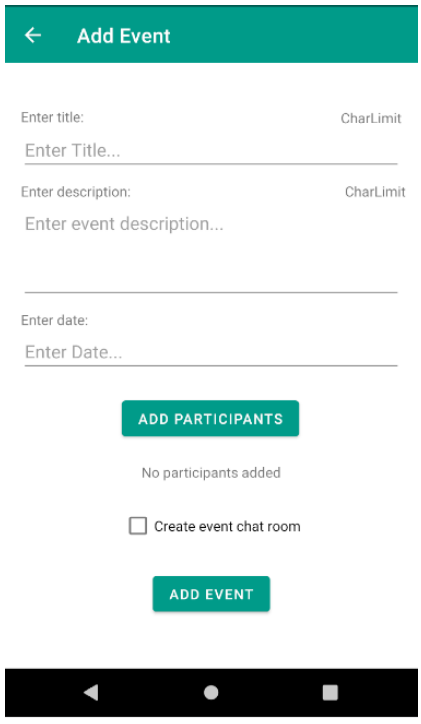
γράψουμε τον τίτλο και η δεύτερη για το κείμενο με τις λεπτομέρειες της εκδήλωσης. Υπάρχει όμως και μια επιπλέον περιοχή κειμένου, στην οποία ο χρήστης μπορεί να προσθέσει την ημερομηνία που θα πραγματοποιηθεί η εκδήλωση. Επιλέγοντας την, ανοίγει ένα παράθυρο ημερολογίου, για να επιλεχθεί η ημερομηνία και στην συνέχεια, αφού αυτή επιλεχθεί, αναγράφεται στο πλαίσιο κειμένου της (*Εικόνα 5.16*). Έπειτα, υπάρχει το κουμπί “add participants”, το οποίο παραπέμπει τον χρήστη παραπέμπει σε μια καινούρια σελίδα, για να επιλέξει ποια άτομα θα παρευρεθούν στην εκδήλωση. Εφόσον ο χρήστης κρίνει ποια άτομα θα προσθέσει στην εκδήλωση, μπορεί να πατήσει το κουμπί “add participants” και να ολοκληρώσει την διαδικασία επιλογής καλεσμένων. Το κείμενο που έγραφε “No participants added” κάτω από το κουμπί προσθήκης καλεσμένων, πλέον έχει αλλάξει και εμφανίζονται τα ονόματα των καλεσμένων που προστέθηκαν (*Εικόνα 5.17*). Τέλος, δίνεται στον χρήστη η επιλογή να δημιουργήσει συνομιλία για την εκδήλωση. Αν το επιλέξει, δημιουργείται μια νέα ομαδική συνομιλία στην καρτέλα συνομιλιών, με όνομα τον τίτλο της εκδήλωσης και μέλη τα άτομα που προστέθηκαν σαν καλεσμένοι στην εκδήλωση. Τελειώνοντας με την εισαγωγή των πληροφοριών της εκδήλωσης, ο χρήστης μπορεί να πατήσει το κουμπί Add Event και να πάει στην οθόνη του χάρτη για να προσθέσει την εκδήλωση.



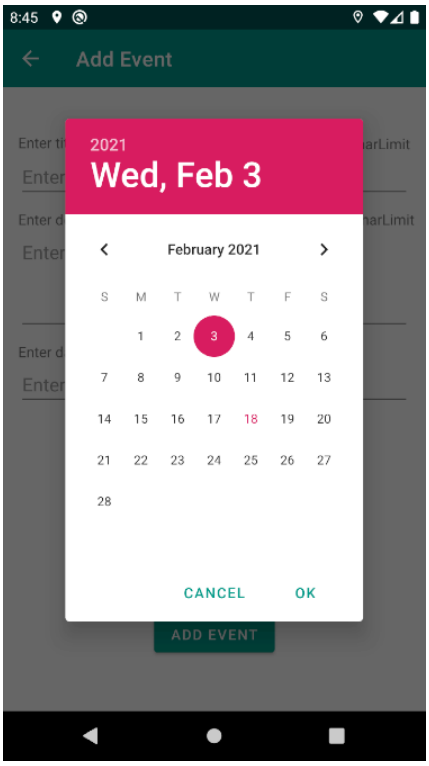
Εικόνα 5.13: Κουμπί προσθήκης Event και Note



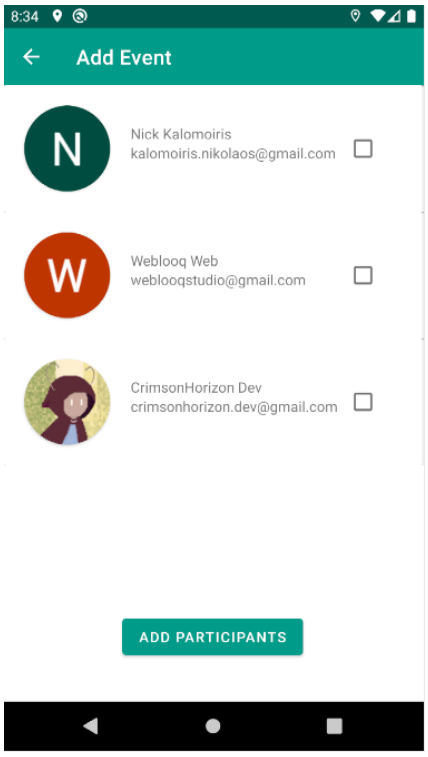
Εικόνα 5.14: Οθόνη Προσθήκης Note



Εικόνα 5.15: Οθόνη Add Event

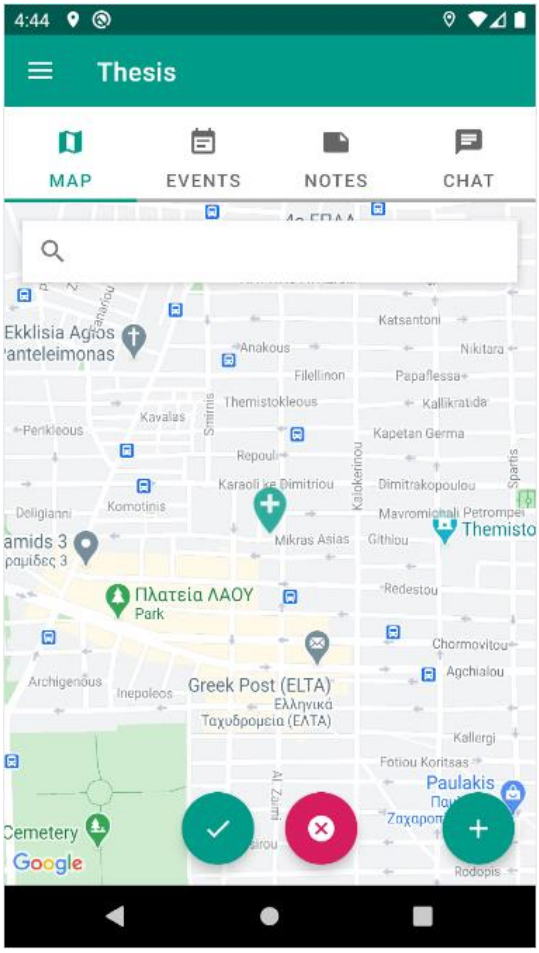


Εικόνα 5.16: Οθόνη Εισαγωγής ημερομηνίας



Εικόνα 5.17: Οθόνη επιλογής συμμετεχόντων

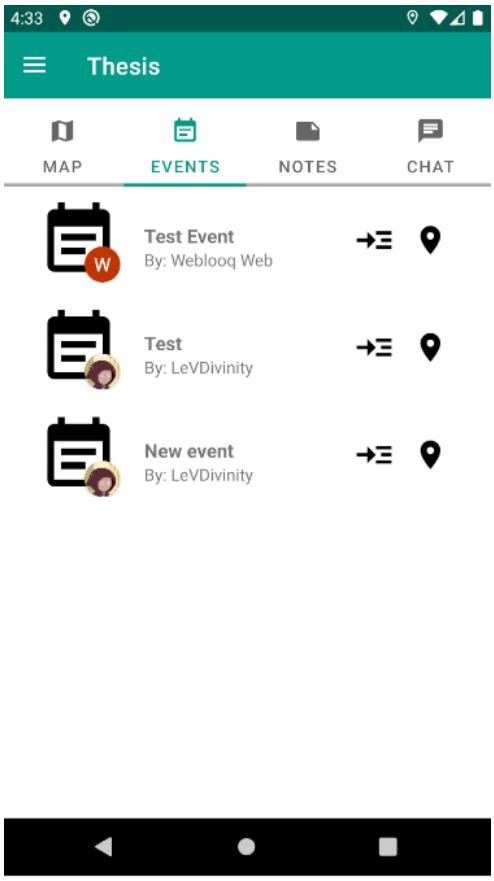
Στην συνέχεια για να είναι σε θέση να επιλέξει ο χρήστης την τοποθεσία της εκδήλωσης ή της σημείωσης πάνω στο χάρτη, με βάση την προσωπική του κρίση, δίνονται κάποιες επιπλέον καινούργιες επιλογές. Πιο συγκεκριμένα στο πάνω μέρος του χάρτη, εμφανίζεται ένα πλαίσιο αναζήτησης, όπου εκεί ο χρήστης, έχει την δυνατότητα να εισάγει την τοποθεσία που θέλει να προσθέσει τον marker και να μεταφερθεί σε αυτή. Επιπλέον, στο κέντρο της οθόνης, εμφανίζεται ένα εικονίδιο, που δείχνει που θα προστεθεί το marker αν ολοκληρωθεί η διαδικασία. Τέλος, στο κάτω μέρος του χάρτη εμφανίζονται δύο κουμπιά. Το πρώτο αναφέρεται στην αποδοχή της προσθήκης του marker στην τοποθεσία που επιλέχθηκε και το δεύτερο στην ακύρωση της διαδικασίας προσθήκης. Εφόσον ο χρήστης αποδεχτεί την τοποθεσία προσθήκης, τοποθετείται ένα marker σημείωσης ή εκδήλωσης στο σημείο όπου επέλεξε (*Εικόνα 5.18*).



