ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Street Advisor

Στεφανία Χ. Πάτσου

Επιβλέποντες: **Αθανασία Αλωνιστιώτη,** Επίκουρος Καθηγητής
Σωκράτης Μπαρμπουνάκης, Μεταδιδακτορικός Ερευνητής
Δημήτρης Σουκαράς, Μεταπτυχιακός Ερευνητής

ΑΘΗΝΑ

Μάρτιος 2019

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Street Advisor

Στεφανία Χ. Πάτσου
Α.Μ.: 1115201400156

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αθανασία Αλωνιστιώτη,** Επίκουρος Καθηγητής
Σωκράτης Μπαρμπουνάκης, Μεταδιδακτορικός Ερευνητής
Δημήτρης Σουκαράς, Μεταπτυχιακός Ερευνητής

ΠΕΡΙΛΗΨΗ

Η συγκεκριμένη πτυχιακή εργασία έχει ως σκοπό τη δημιουργία Android εφαρμογής για την ιχνηλάτηση και την παροχή πληροφοριών πορείας για έναν κινούμενο χρήστη. Η εφαρμογή προειδοποιεί τον χρήστη για ενδεχόμενη σύγκρουση αυτοκινήτου/προσώπου στην περιοχή κίνησης. Πρόκειται για μία εφαρμογή αποφυγής ανεπιθύμητων ατυχημάτων στον δρόμο. Χρησιμοποιεί τη θέση, την ταχύτητα ή τον βηματισμό καθώς και την πληροφορία που προέρχεται από διάφορους άλλους αισθητήρες της συσκευής αλλά και την κατεύθυνση του τηλεφώνου προκειμένου να προβλέψει την επόμενη θέση του χρήστη σε διάστημα 3 δευτερολέπτων στο μέλλον.

Αυτή η εφαρμογή προορίζεται για ταξιδιώτες καθώς και για «πεζοπόρους» στους δρόμους. Ωστόσο, όλοι οι ενδιαφερόμενοι μπορούν να εγκαταστήσουν αυτήν την εφαρμογή για μελλοντική πρόληψη.

Πριν την υλοποίηση της εργασίας, έγινε εκτενής βιβλιογραφική έρευνα σε σχέση με τις υπάρχουσες αλγοριθμικές λύσεις για την πρόβλεψη πορείας χρήστη.

Η εργασία χωρίστηκε σε 3 μέρη υλοποίησης:

- Σχεδιασμός (mockups) του User Interface (UI) της εφαρμογής.
- Υλοποίηση του κώδικα και των επιμέρους λειτουργιών της εφαρμογής.
- Έλεγχος κώδικα, επίλυση σφαλμάτων και βελτίωση.

Στον σχεδιασμό του UI χρησιμοποιήθηκε το “Pencil” [23]. Η υλοποίηση του κώδικα χωρίστηκε σε 2 μέρη, την υλοποίηση της εφαρμογής χρήστη και την υλοποίηση του εξυπηρετητή, ο οποίος λαμβάνει και επεξεργάζεται τα δεδομένα του χρήστη και αποστέλλει πίσω τα σχετικά αποτελέσματα.

Τα κύρια αποτελέσματα της εφαρμογής είναι τα εξής:

- Προσδιορίζει την τοποθεσία του χρήστη.
- Ενημέρωση χρήστη όταν συμβεί κάποιο περιστατικό με την μορφή (ηχητικής) ειδοποίησης.
- Εμφάνιση στατιστικών με την μορφή γραφημάτων.
- Υπολογισμός επόμενης θέσης χρήστη.
- Εμφάνιση αρχικής και επόμενης τοποθεσίας σε χάρτη.
- Ο χρήστης μπορεί να εγγραφεί ή/και να επεξεργαστεί τα προσωπικά του στοιχεία.

Στην παρούσα εργασία παρουσιάζονται εισαγωγικές έννοιες που σχετίζονται με τον σκοπό της υλοποίησής της, αλλά και παραπλήσιες υλοποιήσεις που πρόκειται να αναπτυχθούν στο μέλλον. Στην συνέχεια, συγκρίνονται βασικές μέθοδοι, μέθοδοι οι οποίες έχουν ερευνηθεί εκτενώς από ερευνητικά κέντρα και πανεπιστήμια, που υλοποιούν το συγκεκριμένο πρόβλημα. Στην τρίτη ενότητα, περιγράφεται αναλυτικά ο προτεινόμενος τρόπος που επιλέχθηκε στα πλαίσια της πτυχιακής για την ολοκλήρωσή της. Έπειτα, αναφέρονται διάφορες περιπτώσεις πειραμάτων που αποδεικνύουν την ορθή λειτουργία της εφαρμογής. Στο τελευταίο κεφάλαιο, διατυπώνονται τα αποτελέσματα που απορρέουν από τα πειράματα αλλά και μελλοντικές βελτιώσεις που μπορούν να υλοποιηθούν.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Mobile Development & Internet of Things (IoT)
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Android, Internet of Things, τοποθεσία, online database, ειδοποίηση

ABSTRACT

This graduate project aims to track and provide tracking information for an Android user. The application “advises” the user for a possible car / person collision in the traffic area. This is an application to avoid unwanted accidents on the road. It utilizes the current location, speed or step length and other motion sensors as well as direction of the phone to predict the next user’s location within 3s in the future.

This application is intended for travelers as well as for “walkers” on the streets. However, everyone interested can install this application for future prevention.

Prior to the implementation of the project, extensive research was carried out in order to pinpoint ways with which a user’s path can be predicted.

The project was divided into 3 parts:

- Design application layout (UI mockups).
- Code and features implementation.
- Test code, bug fixes and improvement.

The program “Pencil” was used in the UI design. The code implementation was split into 2 parts, the implementation of the application/user and the Server. The Server processes the user’s data and sends the results back to the user.

The main results of the application are:

- Identifies the user’s location.
- User can sign up and/or edit his/her personal information.
- Show statistics in the form of graphs.
- Notify a user when an incident occurs with an (audible) alert/notification.
- Calculate next user location.
- Show initial and next location of user on map.

SUBJECT AREA: Mobile Development & IoT

KEYWORDS: Android, Internet of Things, location, online database, notification

Αφιερώνω την παρούσα Πτυχιακή Εργασία στους γονείς μου αλλά και φίλους μου, για την ηθική και ανιδιοτελή υποστήριξη που μου παρείχαν στην διάρκεια των σπουδών μου.

ΕΥΧΑΡΙΣΤΙΕΣ

Η εκπόνηση της συγκεκριμένης Πτυχιακής Εργασίας δε θα ήταν δυνατή χωρίς την συνεχή καθοδήγηση του Μεταδιδακτορικού Ερευνητή του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών Σωκράτη Μπαρμπουνάκη. Τον ευχαριστώ εκ βαθέως για τη συνεχή κατεύθυνση των βημάτων μου και την πολύτιμη συμβολή του στην ολοκλήρωση της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω τον Δημήτρη Σουκαρά για την παροχή βοήθειας ως προς τον τρόπο υλοποίησης συγκεκριμένου μέρους της εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα Πτυχιακή εργασία έχει τίτλο “Street Advisor” και εκπονήθηκε στα πλαίσια ολοκλήρωσης προδιαγραφών για την λήψη του πτυχίου στο τμήμα Πληροφορικής και Τηλεπικοινωνιών του ΕΚΠΑ. Η ανάθεσή της έγινε τον Απρίλιο του 2018 και η ολοκλήρωση της πραγματοποιήθηκε, εντός του χρονικού ορίου, τον Μάρτιο 2019.

Σκοπός της συγγραφής αυτής δεν είναι μόνο η διατύπωση μίας πληρέστερης ανάλυσης του θέματος αλλά και η όσο το περισσότερο ορθή και βασιζόμενη σε πειράματα εργασία. Προσπάθησα έτσι ώστε το περιεχόμενο της εργασίας να είναι όσο τον δυνατόν σαφές και κατανοήσιμο. Για τον λόγο αυτό, χρησιμοποιήθηκαν διαγράμματα, εικόνες, αρκτικόλεξα αλλά και παραδείγματα. Επιπροσθέτως, περιγράφονται και πειράματα που μέσω αυτών, θα γίνει πιο κατανοητή η υλοποίηση και το θέμα της εργασίας.

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα μεταδιδακτορικό ερευνητή του τμήματος Πληροφορικής και Τηλεπικοινωνιών του ΕΚΠΑ Σωκράτη Μπαρμπουνάκη αλλά και στον μεταπτυχιακό ερευνητή Δημήτρη Σουκαρά που συνέβαλαν στην βελτίωση της εργασίας, με συμβούλεψαν και προσφέρθηκαν να με βοηθήσουν όταν ήμουν σε αδιέξοδο. Ευχαριστώ όλους τους καθηγητές του τμήματος που συνέβαλαν στην απόκτηση απαραίτητων γνώσεων για την επιτυχή ολοκλήρωση της συγκεκριμένης εργασίας.

Περισσότερο από όλους, θέλω να ευχαριστήσω την οικογένεια και τους φίλους μου που στάθηκαν στο πλευρό μου μετά από χρόνια δυσκολίας με τον φόρτο εργασίας της σχολής. Τους ευχαριστώ που με βοήθησαν ηθικά και πνευματικά να ολοκληρώσω με όσον το δυνατό καλύτερο βαθμό τα μαθήματα της σχολής αλλά και την υπομονή που υπέδειξαν σε περιόδους δυσκολίας.

1. ΕΙΣΑΓΩΓΗ

1.1 Σκοπός

Στην σημερινή εποχή, γίνεται επιτακτική η ανάγκη για την όλο και πιο προβλέψιμη πορεία, τόσο ενός κινούμενου οχήματος, όσο και ενός ανθρώπου που απλά διασχίζει ένα δρόμο. Ο αυξανόμενος ρυθμός ζωής του πολίτη, ειδικά στις πόλεις, έχει καταστήσει αναγκαίο την παροχή μηχανισμών για την ιχνηλάτηση και την εκτίμηση του μελλοντικού σημείου κίνησης του ενδιαφερόμενου. Δεν είναι τυχαία, έτσι και αλλιώς, η άνοδος των ποσοστών ατυχημάτων λόγω βιασύνης ή απροσεξίας.

Με βάση τα στατιστικά που ανακοινώνει η αστυνομία στην ιστοσελίδα της [26], φαίνεται πως σε διάστημα 6 μηνών (α' εξάμηνο 2018), έχουν καταγραφεί περίπου 5000 ατυχήματα. Οι τραυματίες ανέρχονται περίπου τους 6.100 και διαχωρίζονται στους ελαφρά τραυματίες, σοβαρά και στους θανάτους. Αξίζει να σημειωθεί ότι, από τα θανατηφόρα ατυχήματα, οι 74 έχασαν την ζωή τους επειδή παρασύρθηκαν από κινούμενο όχημα. Επιπρόσθετα, 6% των οδηγών σκοτώθηκαν από υπερβολική ταχύτητα, ενώ το 11% από απόσπαση προσοχής.

Από τα προηγούμενα στατιστικά, κρίνεται επείγουσα η εύρεση λύσης για την μείωση των συγκεκριμένων τροχαίων ατυχημάτων. Ήδη έχουν γίνει πολυάριθμες προσπάθειες για την ελάττωσή τους. Νέες τεχνολογίες που έχουν εφαρμοστεί είναι η ESC [27], η οποία σταθεροποιεί το αυτοκίνητο όταν ο οδηγός χάσει τον έλεγχο, η AEB [28], η οποία προβλέπει την επικείμενη σύγκρουση και εφαρμόζει το μεγαλύτερο δυνατό ποσοστό δύναμης φρένων, για να μπορέσει να μειώσει την ταχύτητα του. Άλλο τεχνολογικό προϊόν αποτελεί το ISA [29], το οποίο προειδοποιεί τον οδηγό για προσπέραση του ορίου ταχύτητας σε έναν δρόμο, σημάνοντας με ήχο και εικόνα, ότι πρέπει να μειωθεί η ταχύτητα του οχήματος. Τέλος, μία τεχνολογική εξέλιξη που είναι ακόμη ύπο έρευνα στα πλαίσια του επερχόμενου συστήματος τηλεπικοινωνιών 5^{ης} γενιάς (5G), είναι τα self-driving cars [30], τα οποία προβλέπεται πως θα βοηθήσουν με την αντιμετώπιση των ατυχημάτων.

Σκοπός αυτής της πτυχιακής εργασίας είναι η παροχή λειτουργίας ορθής ιχνηλάτησης και καταγραφής πορείας χρήστη για την προειδοποίηση πιθανών ατυχημάτων στα επόμενα βήματα του χρήστη σε οδικά περιβάλλοντα. Πιο συγκεκριμένα, γίνεται αποδοτική επεξεργασία των δεδομένων του χρήστη έτσι ώστε να αποφανθεί αν στο μέλλον υπάρξει κάποιο αναπάντεχο γεγονός στην πορεία του, όπως, για παράδειγμα, μια πιθανή σύγκρουση με ένα επερχόμενο ταχέως κινούμενο όχημα. Αν και υπάρχουν εφαρμογές ιχνηλάτησης, η ανάγκη πρόβλεψης δυσάρεστων περιστάσεων δεν έχει ακόμη επιλυθεί ή διερευνηθεί, παρά μόνο σε λιγοστές περιπτώσεις, στις οποίες αποτελεί εσωτερικό μηχανισμό νεοσύστατου αυτοκινήτου. Με αυτή την εργασία, επιδιώκουμε να γενικεύσουμε την εφαρμογή αυτή σε κάθε χρήστη ο οποίος χρησιμοποιεί Android [14] λογισμικό, κάτι το οποίο θα ήταν αρκετά θετικό αφού ο αριθμός των χρηστών είναι πολύ μεγάλος (2.87 δισεκατομμύρια χρήστες). Οι χρήστες μπορούν να διαφοροποιηθούν σε τρείς κύριες κατηγορίες, τον «Μη Συνδεδεμένο Χρήστη - Γενικό», το «Άτομο» και το «Αυτοκίνητο», δηλαδή ως προς τι χρησιμοποιούν την συγκεκριμένη εφαρμογή. Επίσης, έχουν την δυνατότητα να επιλέγουν τρόπο προειδοποίησης μέσω εσωτερικών ρυθμίσεων που παρέχει η εφαρμογή Android.

1.2 Γενική Περιγραφή

Η επίλυση του συγκεκριμένου προβλήματος διερευνήθηκε και διατυπώνεται στην παρούσα πτυχιακή εργασία. Η ολοκλήρωση της λύσης του προβλήματος έγινε σε

στάδια. Αρχικά, μέσω συνεδριάσεων, αποφασίστηκε ο τρόπος επίλυσης. Στην συνέχεια, σχεδιάστηκε η εφαρμογή και υλοποιήθηκε επιτυχώς (το οποίο αποτελεί την μεριά του Client). Έπειτα, υλοποιήθηκε ο Server, δηλαδή ο εξυπηρετητής, που λαμβάνει δεδομένα, τα επεξεργάζεται και στέλνει πίσω στον Client τα υπολογισμένα αποτελέσματα. Όπως αποφάνθηκε, η εργασία αποτελείται από δύο μέρη, τον χρήστη – Client που υλοποιήθηκε με την χρήση Android Studio και είναι ο Android χρήστης, και τον εξυπηρετητή – Server, που είναι υλοποιημένος σε γλώσσα Python. Η αντιμετώπιση του συγκεκριμένου προβλήματος έγινε με βάση τον LocationService της εφαρμογής κινητού, ο οποίος μέσω Service [10], υπολογίζει κάθε δεύτερο ή σε κάθε αλλαγή κατάστασης χρήστη, την νέα τοποθεσία που βρίσκεται ήδη ο χρήστης. Στην περίπτωσή μας, ανιχνεύει την τοποθεσία του χρήστη κάθε 3 ή 6 δευτερόλεπτα, για να μην υπάρξει μεγάλη κατανάλωση πόρων. Μέσω MQTT [24], η εφαρμογή Android, στέλνει τα δεδομένα στα αρχεία Python, που αποτελούν το κύριο κομμάτι του εξυπηρετητή. Ο Server επεξεργάζεται τα δεδομένα (συντεταγμένες τοποθεσίας, ταχύτητα ή βηματισμό, επιτάχυνση ή/και κατεύθυνση χρήστη) έτσι ώστε να βρει την επόμενη προβλεφθείσα τοποθεσία του χρήστη. Έπειτα, και πάλι μέσω του MQTT, ξαναστέλνει τα νέα δεδομένα στον χρήστη. Ο χρήστης μπορεί να δει την νέα του τοποθεσία και την υπάρχουσα στον χάρτη που παρέχεται από την εφαρμογή. Επιπλέον, μπορεί να συνδεθεί στον λογαριασμό του, να επεξεργαστεί τις προσωπικές του πληροφορίες ή να δει στατιστικά που αφορούν τον τρόπο λειτουργίας της εφαρμογής (λεπτά/μέρες, ώρες/μήνες, τοποθεσίες λειτουργικότητας). Όλα τα δεδομένα του χρήστη αποθηκεύονται σε online βάση δεδομένων παρεχόμενο από την Google [13] Firebase [17] Realtime Database. Η αυθεντικοποίηση του χρήστη εκπληρώνεται κι αυτή με την σειρά της από το Google Firebase Authentication. Όσον αφορά τον εξυπηρετητή, με την εκτέλεση του κύριου προγράμματος και την συνεχή εκτέλεση του στο παρασκήνιο, μπορεί να λειτουργήσει ως κανονικός Server μέχρι ο ενδιαφερόμενος τερματίσει την λειτουργία του, οπότε και σταματάει και η κύρια λειτουργικότητα και σε όλους τους χρήστες που στέλνουν τα δεδομένα τους. Πιο λεπτομερή ανάλυση υλοποίησης θα περιγραφεί σε επόμενα κεφάλαια της παρούσας πτυχιακής εργασίας.

1.3 Επισκόπηση

Η παρούσα πτυχιακή εργασία χωρίστηκε σε κεφάλαια, τα οποία το καθένα αναλύει ένα συγκεκριμένο κομμάτι της εργασίας. Όπως ειπώθηκε προηγουμένως, το **Πρώτο Κεφάλαιο** αφορά την εισαγωγή, μία γενική περιγραφή στο σκοπό της πτυχιακής εργασίας αλλά και στον τρόπο αντιμετώπισης του ή καλύτερα επίλυσης του συγκεκριμένου προβλήματος που εξετάζεται στα όρια της εργασίας. Τα επόμενα κεφάλαια είναι τα εξής:

Το **Δεύτερο Κεφάλαιο** αναφέρει το State of the Art, δηλαδή σύγχρονες προτάσεις και προσεγγίσεις που υπάρχουν στην βιβλιογραφία για την επίλυση του συγκεκριμένου προβλήματος. Οι τρόποι αυτοί διερευνήθηκαν συλλογικά τόσο από εμένα, όσο και από τους επιβλέποντες της εργασίας. Συνοπτικά, αναφέρονται οι βασικές μέθοδοι επίλυσης του προβλήματος, η ανάλυσή τους, η σύγκρισή τους με την βοήθεια πίνακα και οι προτεινόμενες λύσεις, συμπεριλαμβανομένης και της οποίας υλοποιήθηκε η εργασία.

Το **Τρίτο Κεφάλαιο** πραγματεύεται την αναλυτική περιγραφή της υλοποίησης της πτυχιακής εργασίας. Αναφέρεται, αναλυτικά, το αλγοριθμικό κομμάτι, τα μέρη στα οποία χωρίστηκε η εργασία αλλά και τον κοινό παράγοντα υλοποίησης (MQTT implementation).

Το **Τέταρτο Κεφάλαιο** αφορά την αξιολόγηση της εργασίας με ειδικά πειράματα. Συγκεκριμένα, αναφέρεται το σενάριο πειράματος που αποδεικνύει την ορθότητα της

υλοποίησης με εξήγηση των βημάτων, του στόχου του πειράματος, παραμέτρων που μεταβάλλονται κατά την διάρκεια των πειραμάτων κ.ο.κ.

Η εργασία ολοκληρώνεται με το **Πέμπτο Κεφάλαιο** στο οποίο γίνεται σύνοψη των συμπερασμάτων, η απαρίθμηση των περιορισμών – παραδοχών που εφαρμόστηκαν στην υλοποίηση της εργασίας αλλά και των επόμενων βημάτων που μπορούν να ακολουθηθούν για την βελτίωση της παρούσας πτυχιακής εργασίας.

2. ΣΥΓΧΡΟΝΗ ΤΕΧΝΟΛΟΓΙΑ

2.1 Μέθοδοι Υλοποίησης

Όσον αφορά την βιβλιογραφία, υπάρχουν γενικά πολλές διαφορετικές μέθοδοι για τον υπολογισμό και την πρόβλεψη της επόμενης θέσης ενός αντικειμένου. Οι συγκεκριμένες μέθοδοι χρησιμοποιούν υλοποιήσεις με παραδοχές, οι οποίες αν συνδυαστούν, μπορούν να αποτελέσουν την χρυσή τομή για την λύση του προβλήματος της ακριβής πρόβλεψης.

2.1.1 Βασικές Μέθοδοι

Υπάρχουν πολλές βασικές μέθοδοι για την αντιμετώπιση του συγκεκριμένου προβλήματος. Μία από αυτές αποτελεί η υλοποίηση που εκμεταλλεύεται τα όρια του δρόμου για πιο ακριβέστερη πρόβλεψη της πορείας του κινούμενου αντικειμένου [1].

Μία άλλη υλοποίηση προβλέπει την πορεία ενός αντικειμένου (στην συγκεκριμένη περίπτωση, robot) ενώ πιο πριν προβλέπει όλες τις πιθανές πορείες που μπορεί να λάβει το αντικείμενο. Στην περίπτωση αυτή λαμβάνεται υπόψη το μέγεθος της πατούσας, το βήμα, και με βάση αυτά υπολογίζεται η πιθανότητα ορθής κατατόπισης [2].

Μία άλλη έρευνα, εκτιμά την τοποθεσία του αντικειμένου και την πορεία του με βάση πύργους δικτύου. Χρησιμοποιεί MOBILE Ad Hoc Network (MANET) και Kalman Filter για να εκτιμήσει την επόμενη τοποθεσία του. Η πορεία του αντικειμένου ιχνηλατείται με την βοήθεια σταθμών δικτύου (που είναι 3 κοντινοί σταθμοί) αλλά και του βοηθητικού αντικειμένου που είναι πιο κοντά στο επιλεγμένο που θέλουμε να βρούμε την επόμενη τοποθεσία του [3].

Η τέταρτη υλοποίηση προβλέπει την πορεία του αντικειμένου σε εσωτερικό περιβάλλον. Χρησιμοποιεί γυροσκόπιο, βηματισμό και την ταχύτητα αντικειμένου [4].

Η τελευταία υλοποίηση χρησιμοποιεί την επιτάχυνση, την ταχύτητα και την κλίση ενός κινούμενου οχήματος για την εκτίμηση της επόμενης τοποθεσίας. Επίσης, υπολογίζει το μήκος του εκάστοτε δρόμου που κινείται το όχημα. Τελικά, συνδυάζει δύο υλοποιήσεις πρόβλεψης με συγκεκριμένα βάρη, το Maneuver Recognition Model και το CYRA model Based Trajectory Prediction [42].

2.1.2 Ανάλυση Βασικών Μεθόδων

Πιο αναλυτικά, η κάθε μέθοδος χρησιμοποιεί ίδιες, παραπλήσιες ή τελείως διαφορετικές τεχνικές για τον υπολογισμό της πορείας ενός αντικειμένου. Η πρώτη μέθοδος, όπως αναφέρθηκε, χρησιμοποιεί τα νοητά/υλικά όρια της πορείας του αντικειμένου. Πιο συγκεκριμένα, κάνει χρήση matching PDR αλγόριθμο, το μέγεθος του βήματος, την κατεύθυνση, τα λάθη και τις ανακρίβειες σε συσχέτιση με τις μεγαλύτερες αποστάσεις. Με βάση αυτά, αλλά και τα όρια του χάρτη, τον αριθμό των βημάτων και την δημιουργία γραφήματος των πορειών, μπορεί να γίνει ακριβέστερη πρόβλεψη. Όπως αναφέρεται και στην βιβλιογραφία, η PDR τεχνική χρησιμοποιεί γυροσκόπιο και επιταχυντή για την εκτίμηση της τοποθεσίας του χρήστη. Όμως, στην συγκεκριμένη τεχνική, τα όρια του χάρτη (στενά δρόμων) και ο βηματισμός, βοηθούν στην ακριβέστερη πρόβλεψη της πορείας [1].

Στην δεύτερη υλοποίηση, ο βηματισμός, το Kalman Filter και η εκτίμηση σύγκρουσης είναι αυτά που βοηθούν στην εκτίμηση της πορείας του χρήστη. Αρχικά, είναι σημαντικό

να μετρηθεί το πόδι του χρήστη, η απόσταση των ποδιών μεταξύ τους, αλλά και οι πιθανές πορείες του. Στη συνέχεια, υπολογίζεται το μέσο λάθος εκτίμησης αλλά και η μέση πιθανότητα για την κατεύθυνση (δεξιά, ευθεία, αριστερά). Αυτές οι εκτιμήσεις γίνονται αποδεκτές μέχρι 3 δεύτερα, διότι έπειτα, οι τιμές τους δεν είναι βάσιμες. Υπολογίζονται τα αναφερόμενα ως Correction Step, Prediction Step τα οποία σε συνδυασμό με το Kalman Filter αλλά και την κατεύθυνση του χρήστη ανά χρονική στιγμή, συμπληρώνουν τον πίνακα τω k πρώτων προβλεπόμενων βημάτων (VAR Models) [2].

Στην τρίτη υλοποίηση, γίνεται εκτενής χρήση του Extended Kalman Filter (REKF), των Ad hoc networks, CarNet [20], πορείας και τοποθεσίας. Τα πακέτα πληροφορίας μεταφέρονται ανάμεσα από τους χρήστες στον κινητό κύριο σταθμό. Ως κινητοί σταθμοί αποτελούν τα αυτοκίνητα ή τα robot τα οποία έχουν ενσωματωμένα έναν σταθμό. Για τον υπολογισμό της απόστασης μεταξύ δύο αντικειμένων, χρησιμοποιείται η τεχνική RSSI, η οποία υπολογίζει την απόσταση ενός κινητού από τον κύριο σταθμό με βάση την δύναμη των σημάτων που μπορούν να ανταλλαχθούν. Η συγκεκριμένη μετριέται σε decibels. Τελικά, χρησιμοποιεί έως και 3 βασικούς κινητούς σταθμούς επικοινωνίας και έναν κοντινό χρήστη για την ακριβή πρόβλεψη της πορείας ενός δεύτερου χρήστη [3].

