ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Εφαρμογή εντοπισμού και πλοήγησης σε εσωτερικό χώρο με τη χρήση τεχνολογίας Bluetooth beacons

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νικόλαου Ε. Ανδρουλάκη

Επιβλέπων : Ανδρέας Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2018

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Εφαρμογή εντοπισμού και πλοήγησης σε εσωτερικό χώρο με τη χρήση τεχνολογίας Bluetooth beacons

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νικόλαου Ε. Ανδρουλάκη

Επιβλέπων : Ανδρέας Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25η Οκτωβρίου 2018.

.....	.....	.....
Α. Σταφυλοπάτης	Κ. Καρπούζης	Γ. Στάμου
Καθηγητής Ε.Μ.Π.	Ερευνητής Συνεργάτης	Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2018

.....
Νικόλαος Ε. Ανδρουλάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μιας εφαρμογής κινητού που επιτρέπει την πλοήγηση σε εσωτερικούς χώρους και, πιο συγκεκριμένα, την πλοήγηση προς την έξοδο ενός δωματίου. Για αυτό το σκοπό, γίνεται έρευνα πάνω στην τεχνολογία των Bluetooth Beacons, συσκευών που επιτρέπουν την εκτίμηση της θέσης του κινητού στο χώρο. Για να επιτευχθεί μεγαλύτερη ακρίβεια, γίνεται αναπαράσταση του χώρου και των πιθανών μονοπατιών σαν ένας γράφος, όπου σε κάθε κόμβο γίνονται μετρήσεις των αποστάσεων από τα beacons. Με δεδομένη την τρέχουσα πληροφορία της θέσης και της χαρτογράφησης του χώρου, γίνεται πρόταση βέλτιστης διαδρομής στο χρήστη. Στη συνέχεια παρουσιάζονται ο σχεδιασμός και η υλοποίηση της εφαρμογής, δίνοντας έμφαση τόσο στο αρχιτεκτονικό μοντέλο και τη δομή, όσο και στις τεχνολογίες και στους αλγορίθμους που χρησιμοποιήθηκαν. Τέλος, πραγματοποιείται αξιολόγηση και σχολιασμός του τελικού αποτελέσματος με σκοπό την εξαγωγή συμπερασμάτων και, επιπροσθέτως, προτείνονται πιθανές ιδέες και δυνατότητες επέκτασης.

Λέξεις-κλειδιά

Beacons, Bluetooth, στιγματοθέτηση, πλοήγηση εσωτερικού χώρου, εφαρμογή κινητού, android studio, java, xml, android.

Abstract

The purpose of this thesis is the development of a mobile application that allows indoor navigation and, more specifically, navigation towards the exit of a room. For this purpose, a study is being made on the technology of Bluetooth Beacons, devices that facilitate the estimation of the position of the mobile phone. In order to achieve greater precision, the surrounding space is mapped as a graph, where distance to each beacons is calculated from each vertex. Given the current information regarding the position and the mapping of the room, an estimation of the best route is suggested to the user. Next the design and development of the application are documented, emphasizing as much in the architectural model and structure, as in the technologies and the algorithms that were used. Finally, an evaluation and comment of the final result are made with the purpose of arriving to conclusions and, additionally, ideas and possibilities of expansion are suggested.

Keywords

Beacons, Bluetooth, positioning, indoor navigation, mobile application, android studio, java, xml, android.

Ευχαριστίες

Με την εκπόνηση αυτής της διπλωματικής εργασίας κλείνει ο πολυετής κύκλος της προπτυχιακής φοίτησής μου στο Εθνικό Μετσόβιο Πολυτεχνείο και σε αυτό το σημείο θα ήθελα να αναφερθώ σε όλους αυτούς που στάθηκαν στο πλευρό μου.

Αρχικά, θα ήθελα να εκφράσω τις ευχαριστίες μου προς τον κ. Καρπούζη για όλη τη βοήθεια και το χρόνο που μου αφιέρωσε ώστε να φέρω εις πέρας αυτό το έργο καθώς και τους κ. Σταφυλοπάτη και κ. Στάμου για την αποδοχή αυτής της εργασίας.

Στη συνέχεια, θα ήθελα να ευχαριστήσω τους φίλους μου που έκαναν όλα αυτά τα χρόνια πιο ευχάριστα και παράλληλα με βοήθησαν να διευρύνω τους ορίζοντες μου και να διαμορφώσω το χαρακτήρα μου.

Τέλος, δεν θα μπορούσα να ξεχάσω την οικογένειά μου που ήταν πάντα μαζί μου σε επιτυχίες και αποτυχίες και δεν σταμάτησε να πιστεύει σε μένα, και που, εξ αρχής, με εφοδίασε με το σθένος και την υπομονή ώστε να φτάσω μέχρι εδώ.

Περιεχόμενα

Εισαγωγή

Υπόβαθρο

Συστήματα Στιγματοθέτησης Εσωτερικού Χώρου

Ένα σύστημα στιγματοθέτησης εσωτερικού χώρου (Indoor positioning system) είναι ένα σύστημα το οποίο υπολογίζει τη θέση αντικειμένων ή ανθρώπων μέσα σε ένα κτήριο με τη χρήση φωτός, ραδιοκυμάτων, μαγνητικών πεδίων, ακουστικών σημάτων ή άλλων πληροφοριών από αισθητήρες που συλλέγονται από μία κινητή συσκευή(Curran et al., 2011). Παρότι υπάρχουν αρκετά συστήματα στο εμπόριο, δεν υπάρχει κάποιο στάνταρ για τα συστήματα στιγματοθέτησης εσωτερικού χώρου.