Εικόνα 5.18: Προσθήκη marker στον χάρτη

5.2.4 Καρτέλα λίστας εκδηλώσεων

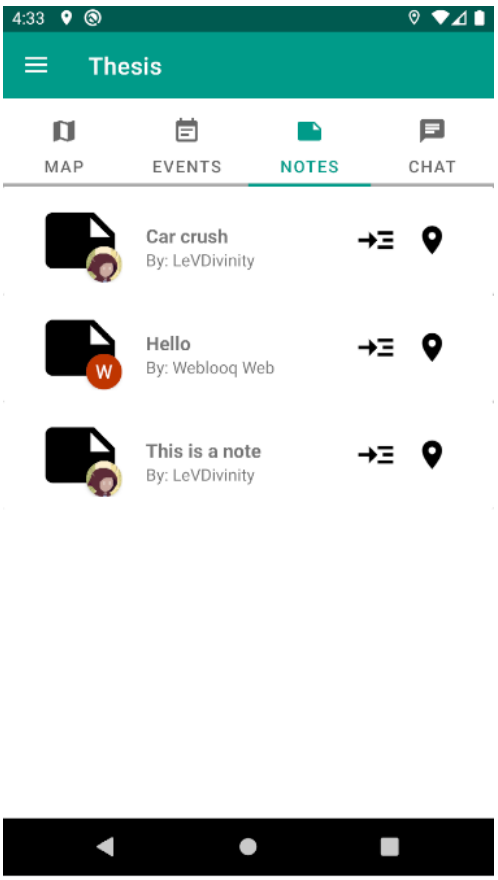
Η επόμενη στην σειρά καρτέλα, είναι η καρτέλα εκδηλώσεων. Εκεί ο χρήστης έχει την δυνατότητα να δει μία λίστα με όλες τις εκδηλώσεις στις οποίες είναι καλεσμένος ή έχει δημιουργήσει ο ίδιος (*Εικόνα 5.19*). Κάθε στοιχείο της λίστας αποτελείται από ορισμένες πληροφορίες για την εκάστοτε εκδήλωση, όπως ο τίτλος, η εικόνα, το όνομα του δημιουργού της καθώς και δύο επιλογές. Με την πρώτη επιλογή ο χρήστης κατευθύνεται στην οθόνη πληροφοριών της εκδήλωσης που έχει προαναφερθεί. Με την δεύτερη επιλογή, μεταφέρεται στην ακριβή τοποθεσία της εκάστοτε εκδήλωσης πάνω στον χάρτη. Στο σημείο αυτό αξίζει να επισημανθεί πως η λίστα αλλάζει σε πραγματικό χρόνο, κάθε φορά που προστίθεται μία νέα εκδήλωση από τον χρήστη ή από κάποιον φίλο του.



Εικόνα 5.19: Καρτέλα λίστας εκδηλώσεων

5.2.5 Καρτέλα λίστας σημειώσεων

Συνεχίζοντας η τρίτη κατά σειρά καρτέλα της αρχικής σελίδας της εφαρμογής, υπάρχει η καρτέλα της λίστας σημειώσεων (*Εικόνα 5.20*). Στο σημείο αυτό υπάρχουν όλες οι σημειώσεις που έχουν δημιουργηθεί από τον χρήστη ή από τους φίλους του πάνω στον χάρτη. Κάθε στοιχείο της λίστας περιέχει τον τίτλο της σημείωσης, το όνομα και την φωτογραφία του δημιουργού της. Όπως και στην λίστα εκδηλώσεων, υπάρχουν δύο επιλογές σε κάθε στοιχείο. Με την πρώτη επιλογή ο χρήστης ανοίγει την οθόνη πληροφοριών της εκάστοτε σημείωσης και με την δεύτερη επιλογή οδηγείται στην τοποθεσία της σημείωσης πάνω στον χάρτη. Στο σημείο αυτό να αναφέρουμε πως η λίστα μεταβάλλεται δυναμικά και σε πραγματικό χρόνο κάθε φορά που προστίθεται μία νέα σημείωση πάνω στον χάρτη, από τον χρήστη ή κάποιον φίλο του.

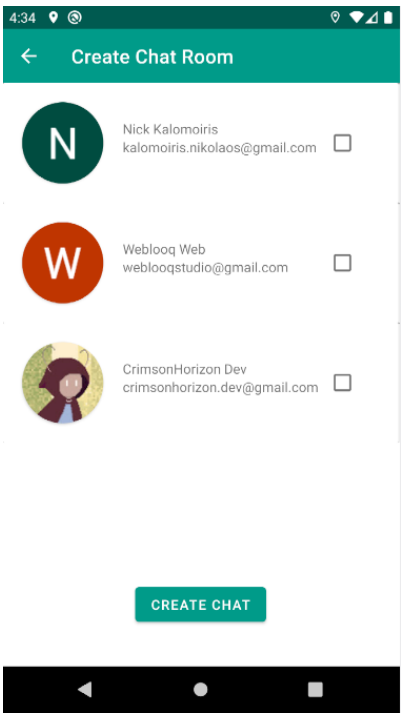


Εικόνα 5.20: Καρτέλα λίστας σημειώσεων

5.2.6 Καρτέλα συνομιλιών

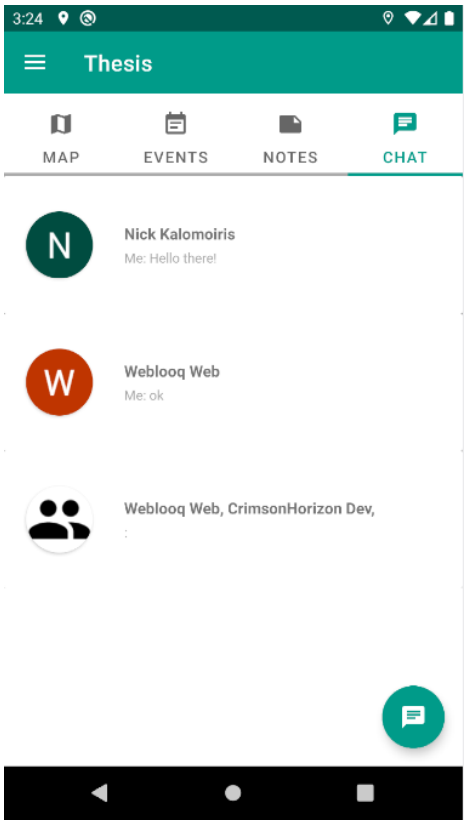
Στην τέταρτη και τελευταία καρτέλα της αρχικής οθόνης, υπάρχει η λίστα συνομιλιών του χρήστη (*Εικόνα 5.22*). Εκεί, σώζονται όλες οι ατομικές και ομαδικές συνομιλίες που έχει δημιουργήσει ο ίδιος ο χρήστης ή έχει προστεθεί σαν μέλος σε αυτές. Κάθε συνομιλία της λίστας έχει ένα εικονίδιο, τον τίτλο της συνομιλίας καθώς και το τελευταίο μήνυμα που έχει σταλεί σε αυτή. Εδώ να αναφέρουμε πως για τις συνομιλίες μεταξύ του χρήστη και ενός μόνο φίλου του, η εικόνα συνομιλίας και το όνομά της αποτελούνται από τα στοιχεία του φίλου του (εικόνα προφίλ και όνομα χρήστη). Από την άλλη στις ομαδικές συνομιλίες, μέσα στις οποίες μπορούν να υπάρχουν πάνω από τρία μέλη, το όνομα της συνομιλίας είναι τα ονόματα των μελών της χωρισμένα με κόμματα και η εικόνα της είναι ένα logo που απεικονίζει μία ομάδα ατόμων.