Στην τέταρτη υλοποίηση, χρησιμοποιείται ο βηματισμός, ο επιταχυντής και το γυροσκόπιο για την πρόβλεψη της πορείας. Η τροχιά κίνησης των πεζών μπορεί να χωριστεί στα βήματα με προσανατολισμό (αζιμούθιο). Αρχικά, εφαρμόζεται ο αλγόριθμος ανίχνευσης βημάτων στην κατεύθυνση άξονα z. Μετά, υπολογίζεται το αζιμούθιο για κάθε βήμα, με την βοήθεια της γωνιακής ταχύτητας (κατεύθυνσης). Με αυτό, υπολογίζεται και η κλίση, η ροπή αλλά και οι γωνίες Euler. Το μήκος του βηματισμού έχει σημαντικό ρόλο στην εκτίμηση της πορείας του χρήστη (Step counter, PDR, Step detection) [4].

Στην τελευταία υλοποίηση, χρησιμοποιείται το μήκος του δρόμου, τα σημεία κάθε μέρους του δρόμου αλλά και το γυροσκόπιο και ο επιταχυντής του οχήματος για να υπολογιστεί η επόμενη τοποθεσία του. Ο τρόπος εκτίμησης συνδυάζει δύο υλοποιήσεις, την Maneuver Recognition Model, η οποία τοποθετεί το είδος οδήγησης σε ένα από τα γνωστά προφίλ, ανάλογα με τον τρόπο που μετακινείται το όχημα. Η άλλη υλοποίηση είναι η CYRA model based Trajectory Prediction, η οποία υπολογίζει συνεχώς, με την βοήθεια της γωνιακής κλίσης και ταχύτητας, αλλά και της επιτάχυνσης, την προβλεφθείσα τοποθεσία. Οι δύο παραπάνω υλοποιήσεις, συνδυάζονται και με κατάλληλα βάρη, φέρουν ένα συνολικό αποτέλεσμα, συνεπώς ένα πιο ακριβές και ορθό από την υλοποίηση ενός και μόνο τρόπου. Σημαντικά δεδομένα που πρέπει να έχει ο χρήστης για να υπολογίσει την επόμενη τοποθεσία, με βάση την τωρινή ιχνηλάτηση του οχήματος, είναι, οι συντεταγμένες, η ταχύτητα, η επιτάχυνση, η γωνιακή κλίση και η γωνιακή ταχύτητα αλλά και δεδομένα του δρόμου, όπως το μήκος του, η ανίχνευση εισόδου σε άλλο κομμάτι δρόμου. Ειδικά, τα δεδομένα που σχετίζονται με τον δρόμο, θα έπρεπε να λαμβάνονται από βάσεις δεδομένων του διαδικτύου. Ωστόσο, δεν αναφέρεται κάτι ανάλογο, οπότε και δεν έχουμε επίγνωση για το αν η συγκεκριμένη υλοποίηση συνδέεται με την λήψη δεδομένων μέσω διαδικτύου [42].

2.1.3 Σύγκριση Μεθόδων

Πίνακας 1 : Σύγκριση Μεθόδων Υλοποίησης

	1η Υλοποίηση	2η Υλοποίηση	3η Υλοποίηση	4η Υλοποίηση	5η Υλοποίηση
Προαπαιτούμενα	Σχεδιάγραμμα χάρτη, μέτρηση βήματος χρήστη, γυροσκόπιο, επιταχυντής	Μέτρηση μεγέθους ποδιού χρήστη, Kalman Filter	Extended Kalman Filter, GPS, επιταχυντής	Βηματισμός, γυροσκόπιο, επιταχυντής	Μήκος δρόμου, γυροσκόπιο, επιταχυντής, ανίχνευση δρόμου και αυτοκινήτου
Τρόπος Εκτίμησης Απόστασης/Πορείας	PDR (Pedestrian-Space Constraints), map matching	VAR Models, Correction Step, Prediction Step, 3 δεύτερα ως όριο	RSSI, Cellular base stations	Αλγόριθμος ανίχνευσης βημάτων στο z άξονα, PDR, step counter, step detection, αζιμούθιο	Maneuver Recognition Model, CYRA model based Trajectory Prediction
Συνδεσιμότητα	Μέσω Internet	Χωρίς	AdHoc Networks, CarNet	Μέσω Internet	Δεν αναφέρεται
Δεδομένα	Step Count, Direction, Step Length	Πίνακας προβλεπόμενων τοποθεσιών	Τοποθεσίες από 3 κινητούς σταθμούς και το πλησιέστερο αντικείμενο	Μήκος βηματισμού, γωνιακή ταχύτητα, κλίση, ροπή.	Τοποθεσία οχήματος ως σημεία στο καρτεσιανό επίπεδο, γωνία κλίσης, γωνιακή ταχύτητα, ταχύτητα, επιτάχυνση

2.2 Προτεινόμενες Λύσεις

Όπως παρουσιάστηκε και παραπάνω, υπάρχουν αρκετές διαφορετικές προσεγγίσεις και λύσεις, ωστόσο στα πλαίσια αυτής της πτυχιακής εργασίας, αποφασίστηκε μία από αυτές. Ο χρήστης θα λαμβάνει τα δεδομένα από την συσκευή του, με βάση το γυροσκόπιο, τον επιταχυντή, την ανίχνευση βημάτων και την τοποθεσία του, θα στέλνει τα δεδομένα με τη χρήση του πρωτοκόλλου MQTT σε εξωτερικό περιβάλλον για επεξεργασία και θα λαμβάνει τις υπολογισμένες τιμές. Αρχιτεκτονικά, θα μπορούσαμε να ακολουθήσουμε την επιλογή της επεξεργασίας να πραγματοποιείται στην μεριά της

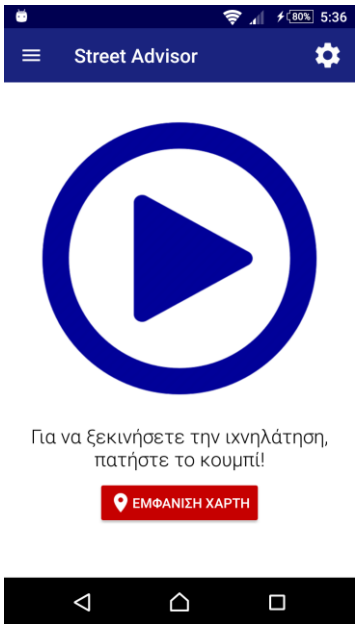
συσκευής, ωστόσο, μια τέτοια υλοποίηση θα κατανάλωνε πολύτιμους ενεργειακούς πόρους, μιας και οι συγκεκριμένοι πόροι είναι αρκετά περιορισμένοι στις κινητές τηλεφωνικές συσκευές. Άλλη εναλλακτική θα ήταν ο υπολογισμός να λαμβάνει χώρα σε ένα Server-υπολογιστή, ρυθμισμένος μόνο για κάτι ανάλογο. Με βάση τα παραπάνω, επιλέχτηκε να υλοποιηθεί ένα είδος αρχιτεκτονικής MVC [25], στο οποίο η λειτουργικότητα και οι υπολογισμοί τίθενται σε ένα άλλο σύστημα (στις περισσότερες περιπτώσεις αυτό είναι ένας εξωτερικός υπολογιστής ή δίκτυο υπολογιστών) ενώ η απεικόνιση των δεδομένων, στην συσκευή του χρήστη. Αυτό επιτυγχάνεται με το IoT, το οποίο ανταλλάσσει αυτές τις πληροφορίες με ευκολία και ασφάλεια, μέσω μηνυμάτων ανάμεσα στις δύο πλευρές (σύστημα - συσκευή), μέσω internet. Επιπρόσθετα, για κάθε κατηγορία, ο Εξυπηρετητής, θα υλοποιεί μέρος των υλοποιήσεων που προαναφέρθηκαν, επειδή η συσκευή δεν υποστηρίζει την λήψη όλων των απαραίτητων δεδομένων προς υπολογισμό.

3. ΥΛΟΠΟΙΗΣΗ

3.1 Αλγοριθμική Περιγραφή

Ως προς το αλγοριθμικό κομμάτι, η εργασία χωρίστηκε σε δύο μέρη, το μέρος του Server και το μέρος του Client. Οι δύο πλευρές επικοινωνούν μεταξύ τους μέσω ειδικών μηνυμάτων που θα περιγραφούν σε επόμενο σημείο.

Ο Client είναι ο χρήστης ο οποίος εκκινεί την εφαρμογή ονόματι «Street Advisor», έτσι ώστε να αρχίσει η ιχνηλάτηση. Η εφαρμογή είναι υλοποιημένη για Android λειτουργικό σύστημα. Κατά το πάτημα του «Play Button», το «Service» «MainService» [2] αρχίζει να λαμβάνει δεδομένα τοποθεσίας, υλοποιώντας το FusedLocationProviderClient και με την βοήθειά του «LocationCallback», δημιουργείται αντικείμενο «Location» το οποίο περιέχει τα δεδομένα που είναι απαραίτητα για την υλοποίηση. Τα δεδομένα που επεξεργαζόμαστε περαιτέρω είναι το γεωγραφικό μήκος και πλάτος, η ταχύτητα και η κατεύθυνση του χρήστη την συγκεκριμένη χρονική στιγμή. Επίσης, η εφαρμογή λαμβάνει πληροφορίες από τους ανιχνευτές της συσκευής, αν τους διαθέτει, όπως του επιταχυντή (TYPE_ACCELEROMETER), του γυροσκοπίου (TYPE_GYROSCOPE) και του ανιχνευτή βημάτων (TYPE_STEP_DETECTOR). Οι συγκεκριμένες πληροφορίες μεταφέρονται μεταξύ ενός πρωτοκόλλου HTTP ονόματι «MQTT». Οι δύο πλευρές είναι συνδεδεμένες με αυτό το πρωτόκολλο μεταφοράς μηνυμάτων ή αλλιώς «payloads» συγκεκριμένου μεγέθους byte. Το MQTT αποτελεί πρωτόκολλο IoT με το οποίο μεταφέρονται δεδομένα μεταξύ διάφορων συσκευών, αν αυτά είναι εγγεγραμμένα σε κάποιο θέμα (Topic). Η μορφή των μηνυμάτων είναι προτίμηση του κάθε προγραμματιστή. Στην συγκεκριμένη εργασία επέλεξα να στέλνω τα απαραίτητα δεδομένα στην μορφή JSON [1]. Ο Client υλοποιήθηκε σε γλώσσα JAVA, οπότε η δημιουργία String JSON μορφής ήταν εύκολη.



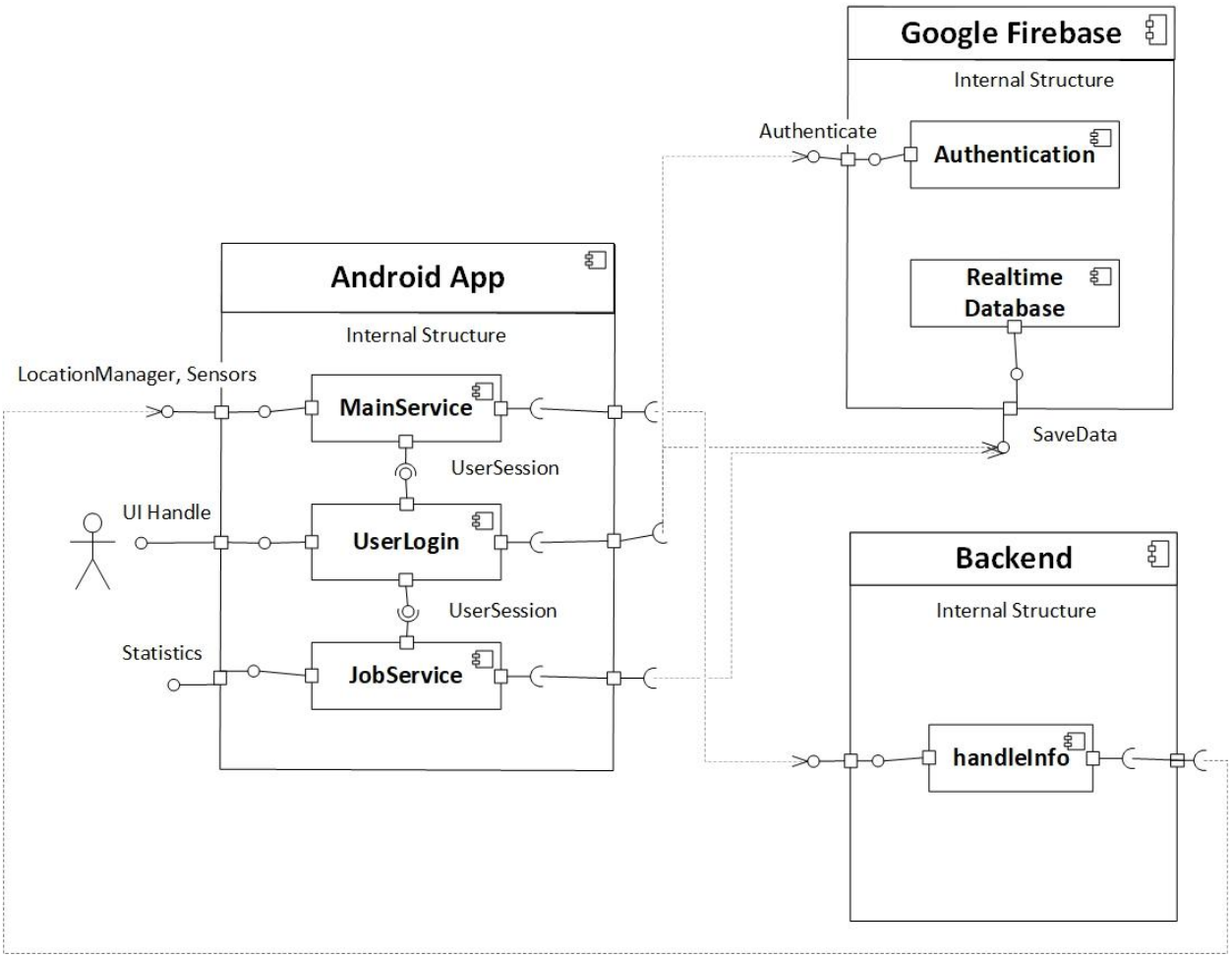
Εικόνα 1 : Αρχική Σελίδα

Ο Server λαμβάνει την συμβολοσειρά και την μετατρέπει σε μορφή λεξικού (dictionary). Ο Server αποτελείται από 3 κύρια modules, ένα για την διαχείριση των πληροφοριών (λήψη - αποστολή), το δεύτερο για τον υπολογισμό της επόμενης προβλεφθείσας θέσης του χρήστη και το τελευταίο για τον χειρισμό των χρηστών στην τοπική βάση δεδομένων. Ο υπολογισμός της τοποθεσίας του χρήστη είναι μία πολύ απλή υλοποίηση, αν ο χρήστης ανήκει στην κατηγορία του «Μη Συνδεδεμένου Χρήστη», αφού λαμβάνει υπόψιν του το γεωγραφικό μήκος και πλάτος, την ταχύτητα και την κατεύθυνση, σε εκτιμώμενο χρόνο των 3 δευτερολέπτων, υπολογίζει την απόσταση που θα μπορούσε να διανύσει ο χρήστης. Μόλις υπολογιστεί η απόσταση, και με βάση την κατεύθυνση του χρήστη, υπολογίζεται το νέο γεωγραφικό πλάτος και μήκος του χρήστη, πριν αυτός βρεθεί στο υποτιθέμενο εκτιμώμενο γεωγραφικό σημείο. Η επόμενη τοποθεσία αποθηκεύεται τοπικά σε μία λίστα η οποία συνεχώς αυξομειώνεται, ανάλογα με την χρήση της εφαρμογής. Για την κατηγορία του «Ατόμου» ή «Αυτοκινήτου», περιγράφεται εκτενώς σε επόμενο κεφάλαιο. Ένας χρήστης, για να ειδοποιηθεί αν είναι σε μικρή απόσταση κάποιος άλλος χρήστης, υπολογίζεται η απόσταση μεταξύ των εκτιμώμενων σημείων τους και όχι των πραγματικών τους. Ο υπολογισμός αυτός επηρεάζεται και από έναν τύπο βαρύτητας, ο οποίος έχει σχέση με τον χρόνο και την ταχύτητα του χρήστη. Αν η βαρύτητα δεν ξεπεράσει ένα threshold, τότε η απόσταση αυτή γίνεται αποδεκτή. Αν η απόσταση είναι μικρότερη ή ίση με μία τιμή (στην

περίπτωσή μας 5 μέτρα), τότε ο χρήστης λαμβάνει ειδοποίηση στο κινητό του, ότι ίσως κάποια περίσταση πραγματοποιηθεί στην γύρω περιοχή του με αποτέλεσμα την ανάγκη ανάληψης προσοχής. Υπάρχουν κι άλλες πολλές λεπτομέρειες, οι οποίες είναι ορθό να περιγραφούν στις αρμόζουσες ενότητες. Αξίζει να σημειωθεί ότι χρησιμοποιείται το κινητό για την λήψη δεδομένων. Με άλλα λόγια, το κινητό λειτουργεί ως επιταχυντής, γυροσκόπιο και δορυφόρος λήψης τοποθεσίας, αρκεί ο χρήστης να είναι συνδεδεμένος στο Internet. Τα δεδομένα μεταφέρονται για υπολογισμό σε έναν υπολογιστή, έτσι ώστε να μην καταναλώνονται πόροι του χρήστη. Παρ ‘όλα αυτά, ο Server πρέπει να έχει εκκινηθεί πριν την χρήση της εφαρμογής. Τέλος, η υλοποίηση ακολουθεί ένα είδος MVC μοντέλου. Πιο συγκεκριμένα, τα δεδομένα λαμβάνονται από την μεριά του χρήστη, αλλά ο υπολογισμός πραγματοποιείται στην μεριά του εξυπηρετητή. Μόλις η επεξεργασία ολοκληρωθεί, αποστέλλει τα αποτελέσματα στον κατάλληλο χρήστη. Τότε μπορεί να ειδοποιηθεί, αλλάζοντας το γραφικό περιβάλλον του χρήστη.

3.2 Περιγραφή υλοποίησης

Η υλοποίηση χωρίστηκε σε 2 βασικά μέρη, την πλευρά του χρήστη και την πλευρά του εξυπηρετητή. Η πρώτη μεριά αποτελείται από την εφαρμογή Android, την οποία χειρίζεται ο χρήστης, Client. Η άλλη μεριά αποτελείται από τον εξυπηρετητή, Server, ο οποίος διαχειρίζεται τους υπολογισμούς και τις εναλλαγές των δεδομένων μεταξύ των δύο πλευρών. Υλοποιεί το πρωτόκολλο MQTT, το οποίο θα εξεταστεί περαιτέρω σε επόμενη ενότητα. Εκτός από τις δύο κύριες πλευρές, υπάρχουν και άλλα συστήματα που λαμβάνουν μέρος, έτσι ώστε να δίνονται περισσότερες επιλογές στους χρήστες (σύνδεση/εγγραφή χρήστη, αποθήκευση δεδομένων, στατιστικά, επιλογές/ρυθμίσεις, εμφάνιση πληροφοριών σε χάρτη). Ακολουθεί ένα γενικό διάγραμμα συστατικών που περιγράφει συνοπτικά την όλη υλοποίηση.



Διάγραμμα 1 : Γενικό Διάγραμμα Συστατικών

3.2.1 Επισκόπηση MQTT

Το MQTT αντιπροσωπεύει το ακρωνύμιο MQ Telemetry Transport. Πρόκειται για ένα publish/subscribe, εξαιρετικά απλό και ελαφρύ πρωτόκολλο ανταλλαγής μηνυμάτων, σχεδιασμένο για περιορισμένες συσκευές και δίκτυα χαμηλού εύρους ζώνης, υψηλής καθυστέρησης ή αναξιόπιστων δικτύων. Οι αρχές σχεδιασμού είναι να ελαχιστοποιηθούν οι απαιτήσεις για το εύρος ζώνης δικτύου και τις πηγές των συσκευών, ενώ ταυτόχρονα επιχειρείται η διασφάλιση της αξιοπιστίας και ο βαθμός διασφάλισης της παράδοσης. Αυτές οι αρχές καταλήγουν επίσης να καταστήσουν το πρωτόκολλο ιδανικό για τον κόσμο των συνδεδεμένων συσκευών "machine-to-machine" (M2M) ή "Internet of Things" και για κινητές εφαρμογές όπου το εύρος ζώνης και η ισχύς της μπαταρίας είναι εξαιρετικά υψηλές. Το συγκεκριμένο πρωτόκολλο ξεκίνησε από τους Dr. Andy Stanford-Clark της IBM και Arlen Nipper της Arcom (τώρα Eurotech), το 1999. Για την υλοποίηση της εργασίας, χρησιμοποιήθηκε το Eclipse Paho [7], το οποίο μπορεί να υλοποιηθεί και σε Android εφαρμογή. Ως σταθερή θύρα, χρησιμοποιήθηκε η θύρα 8883 που είναι κρυπτογραφημένη, μέσω SSL. Αλλά υπάρχει και η θύρα 1883 για MQTT TCP/IP.

Ως προς την υλοποίηση, τόσο ο χρήστης όσο και ο εξυπηρετητής, λειτουργούν ως χρήστες του συγκεκριμένου πρωτοκόλλου. Πιο συγκεκριμένα, ανταλλάσσουν μηνύματα μέσω της θύρας 8883, σε μορφή JSON. Στην Android εφαρμογή, τα μηνύματα δημιουργούνται και στέλνονται μέσω του MQTT μέσω του «MainService», το οποίο υλοποιεί τον LocationCallback, υπολογίζοντας κάθε στιγμή την τοποθεσία του χρήστη. Η συγκεκριμένη τοποθεσία, μεταφέρεται μέσω του MQTT, στον Client που αποτελεί το

Backend [12] μέρος της υλοποίησης. Το Backend με την σειρά του, μέσω του handleInfo.py [3], στέλνει πίσω την επόμενη τοποθεσία του χρήστη, έχοντας πριν αποθηκεύσει τα στοιχεία του Android χρήστη σε τοπική λίστα. Παρακάτω φαίνεται η υλοποίηση του MQTT τόσο για τον Client όσο και για τον Server.

Από την μεριά του Client για την ανταλλαγή μηνυμάτων:

```
1. //MQTT send info to server
2. mqttHelperHandleInfo.publishMessage(Utils.createJsonFromState(getApplicationCo
   ntext(), state, id, search, category));
```

Μέσω του παραπάνω κώδικα, γίνεται αποστολή της τωρινής τοποθεσίας του χρήστη, στον Server, σε κατάλληλη μορφή JSON, στο θέμα που έχει γίνει εγγραφή.

```
1.  private MqttHelper startMqtt() {
2.          // Connect to topic with suffix of the id of the user
3.          MqttHelper helper = new MqttHelper(getApplicationContext(), ((SharedVariable
    s) getApplication()).getId());
4.          helper.setCallback(new MqttCallbackExtended() {
5.              @Override
6.              public void connectComplete(boolean b, String s) {
7.                  Logger.i("Connection completed", s);
8.              }
9.
10.             @Override
11.             public void connectionLost(Throwable throwable) {
12.                 Logger.e(throwable,"Connection lost");
13.             }
14.
15.             @Override
16.             public void messageArrived(String topic, MqttMessage mqttMessage) {
17.                 try {
18.                     JSONObject responseJSON = new JSONObject(mqttMessage.toString
    ());
19.                     Logger.i("Response from server: " + responseJSON.toString());
20.                     if (responseJSON.has(JSON_PURPOSE)) {
21.                         String purpose = responseJSON.getString(JSON_PURPOSE);
22.                         if (purpose.equals(JSON_PURPOSE_CAL)) {
23.                             latNext = responseJSON.getString(JSON_LAT);
24.                             lnNext = responseJSON.getString(JSON_LON);
25.                             latPrev = responseJSON.getString(JSON_LAT_PREV);
26.                             lnPrev = responseJSON.getString(JSON_LON_PREV);
```

```
27.                     ((SharedVariables) getApplication()).getChangedLocation().setLat
    New(latNext);
28.                     Logger.i("New location geocodes are: lat: " + latNext + " ,lon: " + ln
    Next);
29.                 }
30.             } else if (responseJSON.has(JSON_MIN)) {
31.                 String minDistance = responseJSON.getString(JSON_MIN);
32.                 if (minDistance != null) {
33.                     minDistanceNum = Float.valueOf(minDistance);
34.                 }
35.             }
36.         } catch (JSONException e) {
37.             Logger.e(e,"Unable to get JSON Object from Server.");
38.         }
39.     }
40.
41.     @Override
42.     public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
43.         Logger.i("Delivery Completed");
44.     }
45. });
46.
47.     return helper;
48. }
```

Η συγκεκριμένη συνάρτηση startMQTT [2], καλείται κατά την αρχικοποίηση του «MainService» και αν κατά την διάρκεια της ιχνηλάτησης, λάβει μήνυμα από τον Server, προσπαθεί να διεκπεραιώσει κάποια λειτουργία. Αυτή μπορεί να είναι είτε να στείλει τα νέα δεδομένα στον χάρτη προς απεικόνιση, είτε να λάβει την μικρότερη απόσταση των γύρω χρηστών και να εμφανίσει κατάλληλο Notification.