Στο κάτω δεξιά μέρος της οθόνης, υπάρχει το κουμπί δημιουργίας συνομιλίας. Η επιλογή του ανοίγει μία νέα οθόνη στον χρήστη της εφαρμογής, με την λίστα των φίλων του, δίνοντάς του την δυνατότητα να διαλέξει ποιους από αυτούς θα προσθέσει σαν μέλη στην συνομιλία. Αφού γίνει η επιλογή των μελών της συνομιλίας, στην συνέχεια πατάει το κουμπί “create chat” και με αυτό τον τρόπο προστίθεται η καινούργια συνομιλία στην λίστα (*Εικόνα 5.21*).

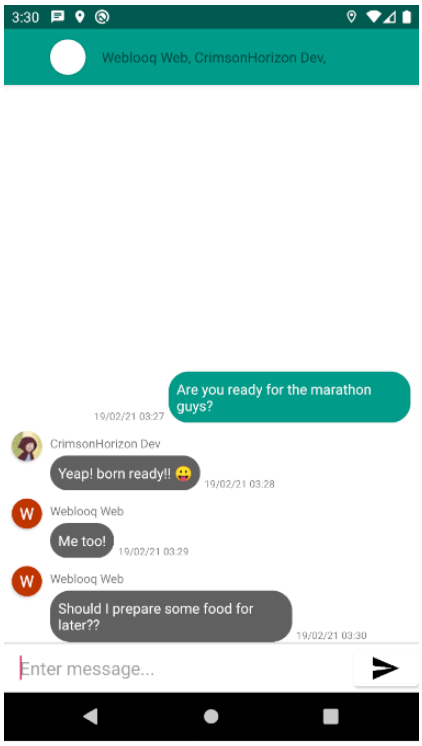


Εικόνα 5.21: Οθόνη δημιουργίας συνομιλίας

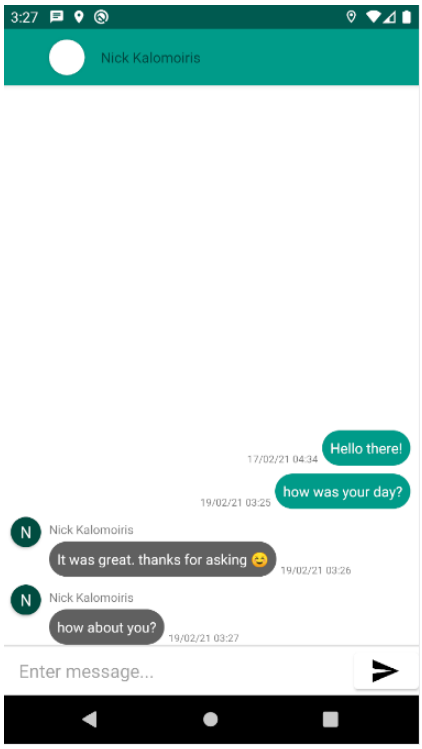
Έπειτα επιλέγοντας μία συνομιλία, ο χρήστης κατευθύνεται στην οθόνη της. Εκεί έχει την ευκαιρία να δει τα μηνύματα που έχουν σταλεί αλλά και ποιοι χρήστες συμμετείχαν ενεργά σε αυτή. Κάθε μήνυμα φέρει την ημερομηνία και ώρα αποστολής του. Επιπλέον, τα μηνύματα που στέλνονται από τον χρήστη εμφανίζονται στα δεξιά της λίστας, σε μία φυσαλίδα συνομιλίας πράσινου χρώματος. Τα μηνύματα των υπολοίπων μελών – φίλων εμφανίζονται στα αριστερά της λίστας, σε μία φυσαλίδα συνομιλίας γκρι χρώματος, μαζί με την φωτογραφία προφίλ τους. Τέλος, στο κάτω μέρος της οθόνης, υπάρχει ένα κενό πλαίσιο κειμένου, μέσα στο οποίο καθένας μπορεί να πληκτρολογήσει το μήνυμα που επιθυμεί και να το στείλει στα άλλα μέλη της ομάδας του πατώντας στην συνέχεια το κουμπί αποστολής, το οποίο βρίσκεται δίπλα από το κείμενο (*Εικόνες 5.23 – 5.24*).



Εικόνα 5.22: Καρτέλα συνομιλιών



Εικόνα 5.23: Συνομιλία με ένα φίλο



Εικόνα 5.24: Ομαδική συνομιλία

ΚΕΦΑΛΑΙΟ 6

ΥΠΗΡΕΣΙΕΣ ΕΦΑΡΜΟΓΗΣ

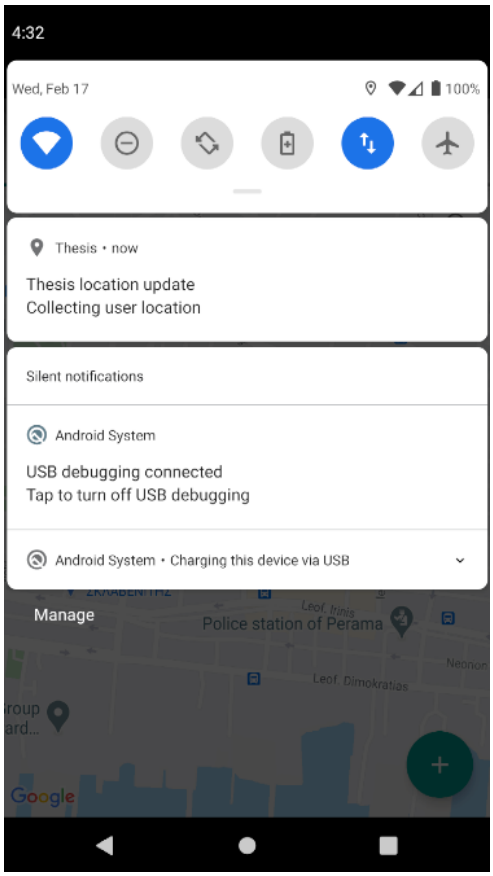
Σε αυτό το κεφάλαιο θα αναλύσουμε τις υπηρεσίες που προσφέρει η εφαρμογή στον χρήστη, καθώς και τον τρόπο που λειτουργούν κάποιες από τις υπηρεσίες που δεν είναι ορατές σε αυτόν, αλλά βοηθούν στην επεξεργασία των δεδομένων και την ορθή λειτουργιά της εφαρμογής.

6.1 Σύνδεση χρήστη

Όπως προαναφέρθηκε και σε προηγούμενα κεφάλαια, η σύνδεση του χρήστη στην εφαρμογή γίνεται με την βοήθεια του Firebase Authentication. Χρησιμοποιώντας το API που προσφέρεται, δημιουργείται ένα Sign in Intent, που περιέχει τους τρόπους εξουσιοδότησης που χρησιμοποιεί η εφαρμογή (στην περίπτωση μας, μόνο μέσω λογαριασμού Google). Εφόσον εκτελεστεί το συγκεκριμένο intent, παρουσιάζεται στον χρήστη η οθόνη σύνδεσης λογαριασμού Google. Εκεί, μπορεί να επιλέξει έναν λογαριασμό που έχει αποθηκευμένο ήδη στην συσκευή του ή να συνδεθεί με έναν διαφορετικό.

6.2 Foreground υπηρεσία ανάκτησης τοποθεσίας χρήστη

Όταν ο χρήστης εισέρχεται στην αρχική οθόνη, εκτελείται μια νέα υπηρεσία προσκηνίου, που είναι υπεύθυνη για την συλλογή της τοποθεσίας του χρήστη και την αποθήκευση της στην βάση δεδομένων. Η εκτέλεσης της είναι εμφανής στον χρήστη, καθώς εμφανίζεται μία νέα ειδοποίηση, η οποία τον ενημερώνει ότι η τοποθεσία του συλλέγεται από την εφαρμογή (*Εικόνα 6.1*). Η ειδοποίηση και η συλλογή της τοποθεσίας, παραμένει σε λειτουργία σε όλη την διάρκεια εκτέλεσης της εφαρμογής, ακόμα και αν αυτή μεταφερθεί στο παρασκήνιο και δεν χρησιμοποιείται. Η εκτέλεση της τερματίζει, όταν ο χρήστης αφαιρέσει εντελώς την εφαρμογή και από το παρασκήνιο. Η υπηρεσία συλλέγει την τοποθεσία του χρήστη ανά διαστήματα τεσσάρων δευτερολέπτων και τα μεταφέρει στην βάση δεδομένων.



Εικόνα 6.1: Ειδοποίηση συλλογής τοποθεσίας

6.3 Ανάκτηση τοποθεσίας φίλων στον χάρτη

Ο χρήστης μπορεί να δει την κίνηση των φίλων του με καθυστέρηση περίπου δύο με τεσσάρων δευτερολέπτων πάνω στον χάρτη. Στο Fragment του χάρτη, δημιουργείται ένας listener που παρατηρεί τις αλλαγές των δεδομένων στην βάση και ειδικότερα στην συλλογή δεδομένων User Location. Κάθε φορά που υπάρχει μία αλλαγή στην τοποθεσία ενός χρήστη, ελέγχεται αν παρατηρήθηκε από φίλο του χρήστη ή όχι. Αν είναι στην λίστα φίλων του χρήστη, λαμβάνει τα δεδομένα τοποθεσίας του και μεταβάλει την θέση του marker του πάνω στον χάρτη.

6.4 Ανάκτηση Events και Notes στον χάρτη

Μετά την φόρτωση του χάρτη της εφαρμογής, δημιουργούνται δύο listener στιγμιότυπου δεδομένων, με την βοήθεια του API της Realtime Database. Ο πρώτος listener λαμβάνει όλα τα δεδομένα από την συλλογή Notes της βάσης δεδομένων και ο δεύτερος όλα τα δεδομένα από την συλλογή Events. Εφόσον τα δεδομένα παραληφθούν επιτυχώς, δημιουργούνται δύο λίστες αντικειμένων Cluster Marker, στις οποίες σώζονται

τα νέα αντικείμενα Marker που περιέχουν τις πληροφορίες των δεδομένων που παραλήφθηκαν. Το αντικείμενο Cluster Marker αποτελεί επέκταση του κλασσικού δείκτη (marker) πάνω σε έναν χάρτη. Ουσιαστικά, δημιουργείται ένας marker που έχει σαν φωτογραφία, την φωτογραφία του Note ή του Event και σαν πληροφορίες τα δεδομένα του. Αφού δημιουργηθούν οι δύο αυτές λίστες, η εφαρμογή τις ανατρέχει και απεικονίζει το κάθε event και note πάνω στον χάρτη, ανάλογα την γεωγραφική τους τοποθεσία.

6.5 Σχεδίαση διαδρομής στον χάρτη – πλοήγηση Google Maps

Για την σχεδίαση και εύρεση των γρηγορότερων διαδρομών μεταξύ των marker του χάρτη, χρησιμοποιείται το Google Maps API και το Directions API. Όταν ο χρήστης διαλέγει την επιλογή εύρεσης διαδρομής για μία εκδήλωση ή σημείωση πάνω στον χάρτη, γίνεται μία κλήση στο Directions API, με δεδομένα την θέση εκκίνησης της διαδρομής και την θέση τερματισμού της διαδρομής. Σαν αποτέλεσμα παίρνουμε κάποια δεδομένα σε μορφή JSON που αφορούν την συγκεκριμένη διαδρομή. Από αυτά τα δεδομένα, εμείς επιλέγουμε τις τρείς κοντινότερες διαδρομές (Αν αυτές υπάρχουν) και τις σχεδιάζουμε στον χάρτη με Polylines. Τέλος, ο χρήστης έχει την επιλογή να ανοίξει την τοποθεσία που διάλεξε με την χρήση της πλοήγησης στην εφαρμογή Google Maps.

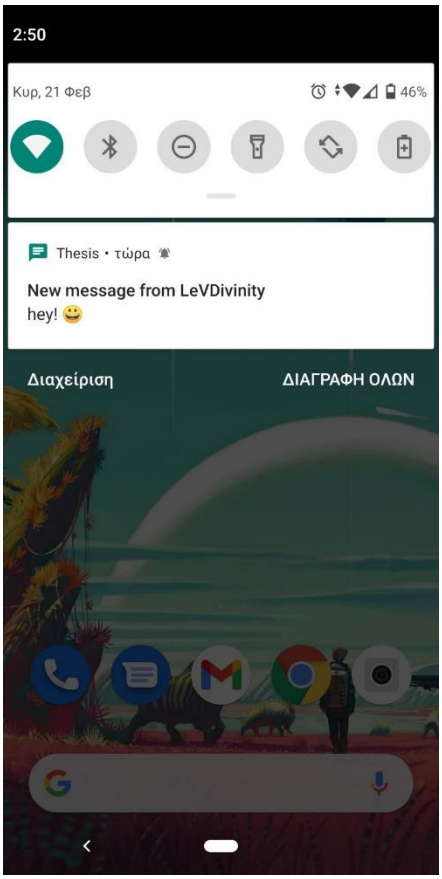
6.6 Λειτουργία συνομιλιών

Κάθε χρήστης έχει την δυνατότητα να δημιουργήσει συνομιλίες για να επικοινωνήσει με τους φίλους του. Κάθε φορά που ένας χρήστης στέλνει μήνυμα στην συνομιλία, αποθηκεύεται στην λίστα μηνυμάτων της εκάστοτε συνομιλίας, στην βάση δεδομένων της εφαρμογής. Όταν ο χρήστης ανοίγει μία συνομιλία από την λίστα συνομιλιών μέσα στην εφαρμογή, δημιουργείται ένας listener που παρατηρεί τις αλλαγές στην λίστα μηνυμάτων της εκάστοτε συνομιλίας στην βάση. Επιπλέον, το σύστημα λαμβάνει τα τελευταία είκοσι μηνύματα της συνομιλίας και τα εμφανίζει στην διεπαφή χρήστη. Αν ο χρήστης αρχίσει να ψάχνει παλαιότερα μηνύματα της συνομιλίας, τότε το σύστημα τα φορτώνει ανά είκοσι, για να υπάρχει γρηγορότερη απόκριση και να αποφευχθεί η άσκοπη παραλαβή μηνυμάτων που δεν θα διαβαστούν από τον χρήστη. Τέλος, ο χρόνος αποστολής του μηνύματος υπολογίζεται κατά την αποστολή του και μετατρέπεται σε

timestamp, μία μορφή που είναι εύκολο να διαβαστεί και να μετατραπεί στην ημερομηνία και ώρα διαφορετικών χρονικών ζωνών.

6.7 Background υπηρεσία push notifications για νέα μηνύματα

Η εφαρμογή παρέχει την δυνατότητα στον χρήστη να λαμβάνει ειδοποιήσεις για κάθε νέο μήνυμα που του στέλνεται σε μία συνομιλία (*Εικόνα 6.2*). Αυτό επιτυγχάνεται με την δημιουργία μιας υπηρεσίας παρασκηνίου στην εφαρμογή, που λειτουργεί ακόμα και αν ο χρήστης έχει κλείσει την εφαρμογή του. Η υπηρεσία αυτή υλοποιεί το API του Firebase Cloud messaging, για να μπορεί να επικοινωνεί με τον διακομιστή και να λαμβάνει μηνύματα δεδομένων από αυτόν. Στον διακομιστή, με την χρήση του Firebase Cloud Functions, έχει δημιουργηθεί μία συνάρτηση που ελέγχει την ροή των μηνυμάτων στην κάθε συνομιλία που βρίσκεται στην συλλογή Chat Rooms της βάσης δεδομένων. Κάθε φορά που εισάγεται ένα νέο μήνυμα σε μία συνομιλία, η συνάρτηση το ελέγχει και λαμβάνει τα δεδομένα του, όπως το ποιος είναι ο αποστολέας και το περιεχόμενο του μηνύματος. Στην συνέχεια, βρίσκει ποια είναι τα υπόλοιπα μέλη της εκάστοτε συνομιλίας και λαμβάνει τα token των συσκευών τους από την συλλογή Users στην βάση. Με αυτό το token, καθορίζεται σε ποιες συσκευές θα αποσταλούν τα δεδομένα για την δημιουργία της ειδοποίησης στον χρήστη. Εφόσον βρεθούν όλα τα token υπολοίπων μελών της συνομιλίας, δημιουργείται ένα αντικείμενο JSON με τα δεδομένα του μηνύματος και στέλνεται στις συσκευές τους. Τέλος, τα δεδομένα αυτά λαμβάνονται από την υπηρεσία παρασκηνίου της εφαρμογής, επεξεργάζονται και δημιουργείται η ειδοποίηση που εμφανίζεται στον χρήστη.



Εικόνα 6.2: Ειδοποίηση νέου μηνύματος

6.8 Αιτήματα φιλίας

Ο χρήστης έχει την δυνατότητα να δεχτεί και να στείλει αιτήματα φιλίας από και προς άλλους χρήστες της εφαρμογής. Όταν ο χρήστης βρίσκεται στην οθόνη προσθήκης φίλων, εισάγει το email του χρήστη που επιθυμεί για να τον προσθέσει. Κατά την διαδικασία εισαγωγής, το email ελέγχεται σε δύο στάδια. Στο πρώτο στάδιο, γίνεται έλεγχος για το αν το αλφαριθμητικό που εισήγαγε ο χρήστης, τηρεί της προδιαγραφές και την μορφή ενός email. Αν δεν τις τηρεί, τότε απορρίπτεται από το σύστημα. Εφόσον περάσει το πρώτο στάδιο, το σύστημα ελέγχει αν υπάρχει κάποιος χρήστης στην βάση δεδομένων με το email που δόθηκε. Αν δεν υπάρχει, εμφανίζεται το μήνυμα “User not found” και η διαδικασία τερματίζει. Αν υπάρχει, τότε στέλνεται το αίτημα φιλίας στον χρήστη με το συγκεκριμένο email.