Από την μεριά του Server για την υλοποίηση MQTT έχουμε:

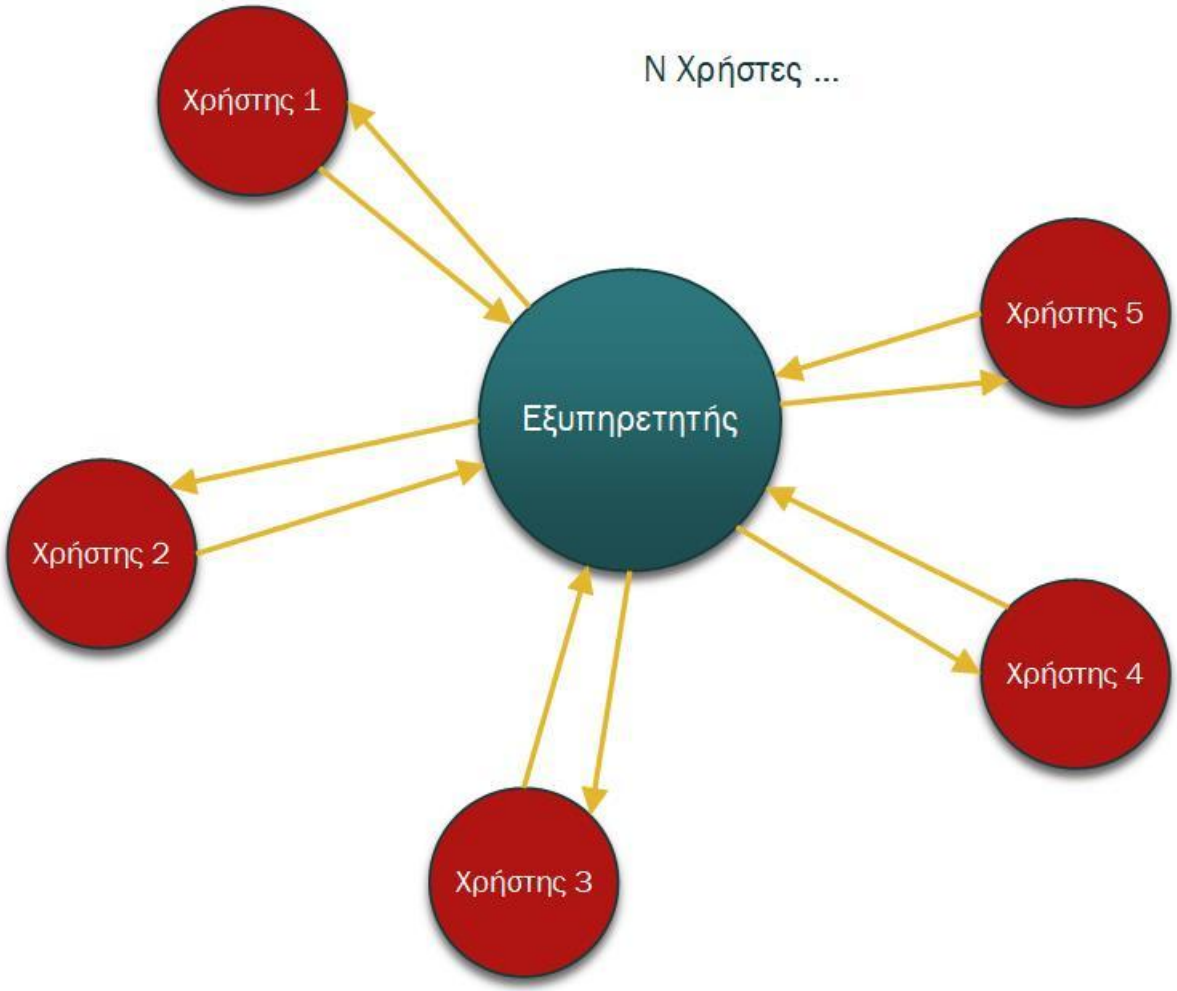
```
1. import paho.mqtt.client as mqtt
2. import ast
3. import ssl
4. import time
5. import sys
6. from calculateDistance import *
7. from dataHandle import *
```

```
8.
9.  #MQTT functions
10. def on_connect(mqttc, userdata, flags, rc):
11.     print('connected...rc=' + str(rc))
12.
13. def on_disconnect(mqttc, userdata, rc):
14.     print('disconnected...rc=' + str(rc))
15.
16. def on_message(mqttc, userdata, msg):
17.     print('message received...')
18.     #get dictionary
19.     dict = ast.literal_eval(str(msg.payload))
20.     print dict
21.     purpose = dict.get('purpose')
22.     #need to calculate next location of user
23.     if(purpose == 'calculate'):
24.         stepLength = 0.0
25.         velocity = 0.0
26.         if 'velocity' in dict:
27.             velocity = dict.get('velocity')
28.         if 'step_length' in dict:
29.             stepLength = dict.get('step_length')
30.         #calculate next position only if user is moving
31.         publishPath = 'StreetAdvisor/handleInfo/' + dict.get('id')
32.         if(velocity > 0.0 or stepLength > 0.0):
33.             print('Calculate next location...')
34.             nextLocation = calculateNextLocation(dict)
35.         else:
36.             if(dict.get('category') == 'person'):
37.                 print("Default next location for person")
38.                 nextLocation = {'purpose': 'calculated', 'id': dict.get('id'), 'latitude': dict.get('l
    atitude'), 'longitude': dict.get('longitude'), 'search': dict.get('search'), 'category': dict.ge
    t('category'), 'timestamp': dict.get('timestamp'), 'step_length': dict.get('step_length'), '
    step_counter': dict.get('step_counter'), 'lat_prev': dict.get('latitude'), 'lon_prev': dict.ge
    t('longitude')}
39.             else:
40.                 print("Default next location for general/car categories")
```

```
41.                 nextLocation = {'purpose': 'calculated', 'id': dict.get('id'), 'latitude': dict.get('l
    atitude'), 'longitude': dict.get('longitude'), 'search': dict.get('search'), 'category': dict.ge
    t('category'), 'timestamp': dict.get('timestamp'), 'velocity': dict.get('velocity'), 'lat_prev':
     dict.get('latitude'), 'lon_prev': dict.get('longitude')}
42.         print('Next Location of user ' + dict.get('id') + ' is: ')
43.         print nextLocation
44.         #send next location to current user
45.         mqttc.publish(publishPath, str(nextLocation),2)
46.         #next location need to be saved in an array if it exists or not
47.         appendUserInfo(nextLocation)
48.         #check if other users are nearby
49.         minDistance = getMinDistanceUser(nextLocation)
50.         #send minimum distance to current user
51.         mqttc.publish(publishPath, str(minDistance),2)
52.     elif(purpose == 'delete'):
53.         deleteUser(dict.get('id'))
54.
55. def on_subscribe(mqttc, userdata, mid, granted_qos):
56.     print('subscribed (qos=' + str(granted_qos) + ')')
57.
58. def on_unsubscribe(mqttc, userdata, mid, granted_qos):
59.     print('unsubscribed (qos=' + str(granted_qos) + ')')
60.
61. def on_publish(client, userdata, result):
62.     print("data published\n")
63.
64.
65. print('Beginning')
66. mqttc = mqtt.Client(client_id="", clean_session=True)
67. mqttc.on_connect = on_connect
68. mqttc.on_disconnect = on_disconnect
69. mqttc.on_message = on_message
70. mqttc.on_subscribe = on_subscribe
71. mqttc.on_unsubscribe = on_unsubscribe
72. mqttc.on_publish = on_publish
73. mqttc.tls_set("./certs/mosquitto.org.crt", tls_version=ssl.PROTOCOL_TLSv1_2)
74. mqttc.connect('test.mosquitto.org', 8883, 60)
75. #mqttc.tls_set("./certs/cert.pem", tls_version=ssl.PROTOCOL_TLSv1_2)
```

```
#mqttc.connect('iot.eclipse.org', 8883, 60)
#mqttc.connect('iot.eclipse.org', 1883, 60)
mqttc.subscribe(topic='StreetAdvisor/handleInfo/+', qos=2)
print('Start loop')
mqttc.loop_forever()
```

Στο κομμάτι κώδικα που αναφέρθηκε παραπάνω, πρώτα υλοποιήθηκαν οι συναρτήσεις on_connect, on_disconnect, on_message, on_subscribe, on_unsubscribe, on_publish, οι οποίες στην συνέχεια ανατίθενται στον MQTT Client. Έπειτα, ο Client συνδέεται στο test.mosquitto:8883 και κάνει εγγραφή στο θέμα StreetAdvisor/handleInfo/+'. Το «+» στο μονοπάτι, δείχνει ότι ο Εξυπηρετητής θα συνδεθεί σε όλα τα θέματα, που λαμβάνει μήνυμα. Με άλλα λόγια, γίνεται μία σύνδεση 1 προς 1, από τον Εξυπηρετητή προς τον Χρήστη, οι οποίοι ανταλλάσσουν μηνύματα μέσω του θέματος StreetAdvisor/handleInfo/tempid, όπου tempid μία μοναδική συμβολοσειρά που διακρίνει κάθε δίοδο, χρησιμοποιείται, πιο συγκεκριμένα, ως tunneling [43]. Έτσι χωρίζεται η κάθε ανταλλαγή μηνυμάτων, χωρίς ο κάθε χρήστης να λαμβάνει και μηνύματα από άλλους χρήστες. Ως συνέπεια, η εφαρμογή καθίσταται πιο ασφαλή και έχει μία πιο ακέραια σύνδεση. Ένα διάγραμμα που περιγράφει την συγκεκριμένη τεχνική, είναι το εξής:



Εικόνα 2 : Αρχιτεκτονική Συστήματος

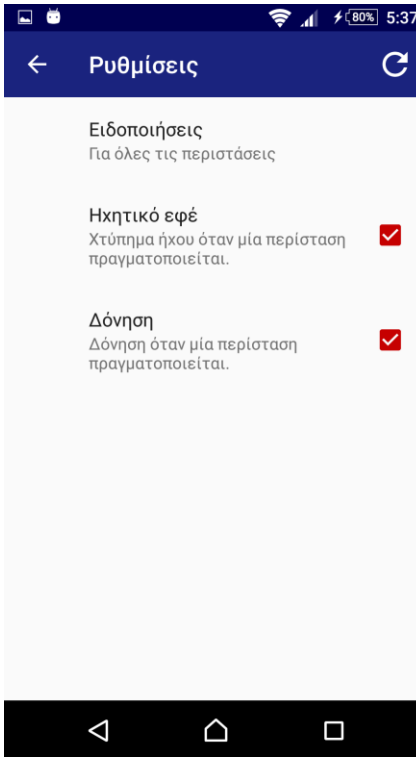
Η κύρια συνάρτηση που υλοποιήθηκε είναι η on_message, που καλείται όταν ο Server λαμβάνει ένα μήνυμα, στην περίπτωσή μας, ένα JSON μήνυμα. Στην συνέχεια, και για κάθε περίπτωση, γίνεται publish, δηλαδή αποστολή της απάντησης του Server πίσω στο θέμα που έχουν εγγραφεί όλοι οι χρήστες. Για περαιτέρω εξήγηση των συναρτήσεων, υπάρχει σχετική αναφορά στην αντίστοιχη ενότητα.

3.2.2 Μέρος χρήστη

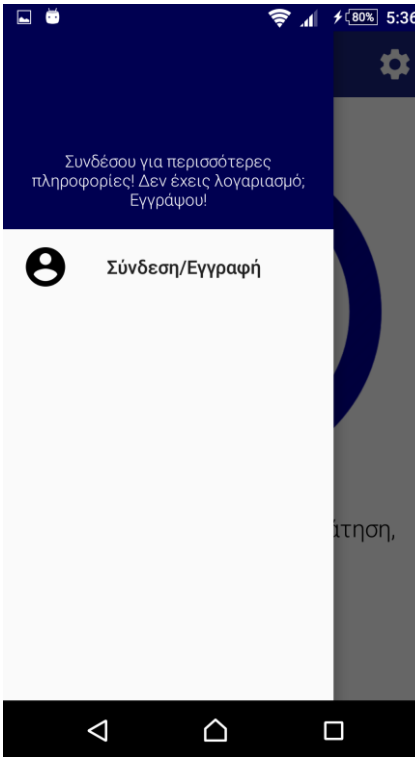
Το μέρος του χρήστη, δηλαδή η Android εφαρμογή, χωρίζεται σε δύο μέρη. Στο εικονικό μέρος θα περιγραφούν τα Activities [6] και η εμφάνισή του, ενώ στο λειτουργικό, θα αναλυθούν οι κλάσεις και οι μεταξύ τους αλληλεπιδράσεις για τον υπολογισμό αναγκαίων τιμών.

3.2.2.1 UI Modules & Λειτουργίες

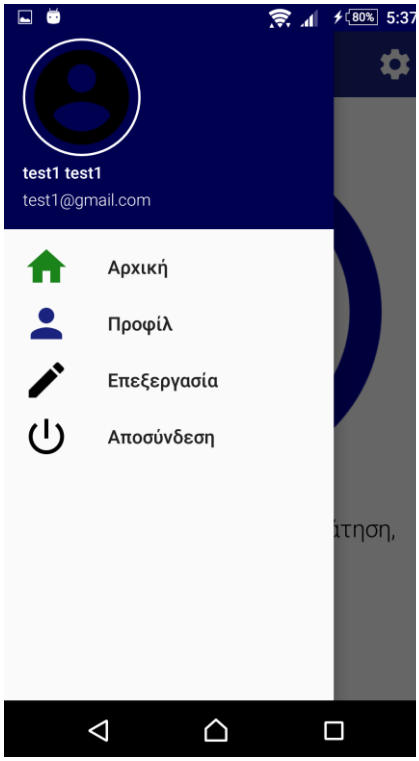
Μία Android εφαρμογή αποτελείται από το UI [19] και τα Activities πίσω από την εμφάνιση κάθε οθόνης. Κάθε Activity ή Fragment, ανάλογα με την αρχιτεκτονική υλοποίησης κάθε προγραμματιστή, μεταφέρει τα δεδομένα στην οθόνη και έπειτα εμφανίζει αυτά τα δεδομένα, μέσω ενός συγκεκριμένου αρχείου .xml. Περισσότερες πληροφορίες θα ανατεθούν στην βιβλιογραφία. Αντ' αυτού, θα δοθούν εξηγήσεις ως προς την πλοήγηση του χρήστη μέσω εικόνων και διαγραμμάτων. Παρακάτω βλέπουμε όλες τις εικόνες για τις οθόνες που διαθέτει η εφαρμογή αλλά και τις μεταξύ τους συνδέσεις, με το κατάλληλο διάγραμμα.



Εικόνα 3 : Ρυθμίσεις



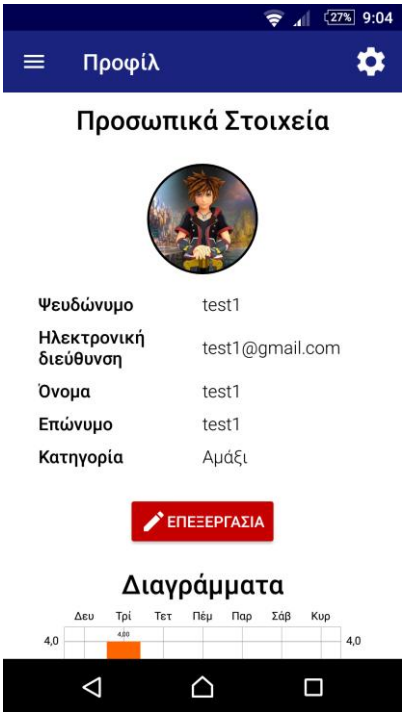
Εικόνα 4 : Drawer – Σύνδεση/Εγγραφή



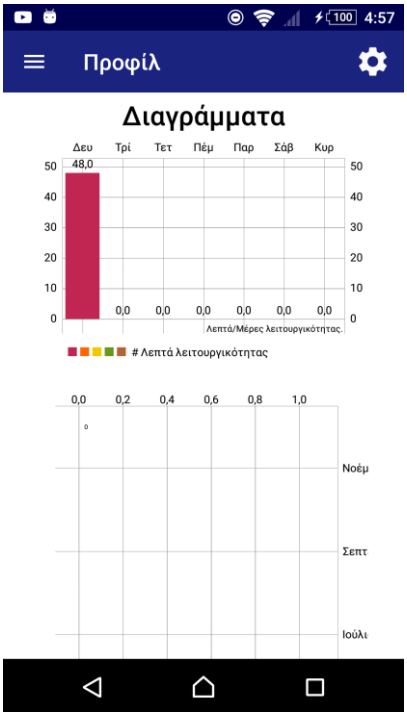
Εικόνα 5 : Επιλογές Drawer – Συνδεδεμένος Χρήστης

Η Εικόνα 3 δείχνει την οθόνη Ρυθμίσεων. Ο χρήστης μπορεί να αλλάξει τί είδος ειδοποιήσεων επιθυμεί να δέχεται, αν η συσκευή δονείται ή αν η ειδοποίηση έχει ήχο. Η Εικόνα 4 δείχνει το Drawer όταν ο χρήστης δεν είναι ακόμη συνδεδεμένος.

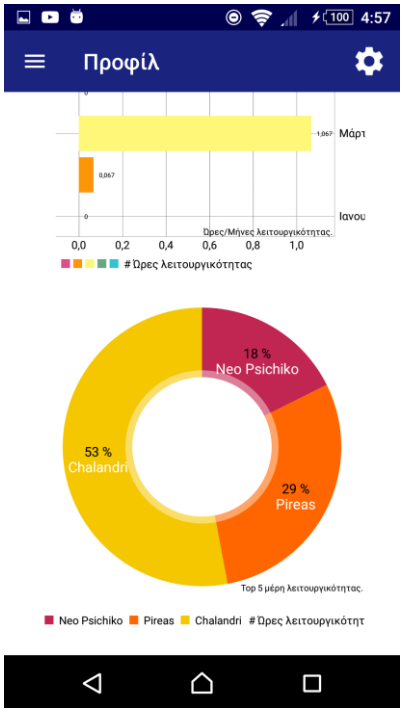
Η Εικόνα 5 δείχνει το Drawer όταν ο χρήστης είναι συνδεδεμένος. Με αυτό τον τρόπο, μπορεί να πλοηγηθεί στο Προφίλ του, να το επεξεργαστεί ή απλά να αποσυνδεθεί.



Εικόνα 6 : Προφίλ

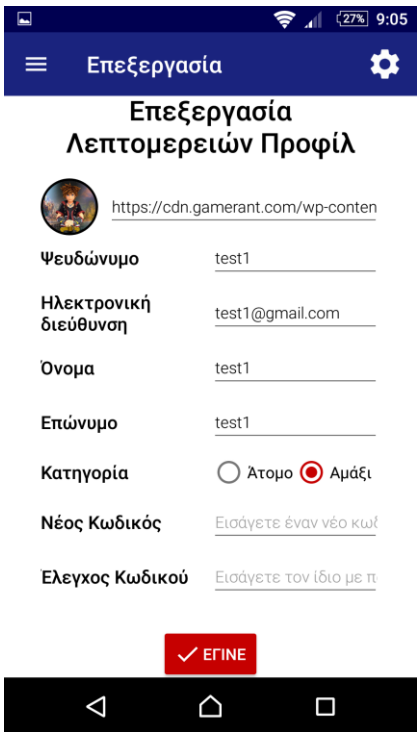


Εικόνα 7 : Προφίλ – Στατιστικά 1

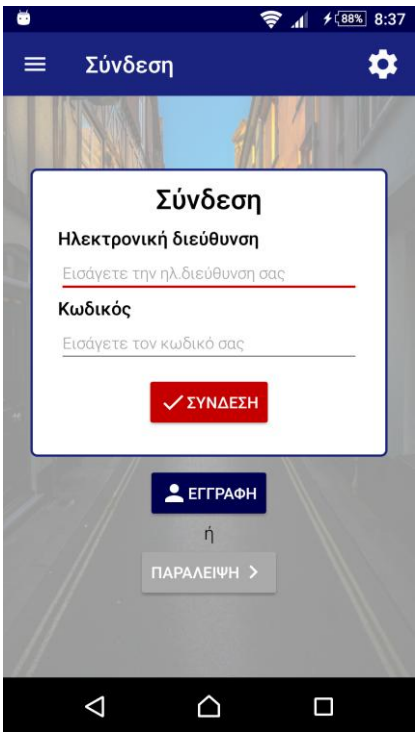


Εικόνα 8 : Προφίλ – Στατιστικά 2

Οι Εικόνες 6, 7, 8 απεικονίζουν την οθόνη Προφίλ του χρήστη, με τα στοιχεία που έδωσε κατά την εγγραφή του, αλλά και τα στατιστικά από την λειτουργία ιχνηλάτησής του. Τα στατιστικά του χρήστη που αφορούν τις ημέρες τις εβδομάδας, αρχικοποιούνται κάθε Δευτέρα. Μέσω του Προφίλ αλλά και του Drawer, μπορεί να πλοηγηθεί στην οθόνη για την Επεξεργασία των στοιχείων του, η οποία αποτελεί την Εικόνα 9.



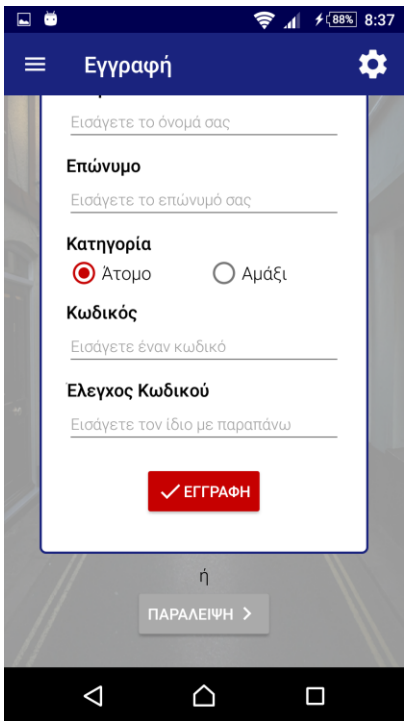
Εικόνα 9 : Επεξεργασία Προφίλ



Εικόνα 10 : Σύνδεση

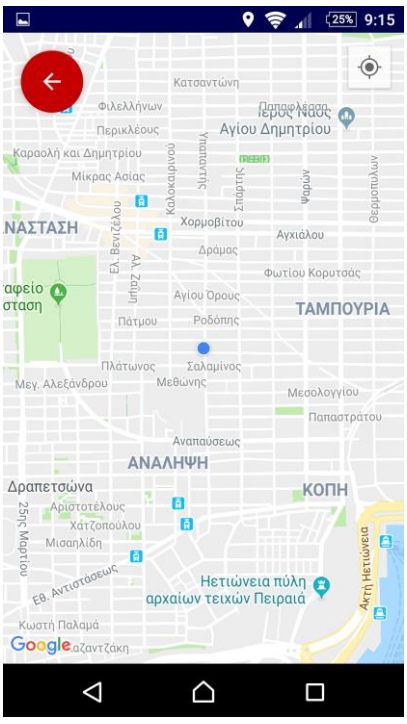
Εικόνα 11 : Εγγραφή 1

Οι Εικόνες 10, 11 , 12 απεικονίζουν είτε την οθόνη Σύνδεσης στον υπάρχον λογαριασμό του χρήστη, είτε στην εκ νέου δημιουργία του, μέσω της εγγραφής. Οι Εικόνες 13, 14, 15, 16 δείχνουν τον χάρτη όταν το MainService είναι ενεργό. Η μπλε κουκίδα αποτελεί την τωρινή τοποθεσία του χρήστη, το κόκκινο σημάδι, την προβλεφθείσα ενώ το μαύρο την αρχική.

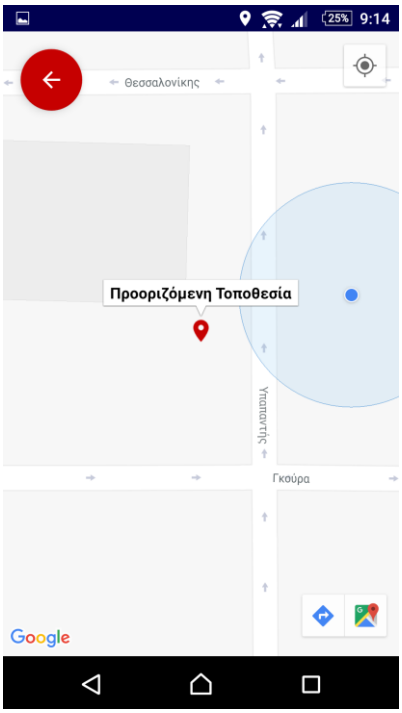


Εικόνα 12 : Εγγραφή 2

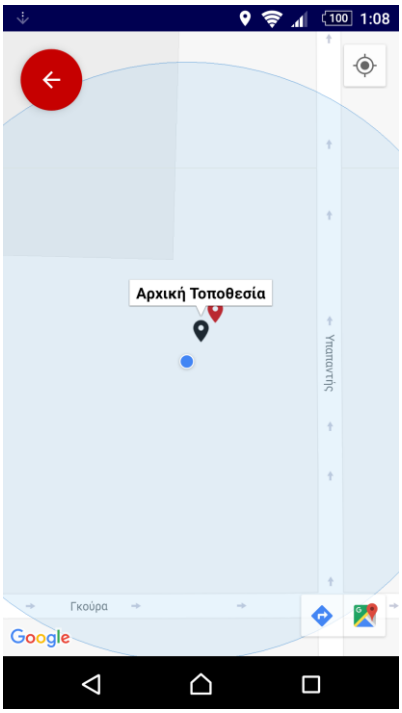
Εικόνα 13 : Χάρτης



Εικόνα 14 : Χάρτης Εστίαση



Εικόνα 15 : Χάρτης Σχέση Θέσεων 1

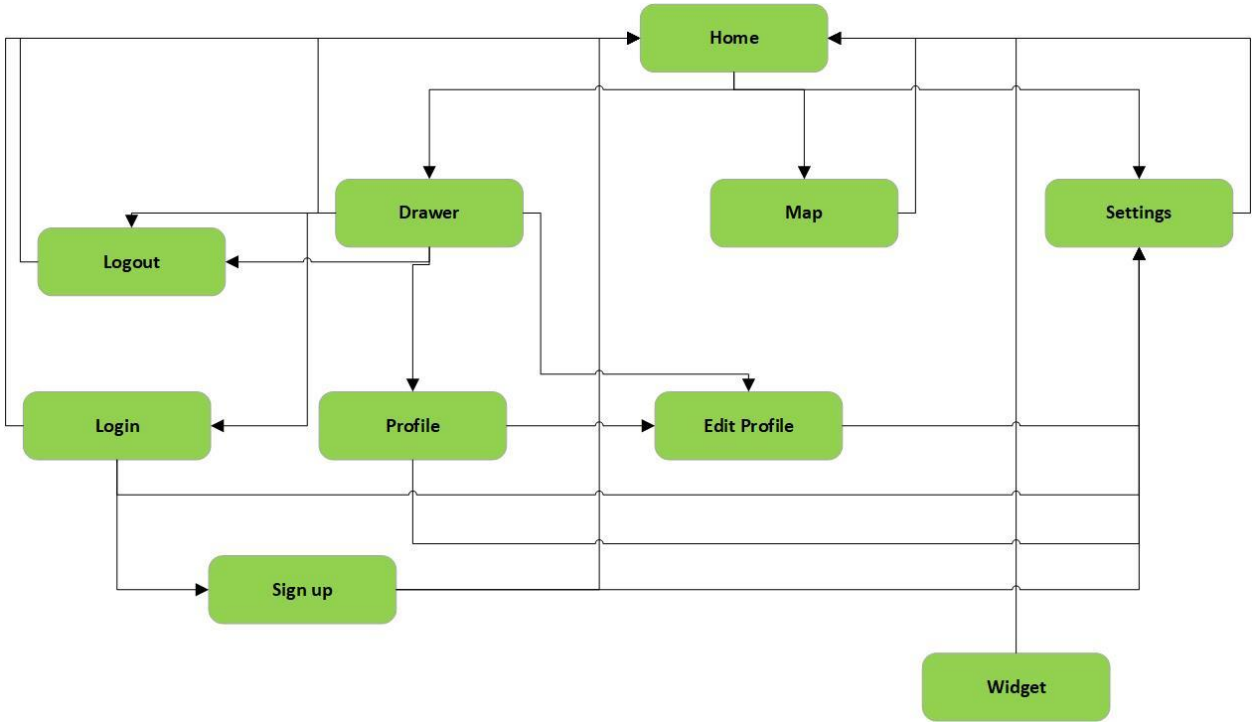


Εικόνα 16 : Χάρτης Σχέση Θέσεων 2

Η Εικόνα 17 αποτελεί απεικόνιση του widget της εφαρμογής. Ο χρήστης μπορεί να εκτελέσει την εφαρμογή χωρίς να χρειαστεί να ανοίξει την κύρια εφαρμογή, αφού το γραφικό περιβάλλον του συνδέεται με την υλοποίηση της εφαρμογής.