ΚΕΦΑΛΑΙΟ 7

ΕΠΙΛΟΓΟΣ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

7.1 Πηγαίος κώδικας

Ο πηγαίος κώδικας της εφαρμογής μπορεί να βρεθεί στο Github, στην εξής ηλεκτρονική διεύθυνση: https://github.com/nikos-kalomoiris/geolocation-social-network-thesis.

Πληροφορίες για την εγκατάσταση και την εκτέλεση της εφαρμογής υπάρχουν στο αρχείο Readme στο Github. Για την εκτέλεση θα χρειαστεί και η δημιουργία κλειδιών API του Google maps και του Directions API, καθώς και έναν λογαριασμό στο Firebase.

7.2 Επίλογος

Κατά την υλοποίηση της συγκεκριμένης διπλωματικής αναλύθηκαν και δοκιμάστηκαν πολλά στοιχεία της κοινωνικής δικτύωσης, καθώς και τα αποτελέσματα που προκύπτουν αν την συνδυάσουμε με την πληροφορία της γεωγραφικής τοποθεσίας των χρηστών της. Παρουσιάστηκε ένας εναλλακτικός τρόπος εμφάνισης της πληροφορίας ενός κλασσικού κοινωνικού δικτύου, πάνω σε στοιχεία του χάρτη, καθώς και πως υπάρχει μία πιο διαδραστική μορφή κοινωνικής δικτύωσης, ειδικά σε ότι αφορά την οργάνωση εκδηλώσεων ή την παρακολούθηση συμβάντων για τα οποία υπάρχει η πληροφορία γεωτοποθεσίας τους.

7.3 Συμπεράσματα και επεκτάσεις

Η εφαρμογή θα μπορούσε να επεκτείνει την λειτουργικότητα της στις εξής κατευθύνσεις:

1. Μπορούν να προστεθούν περισσότεροι τρόποι εγγραφής χρηστών στην εφαρμογή, όπως σύνδεση μέσω Facebook, Twitter, Github και πολλών άλλων τρόπων.
2. Η δημιουργία ενός συστήματος πρότασης νέων φίλων, με βάση τις προτιμήσεις του.

3. Η αυτόματη εισαγωγή από την εφαρμογή πολιτιστικών ή άλλων σημείων ενδιαφέροντος (π.χ. καταστήματα εστίασης) . Κοντά στα σημεία αυτά, ο χρήστης θα μπορούσε να διοργανώνει εκδηλώσεις ή να έχει κάποια προνόμια, όπως εκπτώσεις και προσφορές.
4. Μία προέκταση που θα μπορούσε να γίνει για να βελτιωθεί η λειτουργία της εφαρμογής, θα ήταν η προσθήκη επιπλέων ειδοποιήσεων όταν για παράδειγμα δημιουργείται ένα καινούριο event, όταν στέλνεται ένα αίτημα φιλίας στον χρήστης, καθώς και όταν ένας φίλος πλησιάζει σε κοντινή απόσταση από την τοποθεσία του χρήστης. Αυτές οι αλλαγές θα βελτίωναν την εμπειρία χρήσης της εφαρμογής. Επιπλέον αλλαγές θα μπορούσαν να γίνουν και στην λήψη των δεδομένων από την βάση. Θα μπορούσε να χρησιμοποιηθεί η δυνατότητα λήψης πληροφοριών ακόμα και εκτός σύνδεσης καθώς και να γίνεται καλύτερο caching για την ελαχιστοποίηση των κλήσεων του API του διακομιστή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] M. Frehat and E. Abu-Shanab, “The Role of Social Networking in the Social Reform on Young Society,” *Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems - MEDES '14*, 2014.

[2] D. Quercia, N. Lathia, F. Calabrese, G. Di Lorenzo, and J. Crowcroft, “Recommending Social Events from Mobile Phone Location Data,” *2010 IEEE International Conference on Data Mining*, 2010.

[3] K. Harvey, “Foursquare,” *Encyclopedia of Social Media and Politics*, 2014.

[4] *Yelp*. [Online]. Available: https://www.yelp.ie/dublin. [Accessed: 01-Feb-2021].

[5] *Android*. [Online]. Available: https://www.android.com/. [Accessed: 02-Feb-2021].

[6] C. L. Sabharwal, “Java, Java, Java,” *IEEE Potentials*, vol. 17, no. 3, pp. 33–37, 1998.

[7] V. Oliveira, L. Teixeira, and F. Ebert, “On the Adoption of Kotlin on Android Development: A Triangulation Study,” *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020.

[8] *Platform Architecture* [Online]. Available: https://developer.android.com/guide/platform. [Accessed: 03-Feb-2021].

[9] *Google Play*. [Online]. Available: https://play.google.com/store. [Accessed: 04-Feb-2021].

[10] *Android Studio*. [Online]. Available: https://developer.android.com/studio. [Accessed: 05-Feb-2021].

[11] “Android SDK,” *Wearable Android™*, pp. 87–109, 2015.

[12] “Run apps on the Android Emulator,” *Android Developers*. [Online]. Available: https://developer.android.com/studio/run/emulator. [Accessed: 6-Feb-2021].

[13] *Google*. [Online]. Available: https://firebase.google.com/. [Accessed: 6-Feb-2021].

[14] "Firebase Realtime Database," *Google*. [Online]. Available: https://firebase.google.com/docs/database. [Accessed: 7-Feb-2021].

[15] "Firebase Authentication," *Google*. [Online]. Available: https://firebase.google.com/docs/auth. [Accessed: 7-Feb-2021].

[16] "Firebase Cloud Messaging," *Google*. [Online]. Available: https://firebase.google.com/docs/cloud-messaging. [Accessed: 7-Feb-2021].

[17] "Cloud Functions for Firebase," *Google*. [Online]. Available: https://firebase.google.com/docs/functions. [Accessed: 8-Feb-2021].

[18] *Git*. [Online]. Available: https://git-scm.com/. [Accessed: 8-Feb-2021].

[19] "Where the world builds software," *GitHub*. [Online]. Available: https://github.com/. [Accessed: 08-Feb-2021].

[20] J. L. Harrington, "XML," *Relational Database Design*, pp. 377–385, 2009.

[21] *What is emulation?* [Online]. Available: https://www.kb.nl/en/organisation/research-expertise/research-on-digitisation-and-digital-preservation/emulation/what-is-emulation. [Accessed: 16-Feb-2021].

[22] M. Meng, S. Steinhardt, and A. Schubert, "Application Programming Interface Documentation: What Do Software Developers Want?," *Journal of Technical Writing and Communication*, vol. 48, no. 3, pp. 295–330, 2017.

[23] D. Saha, A. Mandal, "User Interface Design Issues for Easy and Efficient Human Computer Interaction: An Explanatory Approach", *International Journal of Computer Sciences and Engineering,* vol. 3, no. 1, p. 127-135, 2015.

[24] [Online]. Available: https://www.cloudflare.com/learning/serverless/glossary/backend-as-a-service-baas/. [Accessed: 16-Feb-2021].

[25] H. Shafiei, A. Khonsari, and P. Mousavi, “Serverless Computing: A Survey of Opportunities, Challenges and Applications,” 2020.

[26] “Layouts : Android Developers,” *Android Developers*. [Online]. Available: https://developer.android.com/guide/topics/ui/declaring-layout. [Accessed: 20-Feb-2021].

[27] R. Sato, D. Chiba, and S. Goto, “Detecting Android Malware by Analyzing Manifest Files,” *Proceedings of the Asia-Pacific Advanced Network*, vol. 36, p. 23, 2013.

[28] “Activity : Android Developers,” *Android Developers*. [Online]. Available: https://developer.android.com/reference/android/app/Activity. [Accessed: 20-Feb-2021].

[29] “Fragments : Android Developers,” *Android Developers*. [Online]. Available: https://developer.android.com/guide/fragments. [Accessed: 20-Feb-2021].

[30] S. Singh, S. Singh, and B. Mishra, “Artificial User Emulator to Detect Intelligent Malware on Android,” *International Journal of Intelligent Computing Research*, vol. 6, no. 4, pp. 640–646, 2015.

[31] “FCM Architectural Overview | Firebase,” *Google*. [Online]. Available: https://firebase.google.com/docs/cloud-messaging/fcm-architecture. [Accessed: 23-Feb-2021].

ΠΑΡΑΡΤΗΜΑ

ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΛΕΙΤΟΥΡΓΙΩΝ

Α) Μέθοδος signInOptions και onActivityResult για την σύνδεση του χρήστη

```
public void signInOptions() {

   startActivityForResult(
         currInstance
               .createSignInIntentBuilder()
               .setAvailableProviders(Arrays.asList(
                     new AuthUI.IdpConfig.GoogleBuilder().build()))
               .setIsSmartLockEnabled(false)
               .build(),
         RC_SIGN_IN);

}

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   Log.d(TAG, "onActivityResult: called. LoginActivity");
   switch (requestCode) {
      case PERMISSIONS_REQUEST_ENABLE_GPS: {
         if (!locationPermissionGranted) {
            getLocationPermission();
         }
      }
   }

   if (requestCode == RC_SIGN_IN) {

      IdpResponse response = IdpResponse.fromResultIntent(data);
      if (resultCode == RESULT_OK) {
         Intent intent = new Intent(this, MainActivity.class);
         startActivity(intent);
         finish();
      } else {
         Toast.makeText(this, "" + response.getError().getMessage(),
Toast.LENGTH_SHORT).show();
         signInOptions();
      }
   }
}
```

Β) Υπηρεσία ανάκτησης τοποθεσίας χρήστη

```
public class UserLocationUpdateService extends Service {

    private static final String TAG = "LocationService";
    private DatabaseInformationController dbController;
    private FusedLocationProviderClient mFusedLocationClient;
    private final static long UPDATE_INTERVAL = 4000;
    private final static long FASTEST_INTERVAL = 2000;
    private String fcmToken;

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        dbController = new DatabaseInformationController();
        mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
        getFcmToken();
        Log.d("Debug", "FCM token from service: " + fcmToken);
        if (Build.VERSION.SDK_INT >= 26) {
            String CHANNEL_ID = "channel_1";
            NotificationChannel channel = new NotificationChannel(CHANNEL_ID,
                    "Update Location Channel",
                    NotificationManager.IMPORTANCE_HIGH);

            ((NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE)).createNotificationChannel(channel);

            Notification notification = new NotificationCompat.Builder(this, CHANNEL_ID)
                    .setContentTitle(getString(R.string.notification_title))
                    .setContentText(getString(R.string.notification_text))
                    .setSmallIcon(R.drawable.ic_location)
                    .setPriority(NotificationCompat.PRIORITY_MAX).build();

            startForeground(1, notification);
        }
    }

    private void getFcmToken() {
        FirebaseMessaging.getInstance().getToken().addOnCompleteListener(new
OnCompleteListener<String>() {
            @Override
```

```
        public void onComplete(@NonNull Task<String> task) {
            fcmToken = task.getResult();
        }
    });
  }

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    getLocation();
    return START_NOT_STICKY;
  }

  private void getLocation() {

    LocationRequest mLocationRequestHighAccuracy = new LocationRequest();

mLocationRequestHighAccuracy.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
;
    mLocationRequestHighAccuracy.setInterval(UPDATE_INTERVAL);
    mLocationRequestHighAccuracy.setFastestInterval(FASTEST_INTERVAL);


    if (ActivityCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
      Log.d(TAG, "Location Done");
      stopSelf();
      return;
    }
    Log.d(TAG, mFusedLocationClient.toString());
    mFusedLocationClient.requestLocationUpdates(mLocationRequestHighAccuracy,
new LocationCallback() {
          @Override
          public void onLocationResult(LocationResult locationResult) {

            Location location = locationResult.getLastLocation();
            Log.d(TAG, "Location Sending");
            if (location != null) {
              FirebaseUser currUser = FirebaseAuth.getInstance().getCurrentUser();
              UserLocation userLocation = null;
              if(currUser != null) {
                User user = new User(currUser.getDisplayName(), currUser.getEmail(),
currUser.getPhotoUrl().toString(), currUser.getUid(), fcmToken);
                GeoPoint geoPoint = new GeoPoint(location.getLatitude(),
location.getLongitude());
                userLocation = new UserLocation(geoPoint, null, user);
              }
```

```
                    saveUserLocation(userLocation);
                }
            }
        },
        Looper.myLooper());
    }

    private void saveUserLocation(final UserLocation userLocation){

        try{
            Log.d(TAG, userLocation.toString());
            dbController.saveUserLocation(userLocation);
        }catch (NullPointerException e){
            Log.d(TAG, "Service Stopped");
            stopSelf(); //Stop when user is logged out
        }

    }
}
```

Γ) Ανάκτηση και ανανέωση τοποθεσίας φίλων στον χάρτη

```
private void updateFriendsLocation() {
    try {
        for(int i = 0; i < clusterMarkers.size(); i++) {
            final DatabaseReference friendLocRef = FirebaseDatabase.getInstance()
                    .getReference()
                    .child(getString(R.string.users_location_collection))
                    .child(clusterMarkers.get(i).getUser().getuId());
            final int index = i;
            friendLocRef.addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                    UserLocation friendLocation = dataSnapshot.getValue(UserLocation.class);
                    Log.d("Friends", "Fetching Data: " +
friendLocation.getUser().getuDisplayName());

if(!friendLocation.getUser().getuId().equals(FirebaseAuth.getInstance().getUid())) {
                        LatLng newPosition = new
LatLng(friendLocation.getGeoPoint().getLatitude(),
                                friendLocation.getGeoPoint().getLongitude());
                        clusterMarkers.get(index).setPosition(newPosition);
                        clusterManagerRenderer.updateClusterMarker(clusterMarkers.get(index));
                    }
```

```
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
                Log.e("Error", databaseError.getMessage());
            }
        });
    }
  }
  catch (IllegalStateException ex) {
    Log.e("Error", ex.getMessage());
  }
}

private void updateFriendsLocationRunnable() {
  mHandler.postDelayed(mRunnable = new Runnable() {
    @Override
    public void run() {
      updateFriendsLocation();
      mHandler.postDelayed(mRunnable, UPDATE_INTERVAL);
    }
  }, UPDATE_INTERVAL);
}

private void setFriendListListener() {
  clusterMarkers.clear();
  friendListRef = FirebaseDatabase.getInstance().getReference()
        .child(getString(R.string.users_collection))
        .child(user.getUid())
        .child(getString(R.string.friends_list_collection));

  friendListRef.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {

        DatabaseReference singleUserLocation =
FirebaseDatabase.getInstance().getReference()
            .child(getString(R.string.users_location_collection))
            .child(dataSnapshot.getValue(String.class));

        singleUserLocation.addListenerForSingleValueEvent(new ValueEventListener() {
          @Override
          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

            try {
```

```
                UserLocation userLocation = dataSnapshot.getValue(UserLocation.class);
                Uri iconPicture = Uri.parse(userLocation.getUser().getuIconUrl());
                Log.d("friendList", "User: " + userLocation.getUser().getuDisplayName());
                LatLng ltLnPoint = new LatLng(userLocation.getGeoPoint().getLatitude(),
                        userLocation.getGeoPoint().getLongitude());

                String title = userLocation.getUser().getuDisplayName();
                String snippet = userLocation.getUser().getuEmail();

                ClusterMarker userMarker = new ClusterMarker("User", ltLnPoint, title,
snippet, iconPicture, userLocation.getUser());

                clusterMarkers.add(userMarker);
                clusterManager.addItem(userMarker);
                refreshNoteList(userLocation.getUser(), ACTION_ADD);
                clusterManager.cluster();
                friendListModel.setMarkers(clusterMarkers);

            } catch (NullPointerException ex) {
                Log.e("Error", ex.getMessage());
            }

        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            Log.e("Error", databaseError.getMessage());
        }
      });
    }

    @Override
    public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable
String s) {
        //Not used
    }

    @Override
    public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {
        Log.d("Debug", "Data Deleted");

        for(ClusterMarker userMarker: clusterMarkers) {
            if(userMarker.getUser().getuId().equals(dataSnapshot.getValue(String.class))) {
                refreshNoteList(userMarker.getUser(), ACTION_DELETE);
                clusterMarkers.remove(userMarker);
                clusterManager.removeItem(userMarker);
                break;
```

```
            }
        }
        friendListModel.setMarkers(clusterMarkers);
        clusterManager.cluster();
    }

    @Override
    public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {
        //Not used
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
        databaseError.toException().printStackTrace();
    }
  });
}
```