Εικόνα 17 : Widget [9]

Το ακόλουθο διάγραμμα απεικονίζει την πλοήγηση που μπορεί να επιτευχθεί ανάμεσα στις οθόνες που προαναφέρθηκαν.



Διάγραμμα 2 : Διάγραμμα Σχέσεων Οθονών [8]

3.2.2.2 Backend Modules

Κάθε εφαρμογή Android διαθέτει και ένα Thread [18] μέσω του οποίου γίνονται όλες οι ενέργειες για να λάβουν δεδομένα, χωρίς να χρησιμοποιήσουν το Main Thread, δηλαδή την κύρια χρήση της εφαρμογής. Με άλλα λόγια, μία εφαρμογή είναι χωρισμένη σε δύο διακριτά μέρη, αυτό της εμφάνισης, το λεγόμενο UI, γραμμένο κατά κόρων σε γλώσσα XML, και το κομμάτι της υλοποίησης, δηλαδή του υπολογισμού τιμών προς απεικόνιση, το λεγόμενο Backend ή αλλιώς την εκτέλεσή του στο παρασκήνιο, γραμμένο σε γλώσσα JAVA.

Στην συγκεκριμένη υλοποίηση, έχει γίνει εκτενής χρήση του, τόσο για την αποστολή των δεδομένων τοποθεσίας χρήστη στον Backend Server προς υπολογισμό της επόμενης προβλεφθείσας τοποθεσίας, όσο και στην εμφάνιση του λογαριασμού του και την απεικόνιση και υπολογισμό των στατιστικών του. Αρχικά, χρησιμοποιήθηκε το Service «MainService», το οποίο ενεργοποιείται όταν ο χρήστης πατήσει το «Play Button» στην Αρχική Σελίδα και απενεργοποιείται με τον ίδιο ακριβώς τρόπο. Υλοποιεί τον LocationCallback, κλάση προσφερόμενη από την Google, για την συνεχόμενη ενημέρωση αλλαγής τοποθεσίας του χρήστη. Η λειτουργία ιχνηλάτησης που προσφέρει η προαναφερόμενη κλάση εκτελείται στο παρασκήνιο έως ότου ο χρήστης απενεργοποιήσει αυτή ή το κινητό. Νέα τοποθεσία του χρήστη λαμβάνεται κάθε 3 δευτερόλεπτα όταν ο χρήστης κινείται ενώ κάθε 6 δευτερόλεπτα όταν η κατάσταση του είναι αδρανής. Η κλήση για καινούρια τοποθεσία δεν συσχετίζεται και δεν ενεργοποιείται όταν ο χρήστης αλλάζει ταχύτητα, αλλά μόνο κάθε χρονική περίοδο 3 ή 6 δευτερολέπτων. Με αυτόν τον τρόπο, καθίσταται συνέπεια και συνοχή, αφού ο Εξυπηρετητής δεν λαμβάνει συνεχώς δεδομένα προς επεξεργασία. Επιπρόσθετα, επειδή υπολογίζεται η επόμενη τοποθεσία σε μελλοντικό χρόνο (λήψη τοποθεσίας μετά την πάροδο/κίνηση του χρήστη για 3 δευτερόλεπτα), είναι εφικτός ο καλύτερος συγχρονισμός των δύο μερών (Εξυπηρετητή / Χρήστη) Ενδεικτικά, ένα κομμάτι κώδικα είναι το εξής:

```
1.  // Start Location Listener
```

```
2.  mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
3.  if (ActivityCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS
    _FINE_LOCATION) == PackageManager.PERMISSION_GRANTED && ActivityCo
    mpat.checkSelfPermission(this, android.Manifest.permission.ACCESS_COARSE_L
    OCATION) == PackageManager.PERMISSION_GRANTED) {
4.      mFusedLocationClient.requestLocationUpdates(mLocationRequest, mLocationCa
    llback, null);
5.  }
```

Στο συγκεκριμένο κομμάτι κώδικα, αρχικοποιείται το FusedLocationProviderClient στην αρχικοποίηση του Service, θέτοντας αρχικές τιμές, όπως απόσταση που πρέπει να διανύσει ο χρήστης μέχρι να ανανεωθεί η τοποθεσία του. Στο επόμενο κομμάτι κώδικα, υλοποιείται η όλη διαδικασία χρήσης του Location Object, αντικειμένου που περιέχει στοιχεία τοποθεσίας χρήστη (γεωγραφικό μήκος και πλάτος, ταχύτητα, κατεύθυνση κ.ο.κ.), για την αποθήκευσή του στα στατιστικά και την αποστολή του στον Server για λήψη επόμενης τοποθεσίας. Στην συνέχεια, αν οι γύρω χρήστες είναι σε απόσταση μικρότερη ή ίση των 15 μέτρων, εμφανίζει στον χρήστη προειδοποιητικό μήνυμα για λήψη προσοχής.

```
1.  // Create LocationCallback
2.  private LocationCallback mLocationCallback = new LocationCallback() {
3.      @Override
4.      public void onLocationResult(LocationResult locationResult) {
5.          super.onLocationResult(locationResult);
6.          if (locationResult == null) {
7.              return;
8.          }
9.          Location location = locationResult.getLastLocation();
10.         // Calculate average step length
11.         if(lastLocation != null) {
12.             double distance = location.distanceTo(lastLocation);
13.             if(numberOfSteps == 0){
14.                 stepLength = 0;
15.             } else {
16.                 stepLength = distance / numberOfSteps;
17.             }
18.         } else{
19.             // This is the first time we set a location as last, no need for further calculation
    s
20.             lastLocation = location;
21.             return;
22.         }
```

```
23.         double elapsedTime = (double)(location.getTime() -
     lastLocation.getTime()) / 1000; // Convert milliseconds to seconds
24.         double calculatedSpeed = lastLocation.distanceTo(location) / elapsedTime;
25.         double speed = location.hasSpeed() ? location.getSpeed() : calculatedSpeed;
26.         double calculatedBearing = lastLocation.bearingTo(location);
27.         double bearing = location.hasBearing() ? location.getBearing() : calculatedBea
    ring;
28.         Logger.i("Location geocodes: lat: " + location.getLatitude() + ", lon: " + location.
    getLongitude());
29.         // For statistics
30.         sendToJobService(String.valueOf(location.getLatitude()), String.valueOf(locatio
    n.getLongitude()));
31.         // For main function
32.         Logger.i("Speed: " + speed + ", bearing: " + bearing);
33.         location.setSpeed((float)speed);
34.         location.setBearing((float)bearing);
35.         Object data = createDataObject(location);
36.         if(data instanceof DataPerson){
37.             ((DataPerson) data).setNumberOfSteps(numberOfSteps);
38.         }
39.         // Calculate and return next location
40.         // MQTT send info to server
41.         ((SharedVariables) getApplication()).getMqttHelperHandleInfo().publishMessag
    e(Utils.createJsonForData(getApplicationContext(), data, ((SharedVariables) getAppl
    ication()).getId(), search));
42.         if (minDistanceNum <= 15 && minDistanceNum != 0.0) {
43.             // Display notification
44.             alterNotification();
45.             notificationManager.notify(NOTIFICATION_ID, mBuilder.build());
46.         }
47.         lastLocation = location;
48.         numberOfSteps = 0;
49.     }
50.};
```

Ακόμη, το MainService λειτουργεί ως αισθητήρας για τους τύπους δεδομένων που θέλουμε να λάβουμε. Η διαχείρισή τους γίνεται από έναν SensorListener, οποίος για κάθε περίπτωση, λαμβάνει και επεξεργάζεται τα στοιχεία των ανιχνευτών. Το κομμάτι κώδικα είναι το εξής:

```
1.  private SensorEventListener mSensorEventListener = new SensorEventListener() {
```

```
2.
3.      public void onAccuracyChanged(Sensor sensor, int accuracy) {}
4.
5.      @Override
6.      public void onSensorChanged(SensorEvent event) {
7.          float x;
8.          float y;
9.          switch (event.sensor.getType()) {
10.             case Sensor.TYPE_ACCELEROMETER:
11.                 x = event.values[0];
12.                 y = event.values[1];
13.                 double z = event.values[2];
14.                 // Last acceleration including gravity
15.                 double mAccelLast = accelCurrent;
16.                 accelCurrent = Math.sqrt((x * x + y * y + z * z));
17.                 double delta = accelCurrent - mAccelLast;
18.                 // Perform low-cut filter
19.                 accelerationNoGravivy = accelerationNoGravivy * 0.9f + delta;
20.                 break;
21.             case Sensor.TYPE_GYROSCOPE:
22.                 yawRate = Math.round(event.values[1]);
23.                 break;
24.             case Sensor.TYPE_STEP_DETECTOR:
25.                 numberOfSteps++;
26.                 break;
27.             default:
28.                 break;
29.         }
30.     }
31. };
```

Άλλο μέρος της υλοποίησης του Client που αποτελεί μηχανισμό παρασκηνίου είναι το Google Firebase Realtime Database & Authentication. Για την σύνδεση, αποσύνδεση ή εγγραφή του χρήστη στην πλατφόρμα της εφαρμογής, πρέπει να είναι υλοποιημένες ορισμένες συναρτήσεις. Για την αποθήκευση των δεδομένων του χρήστη, πρέπει να δημιουργηθεί λογαριασμός στην κονσόλα Firebase της Google και να συνδεθεί η εφαρμογή με αυτήν. Έπειτα, χρειάζεται να βρεθεί το ζητούμενο DatabaseReference, μέσω του οποίου θα γίνει αποθήκευση των δεδομένων στην μορφή που προτείνει η Google, δηλαδή JSON. Ένα παράδειγμα είναι το εξής:

```
1.  public static void updateUser(Context context, final String username, final String n
    ame, final String surname, final String imageSrc, final String category) {
2.      final FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
3.      if (user == null) {
4.          Logger.e("updateUser: User is empty.");
5.          return;
6.      }
7.      String uid = user.getUid();
8.      DatabaseReference mDatabaseRef;
9.      mDatabaseRef = FirebaseDatabase.getInstance().getReference(TAG_USERS).c
    hild(uid);
10.     mDatabaseRef.child(TAG_USERNAME).setValue(username);
11.     mDatabaseRef.child(TAG_NAME).setValue(name);
12.     mDatabaseRef.child(TAG_SURNAME).setValue(surname);
13.     mDatabaseRef.child(TAG_IMAGESRC).setValue(imageSrc);
14.     mDatabaseRef.child(TAG_CATEGORY).setValue(category);
15.     // Add category for search
16.     ((SharedVariables) ((Activity) context).getApplication()).setCategory(category);
17.     Logger.i("Category updated: " + ((SharedVariables) ((Activity) context).getApplicat
    ion()).getCategory());
18. }
```

Στην συγκεκριμένη συνάρτηση, γίνεται ενημέρωση των πληροφοριών του χρήστη. Λαμβάνουμε από το FirebaseAuth, δηλαδή την πιστοποίηση ότι ο χρήστης είναι συνδεδεμένος στο προφίλ του, ένα αντικείμενο τύπου User. Με βάση την μοναδική συμβολοσειρά που κατέχει ως id, βρίσκουμε τα δεδομένα του με το αντικείμενο DatabaseReference, και για κάθε πεδίο, αναθέτουμε ξανά τις νέες τιμές. Το Google Firebase Authentication, διαθέτει δικές του συναρτήσεις για να απλοποιεί την διαδικασία της δημιουργίας, πιστοποίησης και αποσύνδεσης του χρήστη για την συγκεκριμένη εφαρμογή. Ένα παράδειγμα αποτελεί το μέρος κώδικα

```
1.  public static void logoutUser(Context context) {
2.      FirebaseAuth mFirebaseAuth = FirebaseAuth.getInstance();
3.      if (mFirebaseAuth == null) {
4.          Logger.e("logoutUser: User Authentication is empty.");
5.          return;
6.      }
7.      FirebaseUser firebaseUser = mFirebaseAuth.getCurrentUser();
8.      if (firebaseUser == null) {
9.          ((SharedVariables) ((Activity) context).getApplication()).setLoggedIn(false);
10.         Logger.e("logoutUser: User is empty.");
11.         return;
```

```
12.    }
13.    ((SharedVariables) ((Activity) context).getApplication()).setLoggedIn(false);
14.    String status = context.getResources().getString(R.string.status_info_off);
15.    FirebaseDatabase.getInstance().getReference(TAG_USERS).child(firebaseUser.
    getUid()).child(TAG_STATUS).setValue(status);
16.    mFirebaseAuth.signOut();
17.}
```

το οποίο λαμβάνει ένα αντικείμενο FirebaseAuth, βρίσκοντας στην συνέχεια το αντικείμενο του χρήστη και την απευθείας αποσύνδεσή του με μία γραμμή κώδικα.

Τέλος, υπάρχει περαιτέρω ένα επιπλέον μέρος του κώδικα που εκτελείται στο Background, και αυτό δεν είναι άλλο από το JobService «JobStatisticsService». Το παραπάνω, εκτελείται κάθε 2 δεύτερα και αποθηκεύει την τοποθεσία του χρήστη στην βάση δεδομένων Firebase. Ουσιαστικά, αναθέτει ένα αντικείμενο Job στο Service, με ορισμένα χαρακτηριστικά:

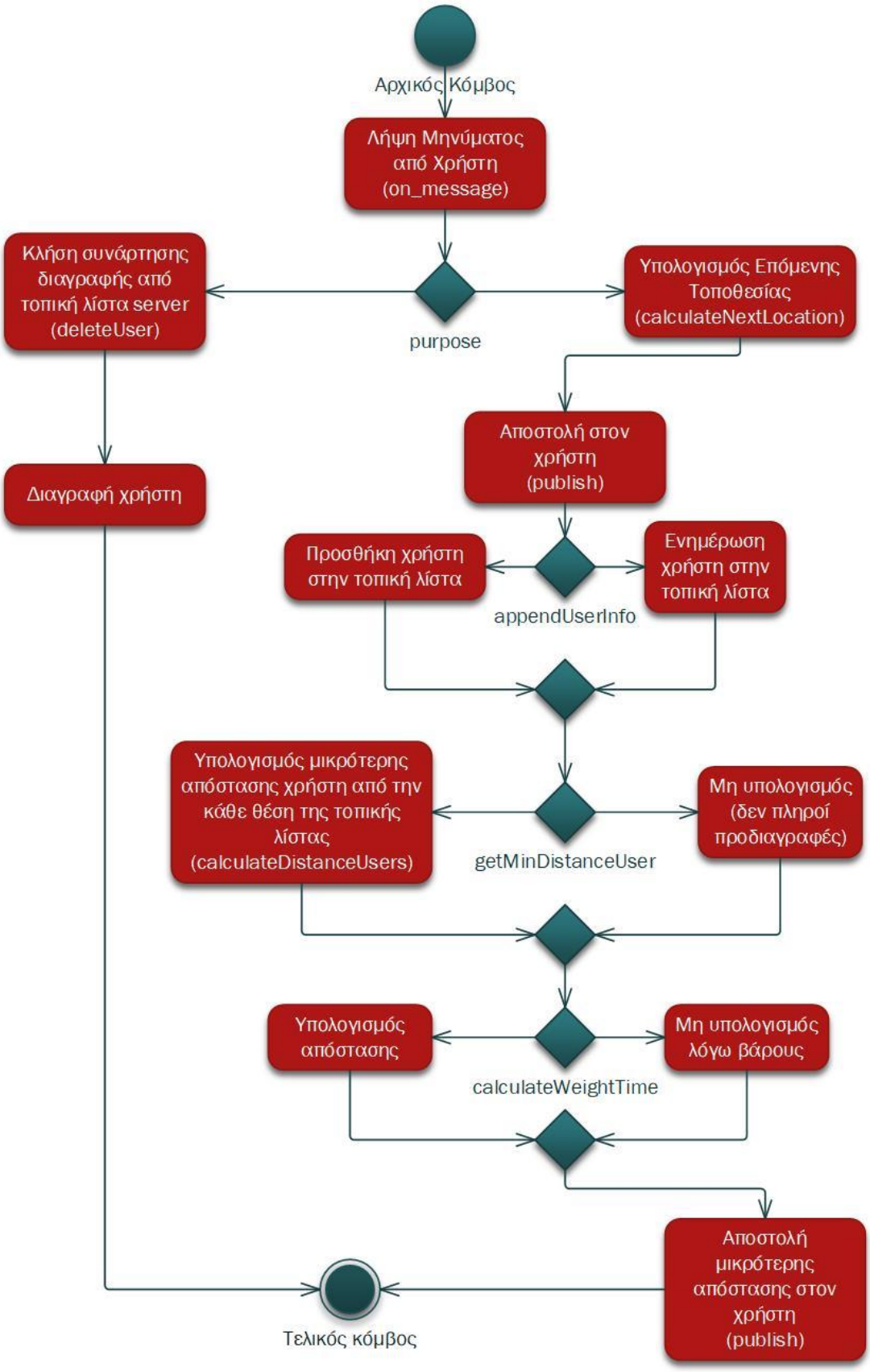
```
1.  private final BroadcastReceiver mJobDispatcherReceiver = new BroadcastReceive
    r() {
2.      @Override
3.      public void onReceive(Context context, Intent intent) {
4.          String lat = intent.getStringExtra(EXTRA_LATITUDE_JOB);
5.          String lng = intent.getStringExtra(EXTRA_LONGITUDE_JOB);
6.          if (lat != null && lng != null && ((SharedVariables) getActivity().getApplication())
    .isLoggedIn()) {
7.              Bundle args = new Bundle();
8.              Logger.i("Job schedule with args: lat: " + lat + " ,lng: " + lng);
9.              args.putString(EXTRA_LATITUDE_JOB, lat);
10.             args.putString(EXTRA_LONGITUDE_JOB, lng);
11.
12.             Job myJob = dispatcher.newJobBuilder()
13.                 .setService(JobStatisticsService.class)
14.                 .setTag(JOB_TAG)
15.                 .setRecurring(false)
16.                 .setLifetime(Lifetime.UNTIL_NEXT_BOOT)
17.                 .setTrigger(Trigger.executionWindow(0, 120))  // Every 2 minutes
18.                 .setReplaceCurrent(false)
19.                 .setRetryStrategy(RetryStrategy.DEFAULT_EXPONENTIAL)
20.                 .setConstraints(Constraint.ON_ANY_NETWORK)
21.                 .setExtras(args)
22.                 .build();
23.             dispatcher.mustSchedule(myJob);
```

```
24.         }
25.     }
26. };
```

Για πιο αναλυτικά, υπάρχουν πληροφορίες στην βιβλιογραφία.

3.2.3 Μέρος Εξυπηρετητή

Όπως προαναφέρθηκε, ο εξυπηρετητής, δηλαδή ο Server, είναι γραμμένος σε προγραμματιστική γλώσσα Python. Αποτελείται από 3 αρχεία, τα οποία είναι αλληλένδετα αλλά το καθένα είναι νοηματικά διαφορετικό από το άλλο. Το επόμενο σχήμα δείχνει πως συνδέονται μεταξύ τους αλλά και τι δεδομένα μεταφέρουν:



Διάγραμμα 3 : Διάγραμμα Ροής Backend [16][16]

3.2.3.1 Τεχνολογίες που χρησιμοποιήθηκαν και Προδιαγραφές

Για την υλοποίηση του Server, χρησιμοποιήθηκε, όπως αναφέρθηκε το MQTT πρωτόκολλο μεταφοράς μηνυμάτων του πακέτου «paho.mqtt.client». Επιπρόσθετα, η έκδοση Python με την οποία εκτελέστηκε είναι η 2.7.15. Πιο αναλυτικά, τα πακέτα που εισήχθησαν είναι :

Πίνακας 2 : Πακέτα Εισαγωγής

Τεχνολογία/βιβλιοθήκη	Έκδοση	Σκοπός
Python	2.7.15	Ολική υλοποίηση και εκτέλεση Backend
paho.mqtt.client	paho	Υλοποίηση MQTT πρωτοκόλλου
ast	-	Μετατροπή μηνύματος σε μορφή λεξικού (JSON)
datetime	datetime	Λήψη/επεξεργασία ώρας σε milliseconds
math	asin,radians,cos,sin,degrees,atan2,sqrt	Τριγωνικές συναρτήσεις και μοίρες

Όσον αφορά το περιβάλλον εκτέλεσης, το Backend μέρος της εργασίας μπορεί να εκκινηθεί με την εκτέλεση της εντολής **python handleInfo.py** ή απλά **handleInfo.py** σε μία γραμμή εντολών (command line), αρκεί ο χρήστης να βρίσκεται ήδη στον φάκελο που περιέχει το συγκεκριμένο αρχείο, πριν την εκτέλεση. Η εκτέλεση του Server συνεχίζεται επ'αόριστον. Οι προδιαγραφές συστήματος που χρησιμοποιήθηκαν είναι οι εξής:

Πίνακας 3 : Προδιαγραφές Συστήματος

Λειτουργικό Σύστημα	Windows 7 Professional 64-bit
Επεξεργαστής	Intel Core i5 2.6GHz 4 CPUs
Μνήμη	16GB RAM
Κάρτα Γραφικών	Intel HD Graphics Family

Τέλος, ήταν αναγκαίο να οριστούν κάποιες παράμετροι, δηλαδή τιμές και λειτουργίες που είναι προκαθορισμένες, τόσο για την διευκόλυνση της υλοποίησης όσο και για την τοποθέτηση ορίων στον κώδικα. Αρχικά, η ανταλλαγή των δεδομένων από και προς τον Client/Server είναι σε μορφή JSON, για την εγκαθίδρυση ενός κοινού άξονα μορφής μηνυμάτων, γνωστό στον τομέα της πληροφορικής για τα πλεονεκτήματά του. Επιπλέον, για την ανταλλαγή αυτών των μηνυμάτων μέσω MQTT, γίνεται εγγραφή του Server/Client στο θέμα με τίτλο **StreetAdvisor/handleInfo**, ενώ τα μηνύματα κρυπτογραφούνται με μία αλυσίδα πιστοποιητικών, γνωστή για το eclipse paho. Τα μηνύματα μεταφέρονται με TLS [44], δηλαδή είναι προστατευμένα από κάθε είδους διαδικτυακή επίθεση. Κομμάτια πιστοποιητικών λήφθησαν από συγκεκριμένη βάση [38], διότι ο Broker που χρησιμοποιείται είναι συμβατός με αυτόν τον τύπο πιστοποιητικών. Ένα παράδειγμα πιστοποιητικού που λειτουργεί και δέχεται την κρυπτογράφηση μηνυμάτων είναι το εξής:

1. -----BEGIN CERTIFICATE-----

2. MIIFazCCA1OgAwIBAgIRAIIQz7DSQONZRGPgu2OCiwAwDQYJKoZIhvcNAQELBQAw
3. TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh
4. cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTUwNjA0MTEwNDM4
5. WhcNMzUwNjA0MTEwNDM4WjBPMQswCQYDVQQGEwJVUzEpMCcGA1UEChMgSW50ZXJu
6. ZXQgU2VjdXJpdHkgUmVzZWFyY2ggR3JvdXAxFTATBgNVBAMTDElTUkcgUm9vdCBY
7. MTCCAiIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54rVygc
8. h77ct984kIxuPOZXoHj3dcKi/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uoxmyF+
9. 0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+3mX6U
10. A5/TR5d8mUgjU+g4rk8Kb4Mu0UlXjIB0ttov0DiNewNwIRt18jA8+o+u3dpjq+sW
11. T8KOEUt+zwvo/7V3LvSye0rgTBIlDHCNAymg4VMk7BPZ7hm/ELNKjD+Jo2FR3qyH
12. B5T0Y3HsLuJvW5iB4YlcNHlsdu87kGJ55tukmi8mxdAQ4Q7e2RCOFvu396j3x+UC
13. B5iPNgiV5+I3lg02dZ77DnKxHZu8A/lJBdiB3QW0KtZB6awBdpUKD9jf1b0SHzUv
14. KBds0pjBqAlkd25HN7rOrFleaJ1/ctaJxQZBKT5ZPt0m9STJEadao0xAH0ahmbWn
15. OlFuhjuefXKnEgV4We0+UXgVCwOPjdAvBbI+e0ocS3MFEvzG6uBQE3xDk3SzynTn
16. jh8BCNAw1FtxNrQHusEwMFxIt4I7mKZ9YIqioymCzLq9gwQbooMDQaHWBfEbwrbw
17. qHyGO0aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo//fb4hVC1CLQJ13hef4Y53CI
18. rU7m2Ys6xt0nUW7/vGT1M0NPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
19. HRMBAf8EBTADAQH/MB0GA1UdDgQWBBR5tFnme7bl5AFzgAiIyBpY9umbbjANBgkq
20. hkiG9w0BAQsFAAOCAgEAVR9YqbyyqFDQDLHYGmkgJykIrGF1XIpu+ILlaS/V9lZL
21. ubhzEFnTIZd+50xx+7LSYK05qAvqFyFWhfFQDlnrzuBZ6brJFe+GnY+EgPbk6ZGQ
22. 3BebYhtF8GaV0nxvwuo77x/Py9auJ/GpsMiu/X1+mvoiBOv/2X/qkSsisRcOj/KK
23. NFtY2PwByVS5uCbMiogziUwthDyC3+6WVwW6LLv3xLfHTjuCvjHIInNzktHCgKQ5
24. ORAzI4JMPJ+GslWYHb4phowim57iaztXOoJwTdwJx4nLCgdNbOhdjsnvzqvHu7Ur
25. TkXWStAmzOVyyghqpZXjFaH3pO3JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRlN8NwdC
26. jNPElpzVmbUq4JUagEiuTDkHzsxHpFKVK7q4+63SM1N95R1NbdWhscdCb+ZAJzVc
27. oyi3B43njTOQ5yOf+1CceWxG1bQVs5ZufpsMljq4Ui0/1lvh+wjChP4kqKOJ2qxq
28. 4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d11TPA

```
mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzwxA57d

emyPxgcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iItreGCc=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFjTCCA3WgAwIBAgIRANOxciY0IzLc9AUoUSrsnGowDQYJKoZIhvcNAQELBQAw
TzELMAkGA1UEBhMCVVMxKTAnBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh
cmNoIEdyb3VwMRUwEwYDVQQDEwxJU1JHIFJvb3QgWDEwHhcNMTYxMDA2MTU0MzU1
WhcNMjExMDA2MTU0MzU1WjBKMQswCQYDVQQGEwJVUzEW
.....
```

αποτελώντας έτσι μία αλυσίδα από πιστοποιητικά.

Το συγκεκριμένο κομμάτι αλυσίδας πιστοποιητικών, χρησιμοποιήθηκε για την σύνδεση στον broker iot.eclipse.org. Για το test.mosquitto.org, χρησιμοποιήθηκε ένα δοκιμαστικό πιστοποιητικό, το οποίο παρέχει η ιστοσελίδα του mosquitto.