Δ) Μέθοδοι ανάκτησης και τοποθέτησης εκδηλώσεων και σημειώσεων στον χάρτη

```
private void setNotesListener() {
  DatabaseReference notesList = FirebaseDatabase.getInstance().getReference()
        .child(getString(R.string.notes_collection));

  notesList.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {

        Note currNote = dataSnapshot.getValue(Note.class);
        String key = dataSnapshot.getKey();

        DatabaseReference author = FirebaseDatabase.getInstance().getReference()
              .child(getString(R.string.users_collection))
              .child(user.getUid())
              .child(getString(R.string.friends_list_collection))
              .child(currNote.getAuthor().getuId());

        author.addListenerForSingleValueEvent(new ValueEventListener() {
          @Override
          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if(dataSnapshot.exists() ||
currNote.getAuthor().getuId().equals(user.getUid())) {
                Log.d("Added", "Note added: " + dataSnapshot.exists() + " " +
dataSnapshot.getValue(String.class));
                setNoteMarkOnMap(currNote, key);
```

```
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            Log.e("Error", databaseError.getMessage());
        }
    });

}

@Override
public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable
String s) {
    //Not Used
}

@Override
public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {
    String key = dataSnapshot.getKey();
    removeNoteMarkFromMap(key);
}

@Override
public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {
    //Not Used
}

@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
    databaseError.toException().printStackTrace();
}
});
}

private void setNoteMarkOnMap(Note currNote, String key) {
try {
    ClusterMarker noteMarker = new ClusterMarker("Note", currNote, key);

    noteMarkers.add(noteMarker);
    clusterManager.addItem(noteMarker);
    clusterManager.cluster();

}
catch (NullPointerException ex) {
    Log.e("Error", ex.getMessage());
```

```
    }
}

private void removeNoteMarkFromMap(String key) {
    try {

        for (ClusterMarker noteMarker: noteMarkers) {
            if(noteMarker.getTag().equals("Note")) {
                if(noteMarker.getKey().equals(key)) {
                    noteMarkers.remove(noteMarker);
                    clusterManager.removeItem(noteMarker);
                    break;
                }
            }
        }
        clusterManager.cluster();


    }
    catch (NullPointerException ex) {
        Log.e("Error", ex.getMessage());
    }
}

private void refreshNoteList(User noteUser, int action) {
    refreshNoteList = FirebaseDatabase.getInstance().getReference()
            .child(getString(R.string.notes_collection));

    refreshNoteList.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for(DataSnapshot noteSnapshot: dataSnapshot.getChildren()) {
                Note note = noteSnapshot.getValue(Note.class);
                String key = noteSnapshot.getKey();

                if(note.getAuthor().getuId().equals(noteUser.getuId())) {
                    ClusterMarker marker = new ClusterMarker("Note", note, key);

                    if(action == ACTION_ADD) {
                        removeNoteMarkFromMap(key);
                        setNoteMarkOnMap(note, key);
                    }
                    else if(action == ACTION_DELETE) {

                        for(int i = 0; i < noteMarkers.size(); i++) {

if(noteMarkers.get(i).getNote().getAuthor().getuId().equals(noteUser.getuId())) {
```

```
                    removeEventMarkOnMap(key);
                    break;
                }
            }

        }

    }
    clusterManager.cluster();
}

@Override
public void onCancelled(@NonNull DatabaseError databaseError) {
    Log.e("Error", databaseError.getMessage());
}
});
}

private void setEventsListener() {
DatabaseReference eventsList = FirebaseDatabase.getInstance().getReference()
        .child(getString(R.string.events_collection));

eventsList.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {

        Event currEvent = dataSnapshot.getValue(Event.class);
        String key = dataSnapshot.getKey();


        for(String participantId: currEvent.getParticipants()) {

            if(participantId.equals(user.getUid())) {
                setEventMarkOnMap(currEvent, key);
            }
        }

    }

    @Override
    public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable
String s) {

    }
```

```
        @Override
        public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {
            Event currEvent = dataSnapshot.getValue(Event.class);
            String key = dataSnapshot.getKey();
            removeEventMarkOnMap(key);
        }

        @Override
        public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {

        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}

private void setEventMarkOnMap(Event currEvent, String key) {
    try {
        ClusterMarker eventMarker = new ClusterMarker("Event", currEvent, key);

        eventMarkers.add(eventMarker);
        clusterManager.addItem(eventMarker);
        clusterManager.cluster();

    }
    catch (NullPointerException ex) {
        ex.printStackTrace();
    }
}

private void removeEventMarkOnMap(String key) {
    try {
        for(ClusterMarker eventMarker: eventMarkers) {
            if(eventMarker.getTag().equals("Event")) {
                if(eventMarker.getKey().equals(key)) {
                    eventMarkers.remove(eventMarker);
                    clusterManager.removeItem(eventMarker);
                    break;
                }
            }
        }
        clusterManager.cluster();
    }
```

```
    catch (NullPointerException ex) {
        Log.e("Error", ex.getMessage());
    }
}
```

Ε) Μέθοδοι για την εύρεση και σχεδίαση των μονοπατιών μεταξύ χρήστη και σημείωσης ή εκδήλωσης.

```
private void calculateDirections(ClusterMarker marker) {

    if(polylines.size() > 0) {
        for (PolylineData singlePolyline: polylines) {
            singlePolyline.getPolyline().remove();
        }
        polylines.clear();
        polylines = new ArrayList<>();
        setSelectedMarkerVisibility(selectedMarker, true);
    }

    com.google.maps.model.LatLng destination = new com.google.maps.model.LatLng(
            marker.getPosition().latitude,
            marker.getPosition().longitude
    );
    DirectionsApiRequest directions = new DirectionsApiRequest(geoApiContext);

    mFusedLocationProviderClient.getLastLocation()
            .addOnSuccessListener(new OnSuccessListener<Location>() {
                @Override
                public void onSuccess(Location location) {

                    directions.alternatives(true);
                    directions.mode(travelMode);
                    directions.origin(
                            new com.google.maps.model.LatLng(
                                    location.getLatitude(),
                                    location.getLongitude()
                            )
                    );

                    directions.destination(destination).setCallback(new
PendingResult.Callback<DirectionsResult>() {
                        @Override
                        public void onResult(DirectionsResult result) {
                            Log.d(DIRECTION_TAG, "Calculating Directions");

                            addPathPolylines(result);
```

```
                }

                @Override
                public void onFailure(Throwable e) {
                    Log.e(DIRECTION_TAG, "Unable to calculate directions. Error " +
e.getMessage() );

                }
            });
        }
    });
}

private void addPathPolylines(DirectionsResult result) {

    new Handler(Looper.getMainLooper()).post(new Runnable() {
        @Override
        public void run() {
            double duration = 999999999;     // A big number for min duration calculation
            for(DirectionsRoute route: result.routes) {
                List<com.google.maps.model.LatLng> decodedPath =
PolylineEncoding.decode(route.overviewPolyline.getEncodedPath());

                List<LatLng> newDecodedPath = new ArrayList<>();

                for(com.google.maps.model.LatLng latLng: decodedPath) {
                    newDecodedPath.add(new LatLng(latLng.lat, latLng.lng));
                }

                Polyline polyline = map.addPolyline(new
PolylineOptions().addAll(newDecodedPath));
                polyline.setColor(ContextCompat.getColor(getActivity(),
R.color.colorPolylineDefault));
                polyline.setClickable(true);
                polylines.add(new PolylineData(polyline, route.legs[0]));

                double tempDuration = route.legs[0].duration.inSeconds;
                if(tempDuration < duration) {
                    duration = tempDuration;
                    onPolylineClick(polyline);
                    setCameraToRoute(polyline.getPoints());
                }

                Log.d("Clicked", "Added polyline with id: " + polyline.getId());
            }
        }
```

```
    });
}
```

ΣΤ) Μέθοδοι ανάκτησης μηνυμάτων και συνομιλιών

```
private void createChatRoomsListener() {
    chatRoomsListener = new ChildEventListener() {
        @Override
        public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {
            Boolean userInChat = false;

            ArrayList<User> chatRoomUsers = new ArrayList<>();
            for(DataSnapshot userSnapshot: dataSnapshot.child("users").getChildren()) {
                chatRoomUsers.add(userSnapshot.getValue(User.class));

if(userSnapshot.getValue(User.class).getuId().equals(FirebaseAuth.getInstance().getUid())
) {
                    userInChat = true;
                }
            }

            if(userInChat) {

                final String chatRoomId = dataSnapshot.getKey();
                boolean chatRoomExists = false;
                String chatRoomName =
dataSnapshot.child("chatRoomName").getValue(String.class);
                String chatRoomType = dataSnapshot.child("type").getValue(String.class);

                for(ChatRoom chatRoom: chatRooms) {
                    if(chatRoomId.equals(chatRoom.getChatRoomId())) {
                        chatRoomExists = true;
                        break;
                    }
                }

                if(!chatRoomExists) {
                    if(chatRoomUsers.size() <= 2 ) {
                        for(User user: chatRoomUsers) {
                            if(!user.getuId().equals(FirebaseAuth.getInstance().getUid())) {
                                chatRoomName = user.getuDisplayName();
                            }
                        }
                    }
```

```
                ChatRoom chatRoom = new ChatRoom(chatRoomId, chatRoomName,
chatRoomUsers, "", "", "", chatRoomType);
                chatRooms.add(chatRoom);
                adapter.updateChatRoomList(chatRoom);
            }

            lastMessageRef = dataSnapshot.child(getString(R.string.messages_collection))
                    .getRef()
                    .limitToLast(1);

            lastMessageListener = new ChildEventListener() {
                @Override
                public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable
String s) {
                    Message message = dataSnapshot.getValue(Message.class);

                    String lastMessage = message.getMessage();
                    String lastMessageSenderName = message.getSender().getuDisplayName();
                    String lastMessageSenderId = message.getSender().getuId();

                    adapter.updateLastMessage(lastMessage, chatRoomId,
lastMessageSenderName, lastMessageSenderId);
                }

                @Override
                public void onChildChanged(@NonNull DataSnapshot dataSnapshot,
@Nullable String s) {
                    //Not Used
                }

                @Override
                public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {
                    //Not Used
                }

                @Override
                public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable
String s) {
                    //Not Used
                }

                @Override
                public void onCancelled(@NonNull DatabaseError databaseError) {
                    Log.e("Error", databaseError.getMessage());
                }
            };
```

```
            lastMessageRef.addChildEventListener(lastMessageListener);
        }
    }

    @Override
    public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable
String s) {
        //Not Used
    }

    @Override
    public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {
        //Not Used
    }

    @Override
    public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {
        //Not Used
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
        Log.e("Error", databaseError.getMessage());
    }
  };
}

private void setChatRoomsListener() {
  chatRoomsRef = FirebaseDatabase.getInstance().getReference()
        .child(getString(R.string.chat_rooms_collection));
  chatRoomsRef.addChildEventListener(chatRoomsListener);
  firstTime = false;
}

@Override
public void onChatRoomClick(int position) {
  Intent intent = new Intent(getContext(), SingleChatActivity.class);
  Bundle bundle = new Bundle();
  bundle.putSerializable("ChatRoom", chatRooms.get(position));
  intent.putExtras(bundle);
  startActivity(intent);
}
```

```
private void getFirstMessages() {
  fetchFirstMessages.addListenerForSingleValueEvent(new ValueEventListener() {
```

```
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        int index = 0;
        for(DataSnapshot messageSnapshot: dataSnapshot.getChildren()) {
            Message message = messageSnapshot.getValue(Message.class);
            messagesList.add(message);
            Log.d("MessageList", messagesList.get(index).getMessage());
            index++;
        }
        adapter = new ChatRecyclerViewAdapter(getApplication(), messagesList);
        recyclerView.setAdapter(adapter);
        if(messagesList.size() > 0) {
            messagesList.remove(messagesList.size() - 1);
        }

        setMessagesListener();
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
  });
}

private void setMessagesListener() {
  chatRoomSingleListener.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {
        Message message = dataSnapshot.getValue(Message.class);
        messagesList.add(message);
        adapter = new ChatRecyclerViewAdapter(getApplication(), messagesList);
        recyclerView.setAdapter(adapter);
    }

    @Override
    public void onChildChanged(@NonNull DataSnapshot dataSnapshot, @Nullable
String s) {
        //Not Used
    }

    @Override
    public void onChildRemoved(@NonNull DataSnapshot dataSnapshot) {
        //Not Used
    }
```

```
    @Override
    public void onChildMoved(@NonNull DataSnapshot dataSnapshot, @Nullable String
s) {
        //Not Used
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
        //Not Used
    }
  });
}

private void fetchMessages() {

  Query fetchMessages = dbController.getDatabase().getReference()
        .child(getString(R.string.chat_rooms_collection))
        .child(chatRoom.getChatRoomId())
        .child(getString(R.string.messages_collection))
        .orderByChild("timestamp")
        .endAt(messagesList.get(0).getTimestamp())
        .limitToLast(20);

  if(!hasEnded) {
    fetchMessages.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

            ArrayList<Message> messageToAdd = new ArrayList<>();
            for(DataSnapshot messageSnapshot: dataSnapshot.getChildren()) {
                Message message = messageSnapshot.getValue(Message.class);
                messageToAdd.add(0, message);
            }
            Collections.reverse(messageToAdd);
            messageToAdd.remove(messageToAdd.size() - 1);
            messagesList.addAll(0, messageToAdd);
            adapter.updateAdapter(messagesList);

            if(dataSnapshot.getChildrenCount() <= 19) {
                hasEnded = true;
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            Log.e("Error", databaseError.getMessage());
        }
```

```
        });
    }
}

private void sendMessage(Message message) {
    chatRoomRef.push().setValue(message);
}

private Long getTimestamp() {
    return System.currentTimeMillis()/1000;
}
```

Z) Υπηρεσία push notifications

```
public class AppFirebaseMessagingService extends FirebaseMessagingService {
    private static final String TAG = "FCMService";
    public static String charRoomId;

    @Override
    public void onMessageReceived(@NonNull RemoteMessage remoteMessage) {
        String notificationTitle = null, notificationBody = null;

        if (remoteMessage.getNotification() != null) {
            notificationTitle = remoteMessage.getNotification().getTitle();
            notificationBody = remoteMessage.getNotification().getBody();
        }
        for(String chatId: remoteMessage.getData().values()) {
            charRoomId = chatId;
        }

        sendLocalNotification(notificationTitle, notificationBody);
    }

    private void sendLocalNotification(String notificationTitle, String notificationBody) {
        Intent intent = new Intent(this, MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent,
                PendingIntent.FLAG_ONE_SHOT);

        Uri defaultSoundUri=
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);

        if (Build.VERSION.SDK_INT >= 26) {

            NotificationChannel channel = new
NotificationChannel(getString(R.string.default_notification_channel_id),
```

```
                "Channel",
                NotificationManager.IMPORTANCE_HIGH);

        NotificationCompat.Builder notificationBuilder = new
NotificationCompat.Builder(this, getString(R.string.default_notification_channel_id))
                .setAutoCancel(true)
                .setSmallIcon(R.drawable.ic_chat)
                .setContentIntent(pendingIntent)
                .setPriority(NotificationManager.IMPORTANCE_HIGH)
                .setContentTitle(notificationTitle)
                .setContentText(notificationBody)
                .setSound(defaultSoundUri);

        NotificationManager notificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.createNotificationChannel(channel);

        notificationManager.notify(1234, notificationBuilder.build());
      }

   }
}
```

Η) Μέθοδοι αποστολής αιτημάτων φιλίας

```
private boolean validateEmailAddress(String email) {
   if(!email.isEmpty() && Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
      return true;
   }
   else {
      makeToast("Invalid email address");
      return false;
   }
}

private void findFriendToAdd(String email) {
   ArrayList<String> friendsIds = new ArrayList<>();

   DatabaseReference friendList = FirebaseDatabase.getInstance().getReference()
         .child(getString(R.string.users_collection))
         .child(FirebaseAuth.getInstance().getUid())
         .child(getString(R.string.friends_list_collection));

   DatabaseReference usersList = FirebaseDatabase.getInstance().getReference()
         .child(getString(R.string.users_collection));
```

```
friendList.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dbSnap) {
        friendsIds.clear();

        for(DataSnapshot fIdSnap: dbSnap.getChildren()) {
            String fId = fIdSnap.getValue(String.class);
            friendsIds.add(fId);
        }

        for (String id: friendsIds) {
            Log.d("Debug", id);
        }

        usersList.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                boolean userFound = false;
                boolean userAlreadyExist = false;
                boolean userIsYou = false;
                for(DataSnapshot userSnapshot: dataSnapshot.getChildren()) {
                    User user = userSnapshot.getValue(User.class);
                    if(user.getuEmail().equals(email)) {
                        for(String id: friendsIds) {
                            if(user.getuId().equals(id)) {
                                userAlreadyExist = true;
                                break;
                            }
                        }
                        Log.d("Debug", "User Already Exist: " + userAlreadyExist);
                        Log.d("Debug", "User is: " + user.getuId() + " | " +
user.getuDisplayName());
                        if(!user.getuId().equals(FirebaseAuth.getInstance().getUid()) &&
!userAlreadyExist) {
                            try {
                                userFound = true;
                                usersList.child(user.getuId())
                                        .child(getString(R.string.requests_list_collection))
                                        .child(FirebaseAuth.getInstance().getUid())
                                        .setValue(FirebaseAuth.getInstance().getUid());
                                break;
                            }
                            catch (NullPointerException ex) {
                                Log.d("Error", ex.getMessage());
                            }
                        }
                        else {
```

```
                    userIsYou = true;
                }
            }
        }
        if(userFound) {
            Intent returnIntent = new Intent();
            returnIntent.putExtra("result", "Request sent successfully!");
            setResult(AddFriendActivity.RESULT_OK, returnIntent);
            finish();
        }
        else if(userAlreadyExist) {
            makeToast("User already in friend list.");
        }
        else if(userIsYou) {
            makeToast("This is your email.");
        }
        else {
            makeToast("User not found.");
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
        Log.e("Error", databaseError.getMessage());
    }
  });
 }

 @Override
 public void onCancelled(@NonNull DatabaseError databaseError) {
    Log.d("Error", databaseError.getMessage());
 }
});

}

private void makeToast(String message) {
  Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
}
```