Έπειτα, συνδέονται και τα δύο μέρη στην σελίδα **test.mosquitto.org,** με θύρα **8883**. Ακόμη, τα μηνύματα που μεταφέρονται μεταξύ των δύο πλευρών, έχουν ως σκοπό είτε την αποστολή τοποθεσίας είτε την εκτέλεση διαγραφής χρήστη από το σύστημα, όταν αυτός σταματήσει την ιχνηλάτηση ή γενικά αν σταματήσει να λειτουργεί η εφαρμογή. Οι συγκεκριμένες προδιαγραφές περιέχονται στο αρχείο **handleInfo.py**. Όσον αφορά την τοπική λίστα, αυτή ανανεώνεται κάθε φορά που ο χρήστης στέλνει τα δεδομένα του οπότε και αποθηκεύεται εκ νέου στην λίστα ή αν διαγραφτεί ο χρήστης όπως προαναφέρθηκε. Αν ο χρήστης έχει μηδενική ταχύτητα ή καθόλου βηματισμό (ανάλογα την κατηγορία), τότε δεν γίνεται κάποιος παραπάνω υπολογισμός, αλλά η επόμενη προβλεφθείσα τοποθεσία είναι παρόμοια (ίδια επόμενη τοποθεσία) με αυτή που δέχτηκε ο Εξυπηρετητής ως μήνυμα από την συσκευή. Η λίστα αυτή με τα δεδομένα των χρηστών διατηρείται στην μνήμη για ένα session, δηλαδή για μία εκτέλεση του Server. Επιπρόσθετα, ως μέγιστη απόσταση μεταξύ των χρηστών κατά την ιχνηλάτηση, έχει οριστεί μία μεταβλητή με όνομα **MAX_DISTANCE** και τιμή **1000 μέτρα**. Στην πλευρά του υπολογισμού, για κάθε αναζήτηση και ανάθεση μικρότερης απόστασης, γίνεται υπολογισμός αποστάσεων του τωρινού χρήστη από την τοπική λίστα, δημιουργώντας μία διαφορετική λίστα που περιέχει όλες τις αποστάσεις. Από την λίστα αυτή, επιλέγεται η μικρότερη τιμή η οποία επιστρέφεται πίσω στον χρήστη, για να αποφανθεί αν χρειάζεται να εμφανιστεί στο κινητό κάποια ειδική ειδοποίηση. Μόλις τελειώσει αυτή η αναζήτηση, η λίστα με τις αποστάσεις διαγράφεται διότι τα δεδομένα, δηλαδή οι τοποθεσίες των χρηστών, αλλάζουν συνεχώς. Στον υπολογισμό των αποστάσεων λαμβάνεται υπόψιν και σε ποια κατηγορία είναι μέλος ο χρήστης (Άτομο, Αυτοκίνητο) αλλά και τι χρήστες συμπεριλαμβάνει στον υπολογισμό (Άτομο και Αυτοκίνητο, Μόνο Άτομο, Μόνο Αυτοκίνητο, Κανένα), αλλάζοντας την ελάχιστη απόσταση. Αυτό συμβαίνει διότι, από την τοπική λίστα των χρηστών, περιορίζεται ο αριθμός αυτών που είναι δυνατό να υπολογιστεί η μεταξύ τους απόσταση. Όλα αυτά περιέχονται στο αρχείο **dataHandle.py** [3]. Στο αρχείο **calculateDistance.py** [3] υπάρχουν προδιαγραφές ορισμένες για τον υπολογισμό των τοποθεσιών. Μία από αυτές είναι ο μέγιστος χρόνος που μπορεί να προβλεφθεί μία τοποθεσία, με την μεταβλητή **TIME_SEC** και τιμή **3 δευτερόλεπτα**. Άλλη είναι η ακτίνα της γης σε μέτρα, με την μεταβλητή **R** = **6371 * 1000** μέτρα και τέλος ένα άνω βάρος δευτερολέπτων που

είναι αποδεκτό να υπολογιστεί η απόσταση μεταξύ ενός χρήστη από την λίστα και τον τωρινό χρήστη, **MAX_SECS** = **30 δευτερόλεπτα**, για χρήστη που χρησιμοποιεί την ταχύτητα για τον υπολογισμό, ενώ **MAX_SECS_PERSON = 200**, για την κατηγορία του «Ατόμου». Όπως και πριν, υπάρχει η μεταβλητή **MAX_DISTANCE** με τιμή **1000 μέτρα** αλλά και η μεταβλητή **PI = 3.14159265359**. Περαιτέρω πληροφορίες για τον υπολογισμό θα αναφερθούν στην επόμενη ενότητα.

3.2.3.2 Συναρτήσεις υπολογισμού

Σε αυτή την ενότητα θα περιγραφεί ο τρόπος με τον οποίο υπολογίζεται η επόμενη προβλεφθείσα τοποθεσία του χρήστη. Το αρχείο **calculateDistance.py** υπολογίζει την απόσταση τόσο μεταξύ δύο χρηστών όσο και την επόμενη τοποθεσία. Αποτελείται από 6 συναρτήσεις, μία για τον υπολογισμό της προβλεφθείσας τοποθεσίας όταν ο χρήστης δεν έχει συνδεθεί στην εφαρμογή και άρα δεν ξέρουμε σε τι κατηγορία ανήκει (Άτομο, Αυτοκίνητο), μία για τον υπολογισμό της επόμενης τοποθεσίας όταν ο χρήστης είναι της κατηγορίας «Αυτοκίνητο», υλοποιώντας το CYRA model based Trajectory Prediction από τις υλοποιήσεις που αναφέρθηκαν σε προηγούμενη ενότητα, μία για τον υπολογισμό της επόμενης τοποθεσίας του χρήστη που αποτελεί «Άτομο» με την εκτίμηση της απόστασης βηματισμού του χρήστη ως μέσο όρο απόστασης βήματος,μία για την μέτρηση της απόστασης μεταξύ δύο χρηστών, οι οποίοι αποτελούν παραμέτρους της συνάρτησης και τέλος μία που υπολογίζει το βάρος των δευτερολέπτων για τα οποία είναι αποδεκτό να επιτρέψουμε τον υπολογισμό της απόστασης των χρηστών στη λίστα του **dataHandle.py**. Οι υπολογισμοί θεωρούνται ορθοί πιθανότερα μετά την πρόβλεψη της πρώτης τοποθεσίας και έπειτα, ενώ τα δεδομένα που λαμβάνει ο χρήστης από την συσκευή του είναι πιο σταθερά όταν είναι σε ανοιχτούς χώρους και όχι σε εσωτερικούς (σπίτι). Η πρώτη συνάρτηση είναι η εξής:

```
1.  #calculate next location(lat,lon) of user
2.  def calculateNextLocation(dict):
3.      category = dict.get('category')
4.      response = 'null'
5.      if category == 'null':
6.          print "General category"
7.          response = calculateDistanceGeneral(dict)
8.      elif category == 'car':
9.          print "Car category"
10.         response = calculateDistanceCar(dict)
11.     else:
12.         print "Person category"
13.         response = calculateDistancePerson(dict)
14.     return response
```

Πιο συγκεκριμένα, εκτελεί έναν από τους τρείς τρόπους υπολογισμού της επόμενης θέσης ανάλογα με την κατηγορία στην οποία ανήκει ο χρήστης. Τα δεδομένα που λαμβάνει και στέλνει είναι διαφορετικά για κάθε κατηγορία. Η δεύτερη συνάρτηση είναι η εξής:

```
#calculate next location if user has not logged in and therefore we do not know in wh
ick category he belongs to
def calculateDistanceGeneral(dict):
    #get data info
    latitude = dict.get('latitude')
    longitude = dict.get('longitude')
    direction = dict.get('direction')
    velocity = dict.get('velocity')
    id = dict.get('id')
    search = dict.get('search')
    category = dict.get('category')
    timestamp = dict.get('timestamp')
    distance = velocity * TIME_SEC
    lat2 = asin(sin(radians(latitude)) * cos(distance / R) + cos(radians(latitude)) * sin(di
stance / R) * cos(radians(direction)))
    lon2 = radians(longitude) + atan2(sin(radians(direction)) * sin(distance / R) * cos(r
adians(latitude)), cos(distance / R) - sin(radians(latitude)) * sin(lat2))
    lat2 = degrees(lat2)
    lon2 = degrees(lon2)
    return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': sear
ch, 'category': category, 'timestamp': timestamp, 'velocity': velocity, 'lat_prev': latitude
, 'lon_prev': longitude}
```

Λαμβάνει ως δεδομένα από τον χρήστη την τοποθεσία, την ταχύτητα, την κατεύθυνση, το id, την ημερομηνία λήψης δεδομένων, τον τύπο αναζήτησης και την κατηγορία στην οποία ανήκει. Μέσω αυτών στέλνει πίσω στον χρήστη την νέα τιμή με ‘purpose’:’calculated’ και το ίδιο id, για να ξεχωρίζει από τα μηνύματα που λαμβάνει ο χρήστης από τον Server, απλά στέλνοντας στο θέμα(topic) με κατάληξη το συγκεκριμένο id (StreetAdvisor/handleInfo/id). Αρχικά, υπολογίζει την απόσταση που θα διανύσει ο χρήστης σε χρόνο τριών δευτερολέπτων και με βάση την ταχύτητά του. Έπειτα, με τις τριγωνομετρικές συναρτήσεις που ακολουθούν στο κομμάτι του κώδικα, βρίσκει για την συγκεκριμένη απόσταση και κατεύθυνση το νέο γεωγραφικό μήκος και πλάτος. Για τον υπολογισμό της επόμενης τοποθεσίας με βάση την κατηγορία, έχουν υλοποιηθεί δύο περαιτέρω συναρτήσεις, μία για την κατηγορία του «Ατόμου» και άλλη μία για την κατηγορία του «Αυτοκινήτου». Οι δύο αυτές συναρτήσεις, υλοποιούν εν μέρη κάποιες προτεινόμενες μεθόδους, μέσω ερευνητικών πληροφοριών (Papers). Όσον αφορά την υλοποίηση σε σχέση με την κατηγορία του «Αυτοκινήτου», εφαρμόστηκε εν μέρη. Πιο συγκεκριμένα, η επόμενη τοποθεσία εκτιμάται με την βοήθεια του CYRA model, διότι η συσκευή μας μπορεί να λάβει τα απαραίτητα δεδομένα προς επεξεργασία μόνο για το συγκεκριμένο μοντέλο ιχνηλάτησης και όχι για το Maneuver Recognition Model, που σχετίζεται με το προφίλ κίνησης του οχήματος. Παρακάτω φαίνονται οι τύποι που εφαρμόστηκαν για την συγκεκριμένη υλοποίηση:

B. Trajectory prediction with motion model (T_{mdl})

Assuming CYRA motion model, the components of the velocity along each dimension in the Cartesian frame are

$$\begin{cases} v_x(t) = & v(t) \cdot cos(\omega_0 \cdot t + \theta_0) \\ v_y(t) = & v(t) \cdot sin(\omega_0 \cdot t + \theta_0) \end{cases} \quad (18)$$

where $v(t) = a_0 \cdot t + v_0$. T_{mdl} is obtained on a closed form by integrating the velocity.

$$T_{mdl} = \begin{cases} x(t) = & \frac{a_0}{\omega_0^2} cos(\theta(t)) + \frac{v(t)}{\omega_0} sin(\theta(t)) + c_x \\ y(t) = & \frac{a_0}{\omega_0^2} sin(\theta(t)) - \frac{v(t)}{\omega_0} cos(\theta(t)) + c_y \end{cases} \quad (19)$$

Εικόνα 18 : Τύποι για το CYRA Model 1

where c_x and c_y are constants fixed with the initial values.

$$\begin{cases} c_x = & x_0 - \frac{v_0}{\omega} sin(\theta_0) - \frac{a_0}{\omega_0^2} cos(\theta_0) \\ c_y = & y_0 + \frac{v_0}{\omega_0} cos(\theta_0) - \frac{a_0}{\omega_0^2} sin(\theta_0) \end{cases} \quad (20)$$

If $\omega_0 = 0$, the predicted trajectory is rectilinear and given by

$$T_{mdl} = \begin{cases} x(t) = & \left(\frac{1}{2} \cdot a_0 \cdot t^2 + v_0\right) cos(\theta_0) + x_0 \\ y(t) = & \left(\frac{1}{2} \cdot a_0 \cdot t^2 + v_0\right) sin(\theta_0) + y_0 \end{cases} \quad (21)$$

Εικόνα 19 : Τύποι για το CYRA Model 2

Λαμβάνει ως πληροφορίες το id, την τοποθεσία, την ταχύτητα, την κατεύθυνση και ταχύτητα, την επιτάχυνση αλλά και την ημερομηνία, την κατηγορία του και το είδος αναζήτησης. Όπως και πριν, επιστρέφει ‘purpose’:’calculated’ και τις ίδιες πληροφορίες με την προηγούμενη, γενική, υλοποίηση.

```
1.  #calculate next location if a user drives a car
2.  def calculateDistanceCar(dict):
3.      #get data info
4.      id = dict.get('id')
5.      latitude = dict.get('latitude')
6.      longitude = dict.get('longitude')
7.      acceleration = dict.get('acceleration')
8.      velocity = dict.get('velocity')
9.      direction = dict.get('direction')
10.     yawRate = dict.get('yaw_rate')
11.     search = dict.get('search')
12.     category = dict.get('category')
```

```
13.    timestamp = dict.get('timestamp')
14.    #Tmdl ~ next position
15.    if(yawRate == 0):
16.        print("Yaw Rate is 0")
17.        distance = (((1/2) * acceleration * pow(TIME_SEC,2)) + (velocity * TIME_SEC))

18.        lat2 = asin(sin(radians(latitude)) * cos(distance / R) + cos(radians(latitude)) * sin
    (distance / R) * cos(radians(direction)))
19.        lon2 = radians(longitude) + atan2(sin(radians(direction)) * sin(distance / R) * co
    s(radians(latitude)), cos(distance / R) - sin(radians(latitude)) * sin(lat2))
20.        lat2 = degrees(lat2)
21.        lon2 = degrees(lon2)
22.    else:
23.        # Cx
24.        distance1 = - ((velocity/yawRate) * sin(radians(direction))) -
     ((acceleration / pow(yawRate,2)) * cos(radians(direction)))
25.        Cx = latitude + (180/PI)*(distance1/R)
26.        # Cy
27.        distance2 = ((velocity/yawRate) * cos(radians(direction))) -
     ((acceleration / pow(yawRate,2)) * sin(radians(direction)))
28.        Cy = longitude + (180/PI)*(distance2/R)/cos(latitude)
29.        # Ut
30.        Ut = acceleration * TIME_SEC + velocity
31.        # yawAngleT
32.        yawAngleT = yawRate * TIME_SEC
33.        distance1 = ((acceleration/pow(yawRate,2)) * cos(radians(yawAngleT))) + ((Ut/
    yawRate) * sin(radians(yawAngleT)))
34.        distance2 = ((acceleration/pow(yawRate,2)) * sin(radians(yawAngleT))) -
     ((Ut/yawRate) * cos(radians(yawAngleT)))
35.        lat2 = Cx + (180/PI)*(distance1/R)
36.        lon2 = Cy + (180/PI)*(distance2/R)/cos(Cx)
37.    return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': sear
    ch, 'category': category, 'timestamp': timestamp, 'velocity': velocity, 'lat_prev': latitude
    , 'lon_prev': longitude}
```

Για την υλοποίηση του βηματισμού για έναν χρήστη που κατηγοριοποιείται ως «Άτομο», χρησιμοποιήθηκε σε μεγάλη έκταση ο αλγόριθμος ανίχνευσης βημάτων, δηλαδή ο υπολογισμός της διάστασης που θα διανύσει ο χρήστης με έναν μέσο όρο μήκους βηματισμού. Το συγκεκριμένο Paper αναφέρει τα παρακάτω ως τύπους:

$$X_{(t)} = X_{(t-1)} + step.\cos(azimuth_{(t)})$$
$$Y_{(t)} = Y_{(t-1)} - step.\sin(azimuth_{(t)}) \quad (1)$$

where

$X_{(t)}$, $Y_{(t)}$ - current position,
$X_{(t-1)}$, $Y_{(t-1)}$ - previous position,
step - average step length,
azimuth - azimuth of the steps divided to four main directions (0°, 90°, 180°, 270°).

Εικόνα 20 : Τύποι για επόμενη θέση με βάση τον βηματισμό

Δεν χρησιμοποιήθηκε το Adaptive Step Length που αναφέρεται ως πιο ακριβής τρόπος, για τον λόγο ότι πρέπει η κάθε μεταβλητή α,β,γ να τροποποιηθεί αρχικά για κάθε νέο χρήστη. Αυτό δεν το προτιμούμε, διότι η συγκεκριμένη εφαρμογή επιθυμούμε να είναι γενική και να μην χρειάζεται ειδική μεταχείριση για κάθε χρήστη. Η συνάρτηση είναι η εξής:

```
1.  #user category is a person, calculate next location with step length for 3 steps after
2.  def calculateDistancePerson(dict):
3.      #get data info
4.      id = dict.get('id')
5.      latitude = dict.get('latitude')
6.      longitude = dict.get('longitude')
7.      direction = dict.get('direction')
8.      stepLength = dict.get('step_length')
9.      stepCounter = dict.get('step_counter')
10.     search = dict.get('search')
11.     category = dict.get('category')
12.     timestamp = dict.get('timestamp')
13.     distance = stepCounter * stepLength
14.     lat2 = asin(sin(radians(latitude)) * cos(distance / R) + cos(radians(latitude)) * sin(di
    stance / R) * cos(radians(direction)))
15.     lon2 = radians(longitude) + atan2(sin(radians(direction)) * sin(distance / R) * cos(r
    adians(latitude)), cos(distance / R) - sin(radians(latitude)) * sin(lat2))
16.     lat2 = degrees(lat2)
17.     lon2 = degrees(lon2)
18.     return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': sear
    ch, 'category': category, 'timestamp': timestamp, 'step_length': stepLength, 'step_cou
    nter': stepCounter, 'lat_prev': latitude, 'lon_prev': longitude}
```

Γενικά, οι παραπάνω τύποι χρησιμοποιούνται για να βρεθεί η τοποθεσία του χρήστη μετά από ένα μόνο βήμα. Στην περίπτωσή μας, θα ήταν πολύ επιβαρυντικό, να ανταλλάσσει το σύστημά μας πληροφορίες σε κάθε βήμα. Για τον λόγο αυτό, απλά δείχνουμε τον τρόπο που θα μπορούσαμε να λάβουμε τον αριθμό των βημάτων και το

μήκος βηματισμού, αλλά σαν απόσταση στις ήδη υπάρχουσες συντεταγμένες, προσθέτω την απόσταση που διένυσε σε προηγούμενο στάδιο ο χρήστης. Η επόμενη συνάρτηση υπολογίζει την απόσταση μεταξύ δύο χρηστών με την βοήθεια μόνο των γεωγραφικών πλατών και μηκών τους. Τα προαναφερθέντα δεδομένα μετατρέπονται σε μοίρες και μέσω τριγωνομετρικών εξισώσεων, λαμβάνεται η απόσταση σε μέτρα. Αυτή η συνάρτηση αποτελεί τον υπολογισμό της επόμενης τοποθεσίας όταν δεν έχει καθοριστεί η κατηγορία στην οποία ανήκει ο χρήστης και άρα αποτελεί μία γενική υλοποίηση πρόβλεψης θέσης. Ο παραπάνω υπολογισμός εκτελείται μόνο αν η τρίτη συνάρτηση που καλείται στην συγκεκριμένη, επιστρέψει αληθές αποτέλεσμα.

```
1.  #calculate the distance between 2 users(with lat,lon)
2.  def calculateDistanceUsers(user1, user2):
3.      if(calculateWeightTime(user1,user2)):
4.          lat1 = radians(user1.get('latitude'))
5.          lon1 = radians(user1.get('longitude'))
6.          lat2 = radians(user2.get('latitude'))
7.          lon2 = radians(user2.get('longitude'))
8.          dlon = lon2 - lon1
9.          dlat = lat2 - lat1
10.         a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
11.         c = 2 * atan2(sqrt(a), sqrt(1 - a))
12.         distance = R * c
13.         return distance
14.     else:
15.         return MAX_DISTANCE
```

Η τελευταία συνάρτηση είναι η εξής:

```
1.  #calculate weight of secs between 2 users and return if it is acceptable
2.  def calculateWeightTime(user1,user2):
3.      timestampUser1 = int(user1.get('timestamp'))/1000
4.      timestampUser2 = int(user2.get('timestamp'))/1000
5.      datetime1 = datetime.fromtimestamp(timestampUser1)
6.      datetime2 = datetime.fromtimestamp(timestampUser2)
7.      #get seconds of last location calculated
8.      if(datetime1 > datetime2):
9.          diff = (datetime1-datetime2).total_seconds()
10.     else:
11.         diff = (datetime2-datetime1).total_seconds()
12.     #get weight depending on speed or speed length of user2
13.     velocity = -1
```

```
14.    stepLength = -1
15.    speed = 0   #use velocity
16.    if 'velocity' in user2:
17.        velocity = user2.get('velocity')
18.    else:
19.        stepLength = user2.get('step_length')
20.        speed = 1
21.    if(speed == 0):
22.        if(velocity < 1 and velocity >= 0):
23.            if(diff <= MAX_SECS):
24.                return 1
25.            else:
26.                return 0
27.        weight = diff / velocity
28.        if(weight <= MAX_SECS):
29.            return 1
30.        else:
31.            return 0
32.    else:
33.        if(stepLength == 0):
34.            if(diff <= MAX_SECS_PERSON):
35.                return 1
36.            else:
37.                return 0
38.        return 1
```

Λαμβάνει από τους δύο χρήστες που επιθυμούμε να υπολογίσουμε την απόστασή τους, τον χρόνο που έλαβαν τα δεδομένα από τα κινητά τους, δηλαδή πότε έγινε η τελευταία ενημέρωση της τοποθεσίας τους. Ο χρόνος μετριέται σε milliseconds και βρίσκουμε την διαφορά τους σε δευτερόλεπτα. Στην συνέχεια, και με βάση την ταχύτητα του δεύτερου χρήστη (δηλαδή του χρήστη της τοπικής λίστας) ή αν ανήκει στην κατηγορία «Άτομο», την απόσταση που διανύει στον βηματισμό του, υπολογίζεται το βάρος ως διαφορά / ταχύτητα και αν είναι μικρότερο που μέγιστου ορίου βάρους, επιστρέφει αληθές.

4. ΑΞΙΟΛΟΓΗΣΗ ΥΛΟΠΟΙΗΣΗΣ

4.1 Πειράματα

Πριν την αφετηρία των πειραμάτων, έγινε μία εκκαθάριση των δεδομένων κάθε χρήστη στην βάση Firebase, αλλά και δημιουργήθηκαν νέοι λογαριασμοί, έτσι ώστε τα στατιστικά αλλά και γενικότερα τα δεδομένα, να είναι ενημερωμένα με τις πρόσφατες αλλαγές στο Contract των δομών.

Ο τρόπος διεξαγωγής ενός ορθού πειράματος, μπορεί να γίνει απλά με την χρήση 1 ή 2 κινητών με λειτουργικό Android και τον υπολογιστή που θα λειτουργήσει ως Server για τον υπολογισμό των νέων τιμών και την επιστροφή τους στους users. Επίσης, είναι επιθυμητό, αν η σύνδεση στο Internet και των δύο πλευρών είναι ισχυρή, για να μην υπάρξουν αρκετές αποκλίσεις. Για την χρήση ενός μόνο user, τα βασικά βήματα που πρέπει να ακολουθηθούν είναι τα εξής:

1) Άνοιγμα εφαρμογής Street Advisor, πατώντας το κατάλληλο εικονίδιο στο UI του κινητού.
2) Αν εμφανιστεί παράθυρο (prompt) για αποδοχή δικαιωμάτων (permissions), τότε ο χρήστης πρέπει να πατήσει «ΝΑ ΕΠΙΤΡΑΠΕΙ».
3) Η ιχνηλάτηση (tracking) ξεκινά, όταν ο χρήστης πατήσει το κεντρικό κουμπί (button) «play».
4) Αν εμφανιστεί prompt για «Πλήρης Λειτουργικότητα», ο χρήστης έχει δύο επιλογές:
   - Να πατήσει «ΣΥΝΕΧΕΙΑ» και να αγνοήσει τις επιπλέον προσφερόμενες λειτουργικότητες όπως, υπολογισμό στατιστικών χρήστη και απεικόνισή τους μέσω γραφημάτων (pie chart, bar charts) ανάλογα με τον τρόπο χρήσης της εφαρμογής, δηλαδή, ώρες ανά εβδομάδα, ώρες ανά μήνα και ώρες ανά τοπολογία, στις οποίες επιλέγονται οι top 5 συχνότερες περιοχές με τις περισσότερες ώρες λειτουργίας.
   - Να πατήσει «ΣΥΝΔΕΣΗ» και να έχει όλες τις λειτουργικότητες που αναφέρθηκαν.
5) Με το πάτημα του «play button», ξεκινά η ιχνηλάτηση της τοποθεσίας του χρήστη.
6) Ο χρήστης μπορεί να πατήσει το κουμπί «ΕΜΦΑΝΙΣΗ ΧΑΡΤΗ», το οποίο θα του δείξει σε ποιό σημείο βρίσκεται και ποιό είναι το επόμενο σημείο που έχει υπολογιστεί από τον Server.

Αυτά ήταν τα βασικά βήματα από την μεριά του χρήστη. Από την πλευρά του υπολογιστή-εξυπηρετητή(Server) έχουμε τα εξής:

1) Πριν την εκκίνηση της εφαρμογής από τον χρήστη, πρέπει να ξεκινήσει την λειτουργία του ο Server.
2) Ο Server αρχίζει να λειτουργεί, με την εκτέλεση του αρχείου «handlelnfo.py» από την κονσόλα εντολών (command line - CMD), πληκτρολογώντας απλώς "handlelnfo.py".
3) Ο Server συνδέεται με το θέμα (topic) που έχει οριστεί, στην περίπτωσή μας, «Street Advisor/handlelnfo/+».

Τα δεδομένα που μεταφέρονται μεταξύ των Server - client είναι σε μορφή JSON, οπότε και η μετάφρασή της είναι πολύ εύκολη. (παράρτημα για JSON).

Αν ο Server λάβει πληροφορίες, πρέπει να τις αποκωδικοποιήσει και μέσω αυτών των αποτελεσμάτων, να εκτελέσει την κατάλληλη λειτουργία (υπολογισμός επόμενης θέσης, διαγραφή χρήστη από τις τοπικές λίστες κ.ο.κ).

Στόχος του πειράματος είναι να υπολογιστεί σωστά η επόμενη προβλεπόμενη θέση του χρήστη. Αυτό μπορεί να εξεταστεί, είτε με την χρήση logs, στα οποία θα αποτυπώνονται οι πληροφορίες που διακινούνται μεταξύ των πλευρών, είτε από τον χρήστη, με το πάτημα για την «ΕΜΦΑΝΙΣΗ ΧΑΡΤΗ» και τον έλεγχο του επόμενου σημείου πάνω στον χάρτη. Αν το σημείο είναι στην ίδια κατεύθυνση με αυτή της κίνησης του χρήστη και τόσο μακριά ανάλογα με την ταχύτητά του, τότε θεωρείται ορθή.

Επίσης, με την κίνηση δύο ή παραπάνω χρηστών στην ίδια περιοχή, πρέπει να εξεταστεί αν εμφανίζεται ειδοποίηση στον χρήστη (ή σε όλους) ότι υπάρχει πιθανότητα διέλευσης από την ίδια τοποθεσία την ίδια χρονική στιγμή σε ακτίνα 15 μέτρων απόστασης. Για τον λόγο αυτόν, χρειάζονται τουλάχιστον 2 χρήστες για την διεξαγωγή του συγκεκριμένου πειράματος. Η μόνη ενέργεια που πρέπει να πραγματοποιήσουν είναι να έχουν πατήσει το «play button» και να αρχίσουν να κινούνται προς το ίδιο σημείο. Επειδή η εφαρμογή προβλέπει την επόμενη θέση του χρήστη που θα είναι μετά από 3 δευτερόλεπτα (seconds), η ειδοποίηση στις συσκευές των χρηστών πρέπει να εμφανιστεί σχεδόν πριν από 3 δεύτερα. Η ειδοποίηση έχει την εξής μορφή:

- Τίτλος: «Προσοχή!».
- Περιεχόμενο: «Μία ή περισσότερες περιστάσεις μπορεί να συμβούν προς τα εκεί που πηγαίνετε».

Προηγουμένως αναφέρθηκε πως οι πληροφορίες που μεταφέρονται μεταξύ Server-Client είναι σε μορφή JSON. Ο Server λαμβάνει τρία διαφορετικά είδη JSON, ανάλογα την κατηγορία στην οποία ανήκει ο χρήστης. Αυτά είναι της μορφής:

Για την «Γενική» περίπτωση αλλά και την περίπτωση του «Αυτοκινήτου»:

```
1. return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': search
   , 'category': category, 'timestamp': timestamp, 'velocity': velocity, 'lat_prev': latitude, 'l
   on_prev': longitude}
```

Για την περίπτωση του «Ατόμου»:

```
1. return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': search
   , 'category': category, 'timestamp': timestamp, 'step_length': stepLength, 'step_count
   er': stepCounter, 'lat_prev': latitude, 'lon_prev': longitude}
```

το οποίο, ως κύριες πληροφορίες έχει, τις συντεταγμένες του χρήστη την χρονική στιγμή που χρησιμοποιεί την εφαρμογή, την ταχύτητα ή το μήκος βηματισμού και τον αριθμό των βημάτων, την κατεύθυνση κίνησής του αλλά και την πλήρης ημερομηνία που διέλευσε από το συγκεκριμένο σημείο. Μέσω αυτών, στέλνει στον συγκεκριμένο χρήστη, ο οποίος ξεχωρίζει από τους άλλους από τη μοναδική συμβολοσειρά id που του έχει δοθεί κατά την εκκίνηση της λειτουργικότητας της εφαρμογής, τις υπολογισμένες συντεταγμένες. Έπειτα, με τις συγκεκριμένες συντεταγμένες, υπολογίζει την μικρότερη δυνατή απόσταση μεταξύ των άλλων χρηστών και του ξαναστέλνει το αποτέλεσμα. Άλλη πληροφορία που είναι εξίσου σημαντική για τον υπολογισμό τιμών, είναι στο ποιά κατηγορία ανήκει ο χρήστης (Άτομο, Αυτοκίνητο) και από ποιές κατηγορίες θέλει να λαμβάνει ειδοποιήσεις (όλες, Άτομο, Αυτοκίνητο, καμία). Τα φίλτρα αναζήτησης τροποποιούν τον υπολογισμό της μικρότερης απόστασης και άρα και την απορρέουσα τιμή της minDistance τιμής. Άρα, οι τιμές που τροποποιούνται συνεχώς είναι οι συντεταγμένες (πλάτος, μήκος), η ταχύτητα ή βηματισμός, η κατεύθυνση, αριθμός βημάτων, η γωνιακή ταχύτητα και η ημερομηνία του χρήστη. Τιμές οι οποίες τροποποιούνται ανάλογα με την ενέργεια του χρήστη στην εφαρμογή είναι η κατηγορία

που αποτελεί, η οποία μπορεί να επιλεχθεί όταν ο χρήστης είναι συνδεδεμένος ή είναι κενή (null), ποιούς θέλει να αναζητήσει στην γύρω περιοχή, και η μικρότερη απόσταση που προκύπτει από την κατηγορία και την αναζήτηση. Αξίζει να σημειωθεί πως η ημερομηνία υπάρχει, έτσι ώστε να απορριφθούν χρήστες που τυχόν δεν είναι αποδεκτό να συμπεριληφθούν στον υπολογισμό της μικρότερης απόστασης.

Όσον αφορά τα πειράματα μεταξύ δύο χρηστών, υπάρχουν τρία γενικά είδη ελέγχων που μπορούν να θεωρηθούν ως πειράματα. Το ένα αποτελεί την κίνηση δύο ανθρώπων με συγκεκριμένη ταχύτητα, προτιμούμε ίδια (αλλά δεν αποτελεί και τη σημαντικότερη, καθοριστική μεταβλητή). Το άλλο έχει σχέση με την σύγκρουση ενός αυτοκινήτου με έναν πεζό και το τελευταίο αποτελεί την δοκιμή της ειδοποίησης κατά την διάρκεια στροφής. Οι συσκευές που χρησιμοποιήθηκαν έχουν τις παρακάτω πληροφορίες υλικού/λογισμικού:

Πίνακας 4 : Συσκευές Πειραμάτων

	Συσκευή 1	**Συσκευή 2**
Μοντέλο	Sony Xperia Z3	Nubia NX 505J (Z7 Max)
Λειτουργικό Σύστημα	Android Marshmallow 6.0.1	Android Lollipop 5.1.1
Μνήμη	3 GB RAM	2 GB RAM
Επεξεργαστής	Quad-core 2.5 GHz Krait 400	Quad-core 2.5 GHz Krait 400
Μέγεθος Οθόνης	5.2 inches	5.5 inches
API Version	23	22

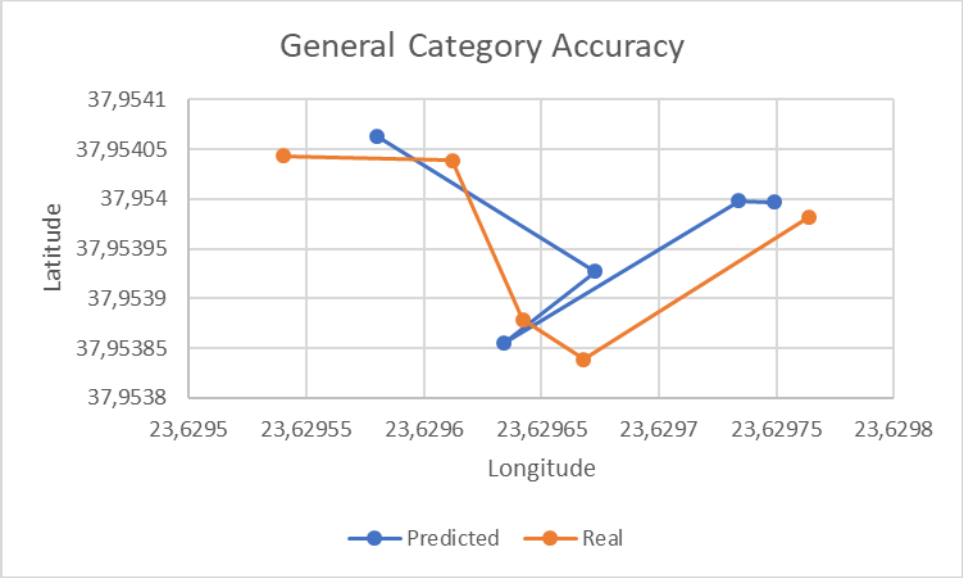
Σε κάθε πείραμα, για κάθε κατηγορία, ισχύει ότι ξεκινάνε να κινούνται από ένα σημείο, διασχίζοντας μία απόσταση το πολύ 50 μέτρων, μέχρι να αποφανθεί αν τελικά θα ειδοποιηθούν για σύγκρουση. Επίσης ξεκινάνε με ταχύτητα 0 και από αντίθετα σημεία του χάρτη. Στο πρώτο πείραμα, ο χρήστης κινείται με ταχύτητα περίπου 2 μέτρων το δευτερόλεπτο (m/s) και ο δεύτερος με περίπου 1 m/s. Ο ένας είναι συνδεδεμένος ως «Άτομο» ενώ ο άλλος δεν έχει λογαριασμό, οπότε και ο υπολογισμός της επόμενης τοποθεσίας του, γίνεται με τον γενικό τρόπο που περιγράφεται στην ενότητα της υλοποίησης. Στο δεύτερο πείραμα που πραγματοποιήθηκε, το «Άτομο» τρέχει με ταχύτητα 2 m/s ενώ το αυτοκίνητο με 10 m/s. Στο τρίτο πείραμα, γίνεται δοκιμή της ειδοποίησης πάνω στην στροφή, με δύο χρήστες εγγραγραμμένους ως «Άτομα».

4.2 Αποτελέσματα

Η σύγκρουση για το πρώτο πείραμα, «Άτομο» - «Μη συνδεδεμένος χρήστης», ειδοποιεί το «Άτομο» 11 δευτερόλεπτα περίπου πριν την κανονική πρόσκρουση, στον «Μη συνδεδεμένο χρήστη» 8 δευτερόλεπτα νωρίτερα από την πραγματική. Στο δεύτερο πείραμα, δηλαδή στο πείραμα με το «Άτομο» και το «Αυτοκίνητο», η ειδοποίηση σε καθένα χρήστη πραγματοποιείται για το «Αυτοκίνητο» 7 δευτερόλεπτα από την κανονική. Για το «Άτομο», ανιχνεύεται 4 δευτερόλεπτα πριν την πραγματική

πρόσκρουση. Τέλος, για το πείραμα με την στροφή, η πρώτη συσκευή ανίχνευσε την σύγκρουση 6 δευτερόλεπτα πριν συμβεί, ενώ η δεύτερη 0 δευτερόλεπτα περίπου.

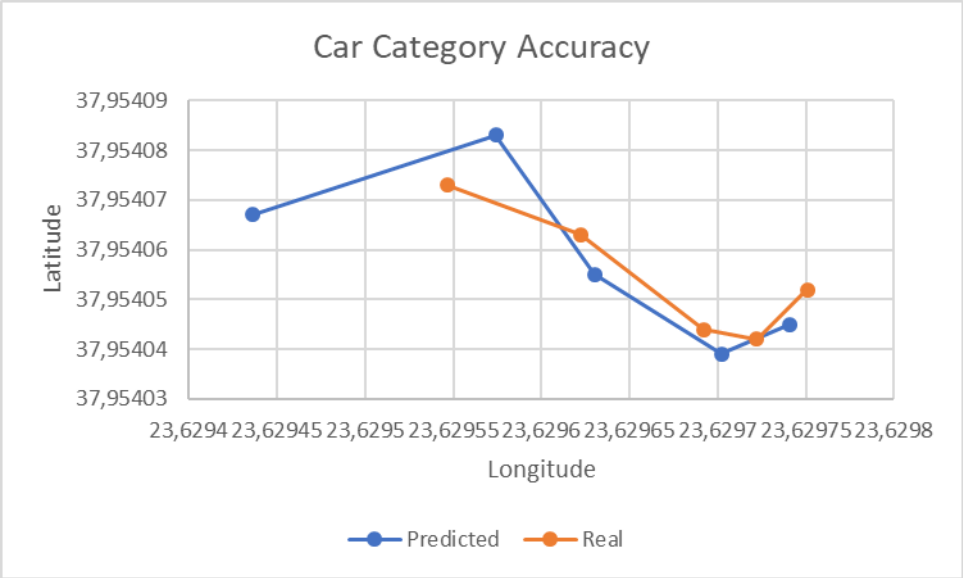
Παρακάτω φαίνονται διάφορα σχεδιαγράμματα που συγκρίνουν τις διάφορες υλοποιήσεις υπολογισμού της επόμενης τοποθεσίας, όσον αφορά την ακρίβεια των αποτελεσμάτων σε σχέση την πραγματική, για τα πρώτα 15 περίπου δευτερόλεπτα λειτουργίας:



Διάγραμμα 4 : Σχέση σημείων Χάρτη Προβλεπόμενου/Πραγματικού - Γενική Κατηγορία

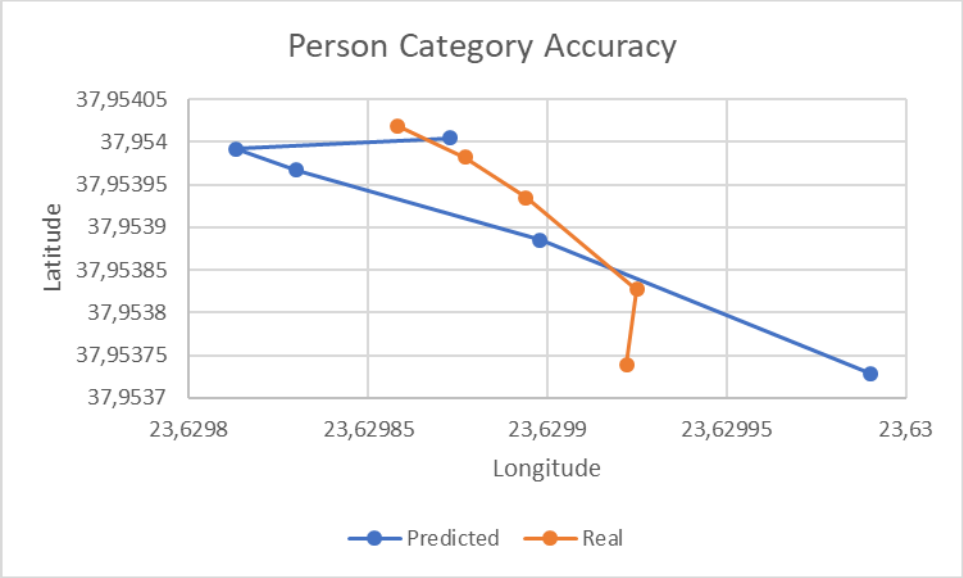
Μέσω του διαγράμματος, φαίνεται πως η αρχική τοποθεσία, αλλά και η πρόβλεψη στις στροφές, είναι λιγότερο ακριβής, απ'ότι στην ευθεία.

Για την κατηγορία του αυτοκινήτου, βλέπουμε πως η εκτίμηση της επόμενης τοποθεσίας ξεκινάει μετά την πρώτη τοποθεσία, αλλά είναι ιδιαίτερα ακριβής.



Διάγραμμα 5 : Σχέση σημείων Χάρτη Προβλεπόμενου/Πραγματικού - Κατηγορία Αυτοκινήτου

Για την κατηγορία του «Ατόμου», φαίνεται πως στις στροφές είναι δύσκολο να εκτιμήσει την επόμενη τοποθεσία, αλλά η απόκλιση είναι γενικά πολύ μικρή. Επίσης, η ορθή εκτίμηση της τοποθεσίας, ξεκινά μετά την πρώτη ανιχνευμένη τοποθεσία.



Διάγραμμα 6 : Σχέση σημείων Χάρτη Προβλεπόμενου/Πραγματικού - Κατηγορία Ατόμου

Μέσω των δοκιμών που έλαβαν μέρος, μπορούμε να καταγράψουμε στατιστικά των χρόνων σύνδεσης του Εξυπηρετητή αλλά και των χρηστών στο **test.mosquitto.org**, του χρόνου λήψης των αρχικών δεδομένων και των επόμενων από τους χρήστες, τον χρόνο υπολογισμού για κάθε κατηγορία από τον Εξυπηρετητή αλλά και τον χρόνο αποστολής των αποτελεσμάτων πίσω στον χρήστη. Η αποσύνδεση και η απεγγραφή από τα θέματα στα οποία ειναι εγγεγραμμένοι οι χρήστες, γίνεται σχεδόν ακαριαία. Οι χρόνοι υπολογίστηκαν για τον Εξυπηρετητή με την βοήθεια της συνάρτησης time.time() της γλώσσας Python [49] και του System.currentTimeMillis() για την γλώσσα JAVA. Οι χρόνοι εξαρτώνται και σε μεγάλο μέρος από την ταχύτητα της σύνδεσης. Παρακάτω φαίνονται αυτές οι πληροφορίες:

Πίνακας 5 : Χρόνοι απόκρισεις ανταλλαγής δεδομένων

	Εξυπηρετητής	Χρήστης
Χρόνος σύνδεσης και εγγραφής σε θέμα	0.50 – 0.60 δευτερόλεπτα περίπου	0.004 – 0.015 δευτερόλεπτα περίπου
Χρόνος μέχρι την πρώτη λήψη	0 – 10 δευτερόλεπτα περίπου	0 – 10 δευτερόλεπτα περίπου
Χρόνος επόμενων λήψεων	3 ή 6 δευτερόλεπτα περίπου	0 – 1.5 δευτερόλεπτα περίπου
Χρόνος αποστολής	0 – 1.5 δευτερόλεπτα περίπου	3 ή 6 δευτερόλεπτα περίπου

Πίνακας 6 : Χρόνος Υπολογισμού Επόμενης Τοποθεσίας

Χρόνος Υπολογισμού	
Γενικός	Περίπου 0 δευτερόλεπτα
Αυτοκίνητο	Περίπου 0 δευτερόλεπτα
Άτομο	Περίπου 0 δευτερόλεπτα

Τα αποτελέσματα που προκύπτουν από την αξιολόγηση είναι ποικίλα. Κατ' αρχάς, μέσω του πειράματος βλέπουμε ότι μπορεί να υπάρξουν βελτιώσεις ως προς τον υπολογισμό της επόμενης τοποθεσίας του χρήστη. Επίσης τα αποτελέσματα της λειτουργίας της εφαρμογής είναι η επόμενη τοποθεσία του χρήστη, η ειδοποίηση για τυχόν σύγκρουση και τα στατιστικά του χρήστη ανάλογα με την χρήση της εφαρμογής. Αυτά μπορούν να χρησιμοποιηθούν για να τροποποιηθεί η εφαρμογή σε επόμενο στάδιο. Όλες αυτές οι πληροφορίες ολοκληρώνουν το σενάριο πειράματος που αποδεικνύουν ότι η υλοποίηση είναι ορθή. Υπάρχουν ακόμη βελτιώσεις που μπορούν να πραγματοποιηθούν για να είναι η εφαρμογή περισσότερο βάσιμη.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

5.1 Γενικά Συμπεράσματα

Σκοπός της παρούσας πτυχιακής εργασίας είναι η ορθή και αποδοτική ιχνηλάτηση του χρήστη έτσι ώστε με την πρόβλεψη της επόμενης τοποθεσίας του, να αποφευχθεί τυχόν αναπάντεχο ατύχημα στην πορεία του. Για την ολοκλήρωση της εργασίας χρειάστηκαν αρκετές ώρες προετοιμασίας, αξιολόγησης και συνεχής επιδιόρθωσης σφαλμάτων.

Μέσα από την επίλυση της εφαρμογής, απορρέουν βασικά συμπεράσματα. Πρώτα απ' όλα, η πρόβλεψη της επόμενης τοποθεσίας του χρήστη είναι ένα κομμάτι το οποίο μπορεί να υλοποιηθεί με διαφορετικούς τρόπους. Ο υπολογισμός αυτός επηρεάζεται τόσο από την μηχανή επεξεργασίας των δεδομένων (υπολογιστής) όσο και από την ισχύ του διαδικτύου το οποίο χρησιμοποιεί ο χρήστης. Ακόμη, έχει άμεση σχέση και με τον τρόπο επικοινωνίας μεταξύ των δύο μερών (MQTT Broker), που σε πολλές περιπτώσεις μπορεί να δημιουργηθεί bottleneck αν η επεξεργασία δεν έχει μεγάλη αποδοτικότητα. Ακόμη, συμπεραίνουμε πως πολλά συστήματα κινητών είναι διαφορετικά από θέμα λειτουργικού, δυνατοτήτων και version οπότε η δημιουργία μίας τόσο πολύπλοκης εφαρμογής πρέπει να είναι συμβατή με την πλειοψηφία αυτών.

Μέσω της εφαρμογής, δόθηκε βάση στην χρήση της έννοιας IoT (Internet of Things), η οποία χρησιμοποιείται ευρύτατα και όλο και περισσότερο για την σύνδεση διαφορετικών συστημάτων και την εναλλαγή μηνυμάτων πάνω από συγκεκριμένο δίκτυο. Συγκεκριμένα, προωθείται έτσι ένας νέος τρόπος επίλυσης επικοινωνίας μεταξύ διαδιεργασιακών συστημάτων.

Τέλος, με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, γίνεται γνωστό πως η ανάγκη για μια όλο και πιο «έξυπνη» τεχνολογία ισχυροποιείται. Στη σήμερον ημέρα, η πρόβλεψη μπορεί και είναι δυνατό να σώσει ζωές, να καλυτερέψει την ποιότητα διαβίωσης. Η τεχνολογία είναι βασικός παράγοντας, που μέσω αυτού, πραγματοποιείται ο στόχος των ανθρώπων.

5.2 Παραδοχές – Περιορισμοί

Για την υλοποίηση της παρούσας πτυχιακής εργασίας, λήφθηκαν υπόψη κάποιες παραδοχές ή περιορισμοί. Αρχικά, πέρα από τους βασικούς τρόπους υλοποίησης που αναφέρθηκαν στο δεύτερο κεφάλαιο, επιλέξαμε να υλοποιήσουμε τον τρόπο πρόβλεψης της επόμενης θέσης με ένα συγκεκριμένο σύνολο τύπων, οιοποίοι αναφέρθηκαν στο κεφάλαιο υλοποίησης και αποτελούν την $4^{η}$ και $5^{η}$ υλοποίηση με αρκετές παραδοχές.

Ακόμη, η πρόβλεψη της επόμενης τοποθεσίας είναι 3 δεύτερα μακριά από την αρχική θέση του χρήστη. Η επόμενη θέση υπολογίζεται με βάση αυτό τον χρόνο αλλά και με την ταχύτητα του χρήστη εκείνη την στιγμή. Όπως προαναφέρθηκε, ο χρήστης στέλνει την τοποθεσία του κάθε 3 ή 6 δευτερόλεπτα για να υπάρξει συνέπεια με τους υπολογισμούς αλλά και τους πόρους της συσκευής. Για τον λόγο αυτό, ο χάρτης ανανεώνεται κάθε φορά που λαμβάνει ο χρήστης την νέα υπολογισμένη τιμή, οπότε και είναι εφικτή η παρατήρηση της αλλαγής της επόμενης τοποθεσίας χωρίς ο χρήστης να αλλάζει την οθόνη της εφαρμογής για να ελέγξει αν η εκτίμηση είναι σωστή. Αυτό υλοποιείται επιτυχώς έχοντας το MainService ως Bound Service [33], δηλαδή όλοι έχουν πρόσβαση σε μία δημόσια συνάρτησή του, για να λάβουν τα απαραίτητα δεδομένα.

Πέρα από τον LocationListener, η επόμενη τοποθεσία που υπολογίζει ο Εξυπηρετητής συσχετίζεται και με τα δεδομένα που χρειάζεται να λάβει η εφαρμογή από την συσκευή. Η συσκευή πρέπει να διαθέτει όλο το απαραίτητο Hardware, δηλαδή τους κατάλληλους ανιχνευτές/αισθητήρες, έτσι ώστε να μπορέσει ο Εξυπηρετητής να υπολογίσει την επόμενη τοποθεσία, για τις κατηγορίες «Άτομο» και «Αυτοκίνητο». Οι ανιχνευτές (Sensors), λαμβάνουν στο σύνολό τους πληροφορίες για την επιτάχυνση, το αζιμούθιο(κατεύθυνση) και γωνιακή ταχύτητα. Επίσης, χρησιμοποιείται ο ανιχνευτής για την ανίχνευση βημάτων, για τον υπολογισμό του μήκους του βηματισμού του χρήστη. Όλα αυτά μαζί, συνθέτουν ένα JSON το οποίο χρησιμοποιείται για την υλοποίηση των ερευνών, οι οποίες αναφέρθηκαν στην υλοποίηση των υπολογισμών στο μέρος του Εξυπηρετητή.

Η τοποθεσία που προβλέπεται είναι κατά πάσα πιθανότητα λανθασμένη, μόνο για την πρώτη εκτίμηση. Αυτό συμβαίνει επειδή, η συλλογή δεδομένων από την συσκευή του χρήστη προϋποθέτει την γνώση την πρώτης ανιχνευμένης τοποθεσίας για τον υπολογισμό της κατεύθυνσης ή της ταχύτητας, όταν αυτά δεν είναι διαθέσιμα από το Location αντικείμενο.

Η τοποθεσία ενός χρήστη χαρακτηρίζεται από τις συντεταγμένες, την ταχύτητα και την κατεύθυνση. Ως κατεύθυνση λαμβάνεται η κατεύθυνση του κινητού την ώρα κίνησης του χρήστη. Τα δεδομένα μεταφέρονται ως συμβολοσειρά JSON, η οποία μετατρέπεται σε dictionary για να γίνει η επεξεργασία. Η επιστροφή πληροφοριών επιτελείται με την ίδια μορφή.

Τέλος, ως τελευταίος περιορισμός λαμβάνεται υπόψη ο τρόπος αποστολής των μηνυμάτων μεταξύ του MQTT Broker. Το επίπεδο προστασίας είναι το 2 (QoS) το οποίο διατηρεί την συνοχή των μηνυμάτων, αφού θα μεταφερθούν και θα ληφθούν από τον εξυπηρετητή μόνο μία φορά, χωρίς να χαθεί πληροφορία.

Γενικότερα, η πλοήγηση μεταξύ των οθονών αλλά και η ενεργοποίηση των λειτουργιών του παρασκηνίου ελέγχθηκε τόσο μέσω των Logs αλλά και με την πλοήγηση σε κανονική συσκευή. Επιπρόσθετα, δημιουργήθηκαν ειδικά Tests για Debugging [45], τα οποία χωρίζονται σε instrumented tests / unit tests [46]. Τα πρώτα ειδικεύονται στην πλοήγηση του χρήστη στην συσκεύη και για αυτό τον λόγο εκτελούνται σε μια. Τα τελευταία, ελέγχουν συναρτήσεις και πιο πολύ υπολογισμούς και αποτελέσματα. Εκτελούνται στο IDE του Android Studio [47]. Για καλύτερη και ασφαλέστερη εκτίμηση του κώδικα αλλά και των προγραμματιστικών πρακτικών, χρησιμοποιήθηκαν εργαλεία όπως το SonarLint [48] αλλά και το built-in εργαλείο του Android Studio, για να έχει ο κώδικας όσο το δυνατόν λιγότερα προγραμματιστικά Bugs.

Γενικά κατά την διάρκεια των πειραμάτων, υπήρξαν πολλές παρεμβολές στην ανίχνευση της τοποθεσίας του χρήστη, καθιστώντας την δοκιμή πιο δύσκολη. Τελικά, τα πειράματα ολοκληρώθηκαν με επιτυχία.

5.3 Επόμενα Βήματα

Η συγκεκριμένη πτυχιακή εργασία είναι σε ικανό επίπεδο, ολοκληρώσιμη. Ωστόσο, υπάρχουν τρόποι περαιτέρω βελτίωσής της. Η αποδοτικότητα μεταξύ της εναλλαγής των μηνυμάτων μπορεί να αυξηθεί. Επιπρόσθετα, ο υπολογισμός της επόμενης τοποθεσίας μπορεί να εγκλειστεί από ένα άνω και κάτω φράγμα αλλά και να υπολογιστεί με μεγαλύτερη ακρίβεια, αν και εφόσον το σύστημα υπολογισμού είναι αρκετά ικανό να το πραγματοποιήσει.

Αυτή η εργασία είναι μία αρχή ως προς τον τομέα της πρόβλεψης. Αν συνεχιστεί, ίσως αποδειχθεί ωφέλιμη για την κοινωνία. Διαφορετικά πειράματα και τρόποι επίλυσης είναι

αυτά που μπορούν να διαφοροποιηθούν από την αρχική υλοποίηση που αναγράφεται στην παρούσα πτυχιακή. Υλοποιήσεις μπορούν να συνδυαστούν και να προσφέρουν το αποτέλεσμα που επιθυμούμε.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Activity	Δραστηριότητα
Advisor	Σύμβουλος
Assistant	Βοηθός
Authentication	Πιστοποίηση
Auto	Αυτόματο/Αυτοκίνητο
Backend	Πίσω Μέρος
Background	Παρασκήνιο
Bar chart	Ραβδόγραμμα
Base	Βάση
Based	Με βάση
Bottleneck	Μποτιλιάρισμα
Bound	Όριο, Περιορισμένος
Braking	Φρενάρισμα
Broker	Διαχειριστής
Bug	Προγραμματιστικό σφάλμα
Built-in	Ενσωματωμένος
Button	Κουμπί
Byte	Ψηφιόλεξη
Cellular	Κινητός
Client	Πελάτης
Command Line	Γραμμή εντολών
Constraint	Περιορισμός
Contract	Σύμβαση
Control	Έλεγχος
Correction	Διόρθωση
Count	Μέτρηση
Counter	Μετρητής
Database	Βάση Δεδομένων
Debugging	Αποσφαλμάτωση
Detection	Ανίχνευση
Direction	Κατεύθυνση
Distance	Απόσταση
Electronic	Ηλεκτρονικό

Emergency	Έκτακτη Ανάγκη
Extended	Επεκταμένο
Filter	Φίλτρο
Fragment	Θραύσμα
Hardware	Υλικό
Inches	Ίντσες
Instrumented	Με όργανα
Intelligent	Έξυπνος
Internet	Διαδίκτυο
Length	Μήκος
Location	Τοποθεσία
Log	Αρχείο Καταγραφής
Main	Κύριος
Manager	Διευθυντής
Maneuver	Ελιγμός
Map	Χάρτης
Matching	Ταίριασμα
Max	Μέγιστος
Millisecond	Μιλιδευτερόλεπτο
Model	Μοντέλο
Module	Μονάδα, τμήμα
Network	Δίκτυο
Object	Αντικείμενο
Payload	Φορτίο
Pedestrian	Πεζός
Permission	Άδεια
Pie chart	Διάγραμμα πίτας
Play	Παίζω
Prediction	Πρόβλεψη
Publish	Δημοσιεύω
Realtime	Πραγματικός Χρόνος
Recognition	Αναγνώριση
Second	Δευτερόλεπτο
Sensor	Ανιχνευτής
Server	Εξυπηρετητής

Service	Υπηρεσία
Space	Χώρος
Speed	Ταχύτητα
Stability	Σταθερότητα
State of the Art	Τεχνολογία Αιχμής
Station	Σταθμός
Step	Βήμα
Street	Δρόμος
String	Συμβολοσειρά
Subscribe	Εγγράφομαι
Test	δοκιμή
Thing	Πράγμα
Thread	Νήμα
Threshold	Κατώφλι
Topic	Θέμα
Trajectory	Τροχιά
Tunneling	Κατασκευάζω σήραγγα
Unit	Μονάδα
User	Χρήστης
Version	Έκδοση

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

AEB	Auto Emergency Braking
API	Application Programming Interface
CMD	Command line
CYRA	Constant Yaw Rate and Acceleration Model
ESC	Electronic Stability Control
GPS	Global Positioning System
HTTP	HyperText Tranfer Protocol
IBM	International Business Machines
IDE	Integrated Development Environment
IoT	Internet of Things
ISA	Intelligent Speed Assistant
JSON	JavaScript Object Notation
M2M	Machine-to-Machine
MANET	Mobile Ad hoc NETwork
MQTT	MQ Telemetry Transport
PDR	Pedestrian Dead Reckoning
REKF	Robust Extended Kalman Filter
RSSI	Received Signal Strength Indication
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/ Internet Protocol
TLS	Transport Layer Security
UI	User Interface
VAR	Vector AutoRegression
XML	eXtensible Markup Language
ΕΚΠΑ	Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΠΑΡΑΡΤΗΜΑ Ι

[1] Το JSON αντιπροσωπεύει το **J**ava**S**cript **O**bject **N**otation και αποτελεί σύνταξη για αποθήκευση και ανταλλαγή δεδομένων. Κατά την ανταλλαγή δεδομένων μεταξύ ενός προγράμματος περιήγησης και ενός διακομιστή, τα δεδομένα μπορούν να είναι μόνο σε μορφή κειμένου. Το JSON είναι κείμενο και μπορούμε να μετατρέψουμε οποιοδήποτε αντικείμενο JavaScript σε JSON και να στείλουμε αυτό το JSON στο διακομιστή. Μπορούμε, επίσης, να μετατρέψουμε κάθε JSON που λαμβάνεται από το διακομιστή σε αντικείμενα JavaScript. Με αυτόν τον τρόπο, μπορούμε να τροποποιήσουμε τα δεδομένα ως αντικείμενα JavaScript, χωρίς πολύπλοκες αναλύσεις και μεταφράσεις. Εάν έχετε αποθηκευμένα δεδομένα σε ένα αντικείμενο JavaScript, μπορείτε να μετατρέψετε το αντικείμενο σε JSON και να το στείλετε σε έναν διακομιστή. Το JSON είναι μια ελαφριά μορφή ανταλλαγής δεδομένων. Είναι "self-describing" και εύκολα κατανοητό. Το JSON είναι ανεξάρτητο από τη γλώσσα * από την οποία χρησιμοποιείται. Το κείμενο μπορεί να διαβαστεί και να χρησιμοποιηθεί ως μορφή δεδομένων από οποιαδήποτε γλώσσα προγραμματισμού. Η μορφή JSON καθορίστηκε αρχικά από τον Douglas Crockford. Ένα παράδειγμα είναι: {name: "John", age: 31, city: "New York"} [5].

[2] Η πλήρης υλοποίηση του Android Client όσον αφορά το backend κομμάτι, είναι η εξής:

- **MainService**:

```
package com.example.stefania.streetadvisor.Services;

import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.content.res.Configuration;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Location;
import android.os.Binder;
import android.os.Build;
import android.os.IBinder;
```

```
import android.provider.Settings;
import android.widget.Toast;

import com.example.stefania.streetadvisor.Activities.MainActivity;
import com.example.stefania.streetadvisor.Activities.SettingsActivity;
import com.example.stefania.streetadvisor.Classes.DataCar;
import com.example.stefania.streetadvisor.Classes.DataGeneral;
import com.example.stefania.streetadvisor.Classes.DataPerson;
import com.example.stefania.streetadvisor.Classes.SharedVariables;
import com.example.stefania.streetadvisor.MQTT.MqttHelper;
import com.example.stefania.streetadvisor.MQTT.Utils;
import com.example.stefania.streetadvisor.R;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationResult;
import com.google.android.gms.location.LocationServices;
import com.orhanobut.logger.Logger;

import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallbackExtended;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import androidx.core.app.ActivityCompat;
import androidx.core.app.NotificationCompat;
import androidx.core.app.NotificationManagerCompat;
import androidx.localbroadcastmanager.content.LocalBroadcastManager;
import androidx.preference.PreferenceManager;

```

```
import static androidx.core.app.NotificationCompat.VISIBILITY_PUBLIC;
import static com.example.stefania.streetadvisor.Classes.Constants.ACTION_LOCATION_BROADCAST_JOB;
import static com.example.stefania.streetadvisor.Classes.Constants.EXTRA_LATITUDE_JOB;
import static com.example.stefania.streetadvisor.Classes.Constants.EXTRA_LONGITUDE_JOB;
import static com.example.stefania.streetadvisor.MQTT.Utils.randomIdGenerator;

public class MainService extends Service implements SharedPreferences.OnSharedPreferenceChangeListener {

    private static final int LOCATION_INTERVAL = 6000;      // 6 seconds
    private static final int FASTEST_LOCATION_INTERVAL = 3000;  // 3 seconds

    private static final int SENSOR_DELAY_TIME = 3000000;    // 3 seconds
    private static final int NOTIFICATION_ID = 2;
    private static final String PREFERENCE_VIBRATION = "vibration";
    private static final String PREFERENCE_SOUND_EFFECT = "sound_effect";

    private static final String CHANNEL_ID = "1";
    private static final String JSON_PURPOSE = "purpose";
    private static final String JSON_PURPOSE_CAL = "calculated";
    private static final String JSON_LAT = "latitude";
    private static final String JSON_LAT_PREV = "lat_prev";
    private static final String JSON_LON = "longitude";
    private static final String JSON_LON_PREV = "lon_prev";
    private static final String JSON_MIN = "minDistance";
    private FusedLocationProviderClient mFusedLocationClient;
    private Map<String, ?> preferences;
    private NotificationCompat.Builder mBuilder;
    private NotificationManagerCompat notificationManager;
    private String latPrev;
    private String lnPrev;
    private String latNext;
    private String lnNext;
```

```
    private double minDistanceNum;
    // Sensors Variables
    private SensorManager mSensorManager;
    // Must - Have info
    // Acceleration without gravity
    private double accelerationNoGravivy;
    // Acceleration with gravity
    private double accelCurrent;
    private int yawRate;
    private int numberOfSteps;
    private Location lastLocation;
    private double stepLength;
    private String search;
    private boolean destroyed = false;
    // Binder given to clients
    private IBinder mBinder = new LocalBinder();

    public class LocalBinder extends Binder {
        public MainService getService() {
            // Return this instance of MainService so clients can call public methods
            return MainService.this;
        }
    }

    // Create LocationCallback
    private LocationCallback mLocationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            super.onLocationResult(locationResult);
            if (locationResult == null) {
                return;
            }
            Location location = locationResult.getLastLocation();
            // Calculate average step length
            if(lastLocation != null) {
```

```
                double distance = location.distanceTo(lastLocation);
                if(numberOfSteps == 0){
                    stepLength = 0;
                } else {
                    stepLength = distance / numberOfSteps;
                }
            } else{
                // This is the first time we set a location as last, no need for further calculations
                lastLocation = location;
                return;
            }
            double elapsedTime = (double)(location.getTime() - lastLocation.getTime()) / 1000; // Convert milliseconds to seconds
            double calculatedSpeed = lastLocation.distanceTo(location) / elapsedTime;
            double speed = location.hasSpeed() ? location.getSpeed() : calculatedSpeed;
            double calculatedBearing = lastLocation.bearingTo(location);
            double bearing = location.hasBearing() ? location.getBearing() : calculatedBearing;
            Logger.i("Location geocodes: lat: " + location.getLatitude() + ", lon: " + location.getLongitude());
            // For statistics
            sendToJobService(String.valueOf(location.getLatitude()), String.valueOf(location.getLongitude()));
            // For main function
            Logger.i("Speed: " + speed + ", bearing: " + bearing);
            location.setSpeed((float)speed);
            location.setBearing((float)bearing);
            Object data = createDataObject(location);
            if(data instanceof DataPerson){
                ((DataPerson) data).setNumberOfSteps(numberOfSteps);
            }
            // Calculate and return next location
            // MQTT send info to server
```

```
            ((SharedVariables) getApplication()).getMqttHelperHandleInfo().publishMessage(Utils.createJsonForData(getApplicationContext(), data, ((SharedVariables) getApplication()).getId(), search));
            if (minDistanceNum <= 15 && minDistanceNum != 0.0) {
                // Display notification
                alterNotification();
                notificationManager.notify(NOTIFICATION_ID, mBuilder.build());
            }
            lastLocation = location;
            numberOfSteps = 0;
        }
    };

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Initialize sensors
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        Sensor mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        boolean haveAccelerometer = mSensorManager.registerListener(mSensorEventListener, mAccelerometer, SENSOR_DELAY_TIME);
        Sensor mGyroscope = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
        boolean haveGyroscope = mSensorManager.registerListener(mSensorEventListener, mGyroscope, SENSOR_DELAY_TIME);
        Sensor mStepDetector = mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);
        boolean haveStepDetector = mSensorManager.registerListener(mSensorEventListener, mStepDetector, SENSOR_DELAY_TIME);

        if (!haveAccelerometer || !haveGyroscope || !haveStepDetector) {
            // Cannot get data for calculations
            Logger.e("haveAccelerometer: " + haveAccelerometer + " haveGyroscope: " + haveGyroscope + " haveStepDetector: " + haveStepDetector);
            mSensorManager.unregisterListener(mSensorEventListener);
            stopSelf();
            Toast.makeText(getApplicationContext(),R.string.service_error,Toast.LENGTH_LONG).show();
```

```
            return START_NOT_STICKY;
        }

        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        preferences = prefs.getAll();
        prefs.registerOnSharedPreferenceChangeListener(this);
        search = (String) preferences.get("notifications");
        Logger.i("MainService start category is: " + ((SharedVariables) this.getApplication()).getCategory());
        if(!destroyed && !((SharedVariables) this.getApplication()).isLoggedIn()) {
            ((SharedVariables) getApplication()).setId(randomIdGenerator());
        }
        // Create notification
        notificationManager = NotificationManagerCompat.from(getApplicationContext());
        createNotificationChannel();
        // Display notification
        mBuilder = createNotificationDefault();
        // Set Location Request
        LocationRequest mLocationRequest = LocationRequest.create();
        mLocationRequest.setInterval(LOCATION_INTERVAL);
        mLocationRequest.setFastestInterval(FASTEST_LOCATION_INTERVAL);
        mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

        Logger.i("Start Location Service");
        // Start Location Listener
        mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
        if (ActivityCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_COARSE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
            mFusedLocationClient.requestLocationUpdates(mLocationRequest, mLocationCallback, null);
        }
        numberOfSteps = 0;
```

```
        stepLength = 0;
        lastLocation = null;
        // MQTT
        ((SharedVariables) getApplication()).setMqttHelperHandleInfo(startMqtt());
        return START_STICKY;
    }

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

    private void sendToJobService(String lat, String lng) {
        Intent intent = new Intent(ACTION_LOCATION_BROADCAST_JOB);
        intent.putExtra(EXTRA_LATITUDE_JOB, lat);
        intent.putExtra(EXTRA_LONGITUDE_JOB, lng);
        LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
    }

    // Method for clients
    public List<Location> getStatesForMap() {
        Logger.i("Send states to clients: latPrev: " + latPrev + " ,lnPrev: " + lnPrev + "
,latnext: " + latNext + " ,lnNext: " + lnNext);
        if(latPrev == null || lnPrev == null || latNext == null || lnNext == null){
            Logger.w("Some of the geocodes are null.");
            return new ArrayList<>();
        }
        Location statePrev = new Location("");
        statePrev.setLatitude(Double.valueOf(latPrev));
        statePrev.setLongitude(Double.valueOf(lnPrev));
        Location stateNew = new Location("");
        stateNew.setLatitude(Double.valueOf(latNext));
        stateNew.setLongitude(Double.valueOf(lnNext));
        List<Location> mapInfo = new ArrayList<>();
        mapInfo.add(statePrev);
```

```
        mapInfo.add(stateNew);
        return mapInfo;
    }

    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String s) {
        sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);
        preferences = sharedPreferences.getAll();
        search = (String) preferences.get("notifications");
    }

    private NotificationCompat.Builder createNotificationDefault() {
        Intent intent = new Intent(this, MainActivity.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

        Intent settingsIntent = new Intent(this, SettingsActivity.class);
        settingsIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
        PendingIntent pendingSettingsIntent = PendingIntent.getActivity(this, 0, settingsIntent, 0);

        return new NotificationCompat.Builder(this, CHANNEL_ID)
            .setSmallIcon(R.drawable.notification_icon)
            .setContentTitle(getResources().getString(R.string.title_notification))
            .setContentText(getResources().getString(R.string.content_notification))
            .setStyle(new NotificationCompat.BigTextStyle().bigText(getResources().getString(R.string.content_notification)))
            .setPriority(NotificationCompat.PRIORITY_HIGH)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true)
            .setCategory(NotificationCompat.CATEGORY_ALARM)
            .setVisibility(VISIBILITY_PUBLIC)
```

```
            .addAction(R.drawable.settings_notification_icon, getString(R.string.settings_label), pendingSettingsIntent);
    }

    private void alterNotification() {
        Boolean vibration = (Boolean) preferences.get(PREFERENCE_VIBRATION);

        Boolean sound = (Boolean) preferences.get(PREFERENCE_SOUND_EFFECT);
        if(vibration == null || sound == null){
            return;
        }
        if (vibration) {
            mBuilder.setVibrate(new long[]{1000, 1000});
        }
        if (sound) {
            mBuilder.setSound(Settings.System.DEFAULT_NOTIFICATION_URI);
        }
    }

    private void createNotificationChannel() {
        // Create the NotificationChannel for API 26+
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            CharSequence name = getString(R.string.channel_name);
            String description = getString(R.string.channel_description);
            int importance = NotificationManager.IMPORTANCE_HIGH;
            NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name, importance);
            channel.setDescription(description);
            NotificationManager notManager = getSystemService(NotificationManager.class);
            notManager.createNotificationChannel(channel);
        }
    }

    @Override
    public void onDestroy() {
```

```
        super.onDestroy();
        Logger.i("Stop Location Service");
        PreferenceManager.getDefaultSharedPreferences(getApplicationContext()).unregisterOnSharedPreferenceChangeListener(this);
        if(mFusedLocationClient != null) {
            mFusedLocationClient.removeLocationUpdates(mLocationCallback);
        }
        // Remove user from mqtt
        if(((SharedVariables) getApplication()).getMqttHelperHandleInfo() != null) {
            ((SharedVariables) getApplication()).getMqttHelperHandleInfo().publishMessage(Utils.createJsonForUserDeletion(((SharedVariables) getApplication()).getId()));
            ((SharedVariables) getApplication()).getMqttHelperHandleInfo().unsubscribeFromTopic();
            ((SharedVariables) getApplication()).getMqttHelperHandleInfo().disconnectMQTTClient();
            ((SharedVariables) getApplication()).setMqttHelperHandleInfo(null);
        }
        if(mSensorManager != null) {
            mSensorManager.unregisterListener(mSensorEventListener);
        }
        destroyed = true;
    }

    private MqttHelper startMqtt() {
        // Connect to topic with suffix of the id of the user
        MqttHelper helper = new MqttHelper(getApplicationContext(), ((SharedVariables) getApplication()).getId());
        helper.setCallback(new MqttCallbackExtended() {
            @Override
            public void connectComplete(boolean b, String s) {
                Logger.i("Connection completed", s);
            }

            @Override
            public void connectionLost(Throwable throwable) {
                Logger.e(throwable,"Connection lost");
```

```
            }

            @Override
            public void messageArrived(String topic, MqttMessage mqttMessage) {
                try {
                    JSONObject responseJSON = new JSONObject(mqttMessage.toStrin
g());
                    Logger.i("Response from server: " + responseJSON.toString());
                    if (responseJSON.has(JSON_PURPOSE)) {
                        String purpose = responseJSON.getString(JSON_PURPOSE);
                        if (purpose.equals(JSON_PURPOSE_CAL)) {
                            latNext = responseJSON.getString(JSON_LAT);
                            lnNext = responseJSON.getString(JSON_LON);
                            latPrev = responseJSON.getString(JSON_LAT_PREV);
                            lnPrev = responseJSON.getString(JSON_LON_PREV);
                            ((SharedVariables) getApplication()).getChangedLocation().setL
atNew(latNext);
                            Logger.i("New location geocodes are: lat: " + latNext + " ,lon: " + l
nNext);
                        }
                    } else if (responseJSON.has(JSON_MIN)) {
                        String minDistance = responseJSON.getString(JSON_MIN);
                        if (minDistance != null) {
                            minDistanceNum = Float.valueOf(minDistance);
                        }
                    }
                } catch (JSONException e) {
                    Logger.e(e,"Unable to get JSON Object from Server.");
                }
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
                Logger.i("Delivery Completed");
            }
        });
```

```
        return helper;
    }

    private SensorEventListener mSensorEventListener = new SensorEventListener() {

        public void onAccuracyChanged(Sensor sensor, int accuracy) {}

        @Override
        public void onSensorChanged(SensorEvent event) {
            float x;
            float y;
            switch (event.sensor.getType()) {
                case Sensor.TYPE_ACCELEROMETER:
                    x = event.values[0];
                    y = event.values[1];
                    double z = event.values[2];
                    // Last acceleration including gravity
                    double mAccelLast = accelCurrent;
                    accelCurrent = Math.sqrt((x * x + y * y + z * z));
                    double delta = accelCurrent - mAccelLast;
                    // Perform low-cut filter
                    accelerationNoGravivy = accelerationNoGravivy * 0.9f + delta;
                    break;
                case Sensor.TYPE_GYROSCOPE:
                    yawRate = Math.round(event.values[1]);
                    break;
                case Sensor.TYPE_STEP_DETECTOR:
                    numberOfSteps++;
                    break;
                default:
                    break;
            }
        }
```

```
    };

    private Object createDataObject(Location location){
        String categoryStr = null;
        if (((SharedVariables) getApplication()).getCategory() != null) {
            Configuration configuration = Utils.getCurrentConfiguration(getApplication
Context());
            String catCmpCar = getApplicationContext().createConfigurationContext(c
onfiguration).getResources().getString(R.string.signup_category_car);
            String catCmpPerson = getApplicationContext().createConfigurationContex
t(configuration).getResources().getString(R.string.signup_category_person);
            if (((SharedVariables) getApplication()).getCategory().equals(catCmpCar))
{
                categoryStr = getApplicationContext().getResources().getString(R.string.
car);
            } else if (((SharedVariables) getApplication()).getCategory().equals(catCm
pPerson)) {
                categoryStr = getApplicationContext().getResources().getString(R.string.
person);
            } else {
                categoryStr = null;
            }
        }
        if(categoryStr == null){
            DataGeneral data = new DataGeneral();
            data.setLatitude(location.getLatitude());
            data.setLongitude(location.getLongitude());
            data.setVelocity(location.getSpeed());
            data.setDirection(location.getBearing());
            return data;
        }
        if(categoryStr.equals("car")){
            DataCar data = new DataCar();
            data.setLatitude(location.getLatitude());
            data.setLongitude(location.getLongitude());
            data.setAcceleration(accelerationNoGravivy);
            data.setVelocity(location.getSpeed());
```

```
            data.setDirection(location.getBearing());
            data.setYawRate(yawRate);
            return data;
        }
        if(categoryStr.equals("person")){
            DataPerson data = new DataPerson();
            data.setLatitude(location.getLatitude());
            data.setLongitude(location.getLongitude());
            data.setDirection(location.getBearing());
            data.setStepLength(stepLength);
            return data;
        }
        return null;
    }
}
```

- **MqttHelper**:

```
package com.example.stefania.streetadvisor.MQTT;

import android.content.Context;

import com.orhanobut.logger.Logger;

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.DisconnectedBufferOptions;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttCallbackExtended;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

import java.nio.charset.StandardCharsets;
```

```
public class MqttHelper {
    private MqttAndroidClient mqttAndroidClient;
    private static final String SERVER_URI_SSL_MOSQUITTO = "ssl://test.mosquitto.org:8883";
    private static final String SERVER_URI_SSL_ECLIPSE = "ssl://iot.eclipse.org:8883";
    private static final String SERVER_URI_TCP_ECLIPSE = "tcp://iot.eclipse.org:1883";
    private static final String SERVER_URI_TCP_MOSQUITTO = "tcp://test.mosquitto.org:1883";
    private static final String PREFIX_TOPIC = "StreetAdvisor/handleInfo/";
    private static final int BUFFER_SIZE = 500;
    private String topic;
    private Context mContext;

    // Create mqtt client
    public MqttHelper(Context context, String id) {
        mContext = context;
        mqttAndroidClient = new MqttAndroidClient(context, SERVER_URI_SSL_MOSQUITTO, MqttClient.generateClientId());
        mqttAndroidClient.setCallback(new MqttCallbackExtended() {
            @Override
            public void connectComplete(boolean b, String s) {
                Logger.i("Connection completed", s);
            }

            @Override
            public void connectionLost(Throwable throwable) {
                Logger.e(throwable,"Connection lost");
            }

            @Override
            public void messageArrived(String topic, MqttMessage mqttMessage) {
                Logger.i("Message Arrived", mqttMessage.toString());
            }

```

```
            @Override
            public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
                Logger.i("Delivery Completed");
            }
        });
        connect(id);
    }

    public void setCallback(MqttCallbackExtended callback) {
        mqttAndroidClient.setCallback(callback);
    }

    private void connect(String id) {
        MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
        mqttConnectOptions.setAutomaticReconnect(true);
        mqttConnectOptions.setCleanSession(true);
        if(mqttAndroidClient.getServerURI().contains("ssl")) {
            try {
                mqttConnectOptions.setSocketFactory(SSLUtils.createSocketFactory(mContext));
            } catch (Exception e) {
                Logger.e(e, "Set SSL error.");
            }
        }
        try {
            mqttAndroidClient.connect(mqttConnectOptions, null, new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    Logger.i("Connected successfully.");
                    DisconnectedBufferOptions disconnectedBufferOptions = new DisconnectedBufferOptions();
                    disconnectedBufferOptions.setBufferEnabled(true);
                    disconnectedBufferOptions.setBufferSize(BUFFER_SIZE);
                    disconnectedBufferOptions.setPersistBuffer(false);
                    disconnectedBufferOptions.setDeleteOldestMessages(true);
```

```
                mqttAndroidClient.setBufferOpts(disconnectedBufferOptions);
                subscribeToTopic(id);
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable except
ion) {
                Logger.e(exception,"Failed to connect to: " + mqttAndroidClient.getSe
rverURI());
            }
        });
    } catch (MqttException e) {
        Logger.e(e,"Exception connecting to topic.");
    }
 }


  public void subscribeToTopic(String id) {
    topic = PREFIX_TOPIC + id;
    Logger.i("Subscribe to topic: " + topic);
    try {
        mqttAndroidClient.subscribe(topic, 2, null, new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                Logger.i("Success subscribing");
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable except
ion) {
                Logger.e(exception,"onFailure: Subscribe failed!");
            }
        });
    } catch (MqttException e) {
        Logger.e(e,"subscribeToTopic: Exception in subscribing to topic.");
    }
```

```
    }

    public void publishMessage(String sendStr) {
        byte[] encodedPayload;
        try {
            encodedPayload = sendStr.getBytes(StandardCharsets.UTF_8);
            MqttMessage message = new MqttMessage(encodedPayload);
            Logger.i("Topic: " + topic + " ,message: " + message);
            if(mqttAndroidClient != null && mqttAndroidClient.isConnected()) {
                mqttAndroidClient.publish(topic, message);
            }
        } catch (MqttException e) {
            Logger.e(e,"publishMessage");
        }
    }

    public void disconnectMQTTClient(){
        Logger.i("Disconnecting client");
        try {
            if(mqttAndroidClient != null) {
                mqttAndroidClient.disconnect();
            }
        } catch (MqttException e) {
            Logger.e(e,"Cannot disconnect client");
        }
    }

    public void unsubscribeFromTopic(){
        Logger.i("Unsubscribing from topic: " + topic);
        try {
            if(mqttAndroidClient != null && mqttAndroidClient.isConnected()) {
                mqttAndroidClient.unsubscribe(topic);
            }
        } catch (MqttException e) {
            Logger.e(e,"Cannot unsubscribe from topic: " + topic);
```

```
        }
    }
}
```

[3] Η πλήρης υλοποίηση του Server είναι η εξής:

- **handleInfo.py:**

```
import paho.mqtt.client as mqtt
import ast
import ssl
import time
import sys
from calculateDistance import *
from dataHandle import *

#MQTT functions
def on_connect(mqttc, userdata, flags, rc):
    print('connected...rc=' + str(rc))

def on_disconnect(mqttc, userdata, rc):
    print('disconnected...rc=' + str(rc))

def on_message(mqttc, userdata, msg):
    print('message received...')
    #get dictionary
    dict = ast.literal_eval(str(msg.payload))
    print dict
    purpose = dict.get('purpose')
    #need to calculate next location of user
    if(purpose == 'calculate'):
        stepLength = 0.0
        velocity = 0.0
        if 'velocity' in dict:
            velocity = dict.get('velocity')
        if 'step_length' in dict:
            stepLength = dict.get('step_length')
```

```
        #calculate next position only if user is moving
        publishPath = 'StreetAdvisor/handleInfo/' + dict.get('id')
        if(velocity > 0.0 or stepLength > 0.0):
            print('Calculate next location...')
            nextLocation = calculateNextLocation(dict)
        else:
            if(dict.get('category') == 'person'):
                print("Default next location for person")
                nextLocation = {'purpose': 'calculated', 'id': dict.get('id'), 'latitude': dict.get('latitude'), 'longitude': dict.get('longitude'), 'search': dict.get('search'), 'category': dict.get('category'), 'timestamp': dict.get('timestamp'), 'step_length': dict.get('step_length'), 'step_counter': dict.get('step_counter'), 'lat_prev': dict.get('latitude'), 'lon_prev': dict.get('longitude')}
            else:
                print("Default next location for general/car categories")
                nextLocation = {'purpose': 'calculated', 'id': dict.get('id'), 'latitude': dict.get('latitude'), 'longitude': dict.get('longitude'), 'search': dict.get('search'), 'category': dict.get('category'), 'timestamp': dict.get('timestamp'), 'velocity': dict.get('velocity'), 'lat_prev': dict.get('latitude'), 'lon_prev': dict.get('longitude')}
        print('Next Location of user ' + dict.get('id') + ' is: ')
        print nextLocation
        #send next location to current user
        mqttc.publish(publishPath, str(nextLocation),2)
        #next location need to be saved in an array if it exists or not
        appendUserInfo(nextLocation)
        #check if other users are nearby
        minDistance = getMinDistanceUser(nextLocation)
        #send minimum distance to current user
        mqttc.publish(publishPath, str(minDistance),2)
    elif(purpose == 'delete'):
        deleteUser(dict.get('id'))

def on_subscribe(mqttc, userdata, mid, granted_qos):
    print('subscribed (qos=' + str(granted_qos) + ')')

def on_unsubscribe(mqttc, userdata, mid, granted_qos):
    print('unsubscribed (qos=' + str(granted_qos) + ')')
```

```
•
• def on_publish(client, userdata, result):
•     print("data published\n")
•
•
• print('Beginning')
• mqttc = mqtt.Client(client_id="", clean_session=True)
• mqttc.on_connect = on_connect
• mqttc.on_disconnect = on_disconnect
• mqttc.on_message = on_message
• mqttc.on_subscribe = on_subscribe
• mqttc.on_unsubscribe = on_unsubscribe
• mqttc.on_publish = on_publish
• mqttc.tls_set("./certs/mosquitto.org.crt", tls_version=ssl.PROTOCOL_TLSv1_2)
• mqttc.connect('test.mosquitto.org', 8883, 60)
• #mqttc.tls_set("./certs/cert.pem", tls_version=ssl.PROTOCOL_TLSv1_2)
• #mqttc.connect('iot.eclipse.org', 8883, 60)
• #mqttc.connect('iot.eclipse.org', 1883, 60)
• mqttc.subscribe(topic='StreetAdvisor/handleInfo/+', qos=2)
• print('Start loop')
• mqttc.loop_forever()
```

- **calculateDistance.py:**

```
• from math import asin,radians,cos,sin,degrees,atan2,sqrt
• from datetime import datetime
•
• #location of user after 3s
• TIME_SEC = 3
• #earth radius in m
• R = 6371 * 1000
• #max seconds for distance acceptance
• MAX_SECS = 30
• MAX_SECS_PERSON = 200
• #max distance between 2 users
• MAX_DISTANCE = 1000
```

```
#Pi variable
PI = 3.14159265359

#calculate next location(lat,lon) of user
def calculateNextLocation(dict):
    category = dict.get('category')
    response = 'null'
    if category == 'null':
        print "General category"
        response = calculateDistanceGeneral(dict)
    elif category == 'car':
        print "Car category"
        response = calculateDistanceCar(dict)
    else:
        print "Person category"
        response = calculateDistancePerson(dict)
    return response

#calculate next location if user has not logged in and therefore we do not know in whick category he belongs to
def calculateDistanceGeneral(dict):
    #get data info
    latitude = dict.get('latitude')
    longitude = dict.get('longitude')
    direction = dict.get('direction')
    velocity = dict.get('velocity')
    id = dict.get('id')
    search = dict.get('search')
    category = dict.get('category')
    timestamp = dict.get('timestamp')
    distance = velocity * TIME_SEC
    lat2 = asin(sin(radians(latitude)) * cos(distance / R) + cos(radians(latitude)) * sin(distance / R) * cos(radians(direction)))
    lon2 = radians(longitude) + atan2(sin(radians(direction)) * sin(distance / R) * cos(radians(latitude)), cos(distance / R) - sin(radians(latitude)) * sin(lat2))
    lat2 = degrees(lat2)
```

```
    lon2 = degrees(lon2)
    return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': search, 'category': category, 'timestamp': timestamp, 'velocity': velocity, 'lat_prev': latitude, 'lon_prev': longitude}

#calculate next location if a user drives a car
def calculateDistanceCar(dict):
    #get data info
    id = dict.get('id')
    latitude = dict.get('latitude')
    longitude = dict.get('longitude')
    acceleration = dict.get('acceleration')
    velocity = dict.get('velocity')
    direction = dict.get('direction')
    yawRate = dict.get('yaw_rate')
    search = dict.get('search')
    category = dict.get('category')
    timestamp = dict.get('timestamp')
    #Tmdl ~ next position
    if(yawRate == 0):
        print("Yaw Rate is 0")
        distance = (((1/2) * acceleration * pow(TIME_SEC,2)) + (velocity * TIME_SEC))
        lat2 = asin(sin(radians(latitude)) * cos(distance / R) + cos(radians(latitude)) * sin(distance / R) * cos(radians(direction)))
        lon2 = radians(longitude) + atan2(sin(radians(direction)) * sin(distance / R) * cos(radians(latitude)), cos(distance / R) - sin(radians(latitude)) * sin(lat2))
        lat2 = degrees(lat2)
        lon2 = degrees(lon2)
    else:
        # Cx
        distance1 = - ((velocity/yawRate) * sin(radians(direction))) - ((acceleration / pow(yawRate,2)) * cos(radians(direction)))
        Cx = latitude + (180/PI)*(distance1/R)
        # Cy
        distance2 = ((velocity/yawRate) * cos(radians(direction))) - ((acceleration / pow(yawRate,2)) * sin(radians(direction)))
```

```
        Cy = longitude + (180/PI)*(distance2/R)/cos(latitude)
        # Ut
        Ut = acceleration * TIME_SEC + velocity
        # yawAngleT
        yawAngleT = yawRate * TIME_SEC
        distance1 = ((acceleration/pow(yawRate,2)) * cos(radians(yawAngleT))) + ((Ut/yawRate) * sin(radians(yawAngleT)))
        distance2 = ((acceleration/pow(yawRate,2)) * sin(radians(yawAngleT))) - ((Ut/yawRate) * cos(radians(yawAngleT)))
        lat2 = Cx + (180/PI)*(distance1/R)
        lon2 = Cy + (180/PI)*(distance2/R)/cos(Cx)
    return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': search, 'category': category, 'timestamp': timestamp, 'velocity': velocity, 'lat_prev': latitude, 'lon_prev': longitude}


#user category is a person, calculate next location with step length for 3 steps after
def calculateDistancePerson(dict):
    #get data info
    id = dict.get('id')
    latitude = dict.get('latitude')
    longitude = dict.get('longitude')
    direction = dict.get('direction')
    stepLength = dict.get('step_length')
    stepCounter = dict.get('step_counter')
    search = dict.get('search')
    category = dict.get('category')
    timestamp = dict.get('timestamp')
    distance = stepCounter * stepLength
    lat2 = asin(sin(radians(latitude)) * cos(distance / R) + cos(radians(latitude)) * sin(distance / R) * cos(radians(direction)))
    lon2 = radians(longitude) + atan2(sin(radians(direction)) * sin(distance / R) * cos(radians(latitude)), cos(distance / R) - sin(radians(latitude)) * sin(lat2))
    lat2 = degrees(lat2)
    lon2 = degrees(lon2)
```

```
    return {'purpose': 'calculated', 'id': id, 'latitude': lat2, 'longitude': lon2, 'search': search, 'category': category, 'timestamp': timestamp, 'step_length': stepLength, 'step_counter': stepCounter, 'lat_prev': latitude, 'lon_prev': longitude}


#calculate the distance between 2 users(with lat,lon)
def calculateDistanceUsers(user1, user2):
    if(calculateWeightTime(user1,user2)):
        lat1 = radians(user1.get('latitude'))
        lon1 = radians(user1.get('longitude'))
        lat2 = radians(user2.get('latitude'))
        lon2 = radians(user2.get('longitude'))
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
        c = 2 * atan2(sqrt(a), sqrt(1 - a))
        distance = R * c
        return distance
    else:
        return MAX_DISTANCE


#calculate weight of secs between 2 users and return if it is acceptable
def calculateWeightTime(user1,user2):
    timestampUser1 = int(user1.get('timestamp'))/1000
    timestampUser2 = int(user2.get('timestamp'))/1000
    datetime1 = datetime.fromtimestamp(timestampUser1)
    datetime2 = datetime.fromtimestamp(timestampUser2)
    #get seconds of last location calculated
    if(datetime1 > datetime2):
        diff = (datetime1-datetime2).total_seconds()
    else:
        diff = (datetime2-datetime1).total_seconds()
    #get weight depending on speed or speed length of user2
    velocity = -1
    stepLength = -1
```

```
    speed = 0   #use velocity
    if 'velocity' in user2:
        velocity = user2.get('velocity')
    else:
        stepLength = user2.get('step_length')
        speed = 1
    if(speed == 0):
        if(velocity < 1 and velocity >= 0):
            if(diff <= MAX_SECS):
                return 1
            else:
                return 0
        weight = diff / velocity
        if(weight <= MAX_SECS):
            return 1
        else:
            return 0
    else:
        if(stepLength == 0):
            if(diff <= MAX_SECS_PERSON):
                return 1
            else:
                return 0
        return 1
```

- **dataHandle.py:**

```
from calculateDistance import calculateDistanceUsers

#list of users
dataUsers = []
#list of distances of between a user and others
distancesCurrentUser = []
#max distance between 2 users
MAX_DISTANCE = 1000

```

```
#insert user in list or edit if already exists
def appendUserInfo(currentUser):
    #user's id must not be null
    if(currentUser.get('id') == 'null'):
        return
    if(len(dataUsers) == 0):
        dataUsers.append(currentUser)
        return
    check = 0
    for user in dataUsers:
        #check if user exists in list
        if(user['id'] == currentUser.get('id')):
            check = 1
            userUpdate = user
    if(check == 0):
        dataUsers.append(currentUser)
    else:
        userUpdate.update(currentUser)

#get min distance from all
def getMinDistanceUser(currentUser):
    minDistance = MAX_DISTANCE
    searchCurrent = currentUser.get('search')
    if(currentUser.get('id') == 'null'):
        return {'minDistance': minDistance, 'id': currentUser.get('id')}
    if(searchCurrent == 'none'):
        return {'minDistance': minDistance, 'id': currentUser.get('id')}
    for user in dataUsers:
        if(user['id'] != currentUser.get('id')):
            #same preferences-category
            categoryUser = user['category']
            if(categoryUser == searchCurrent or searchCurrent == 'all_occasions' or c
ategoryUser == 'null'):
                distancesCurrentUser.append(calculateDistanceUsers(currentUser,user
))
    if(len(distancesCurrentUser) > 0):
```

```
        minDistance = min(distancesCurrentUser)
    #delete list before every search
    del distancesCurrentUser[:]
    return {'minDistance': minDistance, 'id': currentUser.get('id')}

#delete user when tracking stops
def deleteUser(id):
    if(id == 'null'):
        return
    for user in dataUsers:
        if(user['id'] == id):
            print('Delete user with id: ' + id)
            dataUsers.remove(user)
```

[4] Παρακάτω διατίθενται οι βιβλιοθήκες που χρησιμοποιήθηκαν για να υλοποιηθεί ο Client:

Πίνακας 7 : Βιβλιοθήκες Android Studio

Android Studio	com.android.tools.build:gradle	3.3.1
Google	com.google.gms:google-services	4.2.0
Support	androidx.appcompat:appcompat:$androidSupportAppCompatVersion androidx.preference:preference:$androidSupportPrefMatVersion com.google.android.material:material:$androidSupportPrefMatVersion androidx.exifinterface:exifinterface:$androidSupportVersion androidx.legacy:legacy-support-v4:$androidSupportVersion	androidSupportVersion = '1.0.0' androidSupportAppCompatVersion = '1.0.2' androidSupportPrefMatVersion = '1.0.0' androidSupportCoreVersion = '1.0.1'
Firebase	com.google.firebase:firebase-core:$androidFirebaseVersion com.google.firebase:firebase-database:$androidFirebaseDataVersion com.google.firebase:firebase-auth:$androidFirebaseAuthVersion com.firebase:firebase-jobdispatcher:$androidFirebaseJobDispatcherVersion	androidFirebaseVersion = "16.0.4" androidFirebaseAuthVersion = "16.1.0" androidFirebaseDataVersion = "16.0.6"

		androidFirebaseJob DispatcherVersion = "0.8.5"
Butterknife	com.jakewharton:butterknife:$butterknifeVersion com.jakewharton:butterknife-compiler:$butterknifeVersion	10.1.0
MPAndroidChart	com.github.PhilJay:MPAndroidChart:v$MPAndroidChartVersion	3.0.3
Picasso	com.squareup.picasso:picasso:$picassoVersion	2.71828
CircleImageView	de.hdodenhof:circleimageview:$circleImageViewVersion	3.0.0
Google Play Services	com.google.android.gms:play-services-maps:$androidGooglePlayServicesMapsVersion com.google.android.gms:play-services-location:$androidGooglePlayServicesVersion	androidGooglePlayServicesVersion = "16.0.0" androidGooglePlayServicesMapsVersion = "16.1.0"
JSON conversion	com.google.code.gson:gson:$androidGsonVersion	2.8.5
DrawRouteMaps	com.github.ar-android:DrawRouteMaps:$drawRouteMapsVersion	1.0.0
Notification	androidx.core:core:$androidSupportCoreVersion	28.0.0
MQTT	org.eclipse.paho:org.eclipse.paho.client.mqttv3:$eclipsePahoMQTTVersion org.eclipse.paho:org.eclipse.paho.android.service:$eclipsePahoAndroidServVersion	eclipsePahoMQTTVersion = "1.1.0" eclipsePahoAndroidServVersion = "1.1.1"
Testing	junit:junit:$junitVersion androidx.test.ext:junit:$androidXJunitVersion androidx.test:runner:$testRunnerVersion androidx.test:rules:$testRulesVersion androidx.test.espresso:espresso-core:$testEspressoVersion androidx.test.uiautomator:uiautomator:$testAutomatorVersion org.hamcrest:hamcrest-	testHamcrestVersion = '1.3' testAutomatorVersion = '2.2.0' junitVersion = "4.12" androidXJunitVersion = "1.1.0" testRunnerVersion = '1.1.0-alpha4' testRulesVersion =

	library:$testHamcrestVersion com.android.support.test.espresso:espresso-contrib:$testEspressoContribVersion	'1.1.1'testEspressoVersion = '3.1.0-alpha4' testEspressoContribVersion = '2.2.2'
Logger	com.orhanobut:logger:$loggerVersion	2.2.0

ΑΝΑΦΟΡΕΣ

[1] Haruka Tonoki, Ayanori Yorozu and Masaki Takahashi, Model-based Pedestrian Trajectory Prediction using Environmental Sensor for Mobile Robots Navigation, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 2, 2017.

[2] Pubudu N. Pathirana, Andrey V. Savkin, and Sanjay Jha, Location Estimation and Trajectory Prediction for Cellular Networks With Mobile Base Stations, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, VOL. 53, NO. 6, NOVEMBER 2004.

[3] Shun Yoshimi, Kazuya Murao, Kohei Kanagu, Nobuhiko Nishio and Masahiro Mochizuki, PDR Trajectory Estimation using Pedestrian-Space Constraints: Real World Evaluations, UBICOMP/ISWC '15 ADJUNCT, SEPTEMBER 7–11, 2015, OSAKA, JAPAN.

[4] Ľubica ILKOVIČOVÁ, Pavol KAJÁNEK and Alojz KOPÁČIK, Pedestrian Trajectory Determination in Indoor Environment, FIG Working Week 2015, From the Wisdom of the Ages to the Challenges of the Modern World, Sofia, Bulgaria, 17-21 May 2015

[5] JSON – Introduction; https://www.w3schools.com/js/js_json_intro.asp. [Προσπελάστηκε: 1/1/19]

[6] Activity; https://developer.android.com/reference/android/app/Activity. [Προσπελάστηκε: 5/1/19]

[7] Python Client – documentation; https://www.eclipse.org/paho/clients/python/docs/. [Προσπελάστηκε: 27/8/18]

[8] Planning screens and their relationships; https://developer.android.com/training/design-navigation/screen-planning. [Προσπελάστηκε: 3/1/19]

[9] Build an App Widget; https://developer.android.com/guide/topics/appwidgets/. [Προσπελάστηκε: 5/1/19]

[10] Services overview; https://developer.android.com/guide/components/services. [Προσπελάστηκε: 5/1/19]

[11] JobService; https://developer.android.com/reference/android/app/job/JobService. [Προσπελάστηκε: 5/1/19]

[12] Josh Long, I Don't Speak Your Language: Frontend vs. Backend, 2013; https://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend. [Προσπελάστηκε: 5/1/19]

[13] Google, 6 Feb. 2019; https://el.wikipedia.org/wiki/Google. [Προσπελάστηκε: 5/1/19]

[14] Android, 12 Oct. 2018; https://el.wikipedia.org/wiki/Android. [Προσπελάστηκε: 5/1/19]

[15] DxDiag, 8 Oct. 2018; https://en.wikipedia.org/wiki/DxDiag. [Προσπελάστηκε: 5/1/19]

[16] Flowchart, 1 March 2019; https://en.wikipedia.org/wiki/Flowchart. [Προσπελάστηκε: 5/1/19]

[17] Firebase, 20 Feb. 2019; https://en.wikipedia.org/wiki/Firebase. [Προσπελάστηκε: 5/1/19]

[18] Processes and threads overview; https://developer.android.com/guide/components/processes-and-threads. [Προσπελάστηκε: 5/1/19]

[19] UI, 19 Feb. 2019; https://en.wikipedia.org/wiki/User_interface. [Προσπελάστηκε: 5/1/19]

[20] Robert Morris, John Jannotti, Frans Kaashoek, Jinyang Li, Douglas Decouto, CarNet: A Scalable Ad Hoc Wireless Network System, Published in the proceedings of the 9th ACM SIGOPS European Workshop, September 2000; https://pdos.csail.mit.edu/~rtm/papers/carnet00-abstract.html. [Προσπελάστηκε: 5/1/19]

[21] David M, Get lat/long given current point, distance and bearing, Accepted answer, 20 Oct. 2011; https://stackoverflow.com/questions/7222382/get-lat-long-given-current-point-distance-and-bearing. [Προσπελάστηκε: 15/5/18]

[22] Michael0x2a, Getting distance between two points based on latitude/longitude, Accepted answer, 16 Oct. 2016; https://stackoverflow.com/questions/19412462/getting-distance-between-two-points-based-on-latitude-longitude. [Προσπελάστηκε: 20/6/18]

[23] Pencil Project, Top Features of Pencil; https://pencil.evolus.vn/Features.html. [Προσπελάστηκε: 26/2/19]

[24] MQTT.ORG, Frequently Asked Questions; http://mqtt.org/faq. [Προσπελάστηκε: 26/2/19]

[25] MVC Framework – Introduction; https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm. [Προσπελάστηκε: 29/1/19]

[26] Στατιστικά στοιχεία τροχαίας α' εξάμηνο 2018; http://www.astynomia.gr/index.php?option=ozo_content&perform=view&id=81882&Itemid=73&lang. [Προσπελάστηκε: 27/2/19]

[27] Electronic Stability Control; http://www.howsafeisyourcar.com.au/Electronic-Stability-Control/. [Προσπελάστηκε: 27/2/19]

[28] Car Safety Feature, Auto Emergency Braking (AEB); http://www.howsafeisyourcar.com.au/aeb/. [Προσπελάστηκε: 27/2/19]

[29] Intelligent Speed Assist (ISA); http://www.howsafeisyourcar.com.au/Safety-Features/Safety-Features-List/Intelligent-Speed-Assist-ISA/.[Προσπελάστηκε: 27/2/19]

[30] Self-driving car, 5 March 2019; https://en.wikipedia.org/wiki/Self-driving_car. [Προσπελάστηκε: 27/2/19]
[31] Fathima Dilhasha, How to optimize connections between MQTT clients and ActiveMQ broker, 29 May 2017; https://medium.com/@dilhasha.nazeer/how-to-optimize-connections-between-mqtt-clients-and-activemq-broker-6a27ae612ddc. [Προσπελάστηκε: 21/1/19]
[32] Anupam Chugh, Android Google Maps Current Location, Night Mode Features; https://www.journaldev.com/13358/android-google-maps-current-location-night-mode. [Προσπελάστηκε: 28/1/19]
[33] Bound services overview; https://developer.android.com/guide/components/bound-services. [Προσπελάστηκε: 1/2/19]
[34] Philipp Jahoda, Interaction with the Chart, 4 Aug. 2017; https://github.com/PhilJay/MPAndroidChart/wiki/Interaction-with-the-Chart. [Προσπελάστηκε: 1/2/19]
[35] The HiveMQ Team, How does TLS affect MQTT performance?, 21 March 2016; https://www.hivemq.com/blog/how-does-tls-affect-mqtt-performance/. [Προσπελάστηκε: 9/2/19]
[36] steve, Mosquitto SSL Configuration -MQTT TLS Security, 5 January 2019; http://www.steves-internet-guide.com/mosquitto-tls/. [Προσπελάστηκε: 9/2/19]
[37] Motion sensors; https://developer.android.com/guide/topics/sensors/sensors_motion#java. [Προσπελάστηκε: 10/2/19]
[38] Chain of Trust; https://letsencrypt.org/certificates/. [Προσπελάστηκε: 16/2/19]
[39] foibs, When to use accelerometer or gyroscope on Android, Accepted Answer, 19 Dec. 2013; https://stackoverflow.com/questions/20693547/when-to-use-accelerometer-or-gyroscope-on-android. [Προσπελάστηκε: 22/2/19]
[40] Ashraff Hathibelagal, Android Sensors in Depth: Proximity and Gyroscope, 27 Jan. 2017; https://code.tutsplus.com/tutorials/android-sensors-in-depth-proximity-and-gyroscope--cms-28084. [Προσπελάστηκε: 24/2/19]
[41] Average Acceleration; https://www.mansfieldct.org/Schools/MMS/staff/hand/lawsaveaccel2003.htm. [Προσπελάστηκε: 24/2/19]
[42] Adam Houenou, Philippe Bonnifait, Véronique Cherfaoui, Yao Wen. Vehicle Trajectory Prediction based on Motion Model and Maneuver Recognition. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013), Nov 2013, Tokyo, Japan. pp.4363-4369.
[43] Tunneling protocol, 13 Feb. 2019; https://en.wikipedia.org/wiki/Tunneling_protocol. [Προσπελάστηκε: 1/3/2019]
[44] TLS, 25 Nov. 2018; https://el.wikipedia.org/wiki/TLS. [Προσπελάστηκε: 1/3/19]
[45] Αποσφαλμάτωση, 28 June 2017; https://el.wikipedia.org/wiki/Αποσφαλμάτωση. [Προσπελάστηκε: 1/3/19]
[46] Fundamentals of Testing; https://developer.android.com/training/testing/fundamentals. [Προσπελάστηκε: 1/3/19]
[47] Android Studio, 21 Oct. 2017; https://el.wikipedia.org/wiki/Android_Studio. [Προσπελάστηκε 1/3/19].
[48] SonarLint; https://www.sonarlint.org/. [Προσπελάστηκε: 1/3/19]
[49] Md. Abu Nafee Ibna Zahid, Measure time elapsed in Python?, First Answer, 10 Sep 2011; https://stackoverflow.com/questions/7370801/measure-time-elapsed-in-python. [Προσπελάστηκε: 3/3/19]
[50] Francis Brosnan Blázquez, public_brokers, 18 Jan. 2019; https://github.com/mqtt/mqtt.github.io/wiki/public_brokers. [Προσπελάστηκε: 4/3/19]