Τα συστήματα στιγματοθέτησης εσωτερικού χώρου χρησιμοποιούν διάφορες τεχνολογίες κατά την υλοποίηση τους, όπως για παράδειγμα μετρήσεις αποστάσεων από σταθερούς κόμβους στον περιβάλλοντα χώρο (οι κόμβοι αυτοί είναι γνωστοί και ως σταθερά σημεία, όπως WiFi/LiFi access points ή Bluetooth Beacons), μαγνητική στιγματοθέτηση, dead reckoning(Curran et al., 2011; Qiu & Mutka, 2016). Τα συστήματα αυτά είτε προσπαθούν δυναμικά να προσδιορίσουν τη θέση κινητών συσκευών είτε παρέχουν ένα περιβάλλον σύστημα που θα αναγνωρίζει την ύπαρξη των συσκευών(Furey, Curran, & McKevitt, 2012).

Η μικρής κλίμακας φύση των συστημάτων έχει σαν αποτέλεσμα την τμηματοποίηση του σχεδιασμού αυτών. Έτσι βλέπουμε συστήματα να χρησιμοποιούν διάφορες οπτικές(Liu, Xiaohan, Makino, & Mase, 2010), ραδιφωνικές(Chang, Rashidzadeh, & Ahmadi, 2010)(Chiou, Wang, & Yeh, 2010)(Lim, Kung, Hou, & Luo, 2008)(Reza & Geok, 2008)(Zhou, Law, Guan, & Chin, 2011) ή ακόμα και ακουστικές(Pontes, Maciel, & Linhares, 2015)(Qiu & Mutka, 2017) τεχνολογίες.

Ο σχεδιασμός των συστημάτων πρέπει να λαμβάνει υπόψη ότι απαιτούνται τουλάχιστον τρία ανεξάρτητα σημεία μετρήσεων ώστε να προσδιοριστεί μία θέση χωρίς αμφιβολία. Για εξομάλυνση των αποτελεσμάτων πρέπει να υπάρχει μία λογική μέθοδος απαλοιφής σφάλματος ώστε να αντισταθμιστούν τα στοχαστικά λάθη. Το σύστημα ενδέχεται να περιλαμβάνει πληροφορία από άλλα συστήματα ώστε να αντιμετωπίσει την φυσική αμφιβολία και να επιτρέψει την αντιστάθμιση των σφαλμάτων.

Η εποπτεία του προσανατολισμού της συσκευής μπορεί να επιτευχθεί είτε αναγνωρίζοντας γνωστά σημεία μέσα σε εικόνες που λαμβάνονται σε πραγματικό χρόνο, είτε χρησιμοποιώντας trilateration με beacons(Hile & Borriello, 2008). Επίσης υπάρχουν τεχνολογίες για την εύρεση μαγνητομετρικής πληροφορίας μέσα σε κτίρια ή τοποθεσίες με μεταλλικές κατασκευές ή μέσα σε ορυχεία σιδήρου.

Ανάγκη για Συστήματα Στιγματοθέτησης Εσωτερικού Χώρου

Λόγω της εξασθένησης του σήματος που προκαλείται από τα κατασκευαστικά υλικά, το Παγκόσμιο Σύστημα Στιγματοθέτησης (GPS) χάνει σημαντικό μέρος της δύναμής του σε εσωτερικούς χώρους, επηρεάζοντας την απαιτούμενη κάλυψη των δεκτών κατά τουλάχιστον τέσσερις δορυφόρους. Επιπροσθέτως, οι πολλαπλές ανακλάσεις στις επιφάνειες προκαλούν τη δημιουργία πολλών μονοπατιών ταυτόχρονα και οδηγούν σε μη ελέγξιμα σφάλματα. Αυτά τα φαινόμενα υποβαθμίζουν όλες τις γνωστές λύσεις για στιγματοθέτηση σε εσωτερικούς χώρους με χρήση ηλεκτρομαγνητικών κυμάτων από πομπούς εσωτερικού χώρου σε δέκτες εσωτερικού χώρου. Ένα σύνολο από φυσικές και μαθηματικές μεθόδους εφαρμόζονται με σκοπό την αντιστάθμιση των σφαλμάτων που προκύπτουν από αυτά τα προβλήματα. Η χρήση εναλλακτικών πηγών πληροφορίας πλοήγησης, όπως η μονάδα μέτρησης αδράνειας (IMU), η μονοφωνική κάμερα “Ταυτόχρονη στιγματοθέτηση και χαρτογράφηση” (SLAM) και WiFi SLAM, επέτρεψαν νέες ελπιδοφόρες οδούς προς την διόρθωση σφάλματος στην σιγματοθέτηση με ραδιοσυχνότητες. Τέλος, η ενσωμάτωση δεδομένων

από διαφορετικά συστήματα πλοήγησης με διαφορετικές φυσικές αρχές μπορούν να βελτιώσουν την ακρίβεια και την ευρωστία της συνολικής λύσης.

Τεχνολογία Bluetooth low energy

Το Bluetooth χαμηλής ενέργειας (Bluetooth LE) είναι ένα ασύρματο δίκτυο προσωπικής περιοχής σχεδιασμένο από την Bluetooth Special Interest Group (Bluetooth SIG) και έχει σαν στόχο τη υποστήριξη εφαρμογών στους χώρους της Υγείας, της Γυμναστικής, των Beacons, της Ασφάλειας και στις βιομηχανίες διασκέδασης στο σπίτι. Συγκριτικά με την τεχνολογία του κλασικού Bluetooth, το Bluetooth χαμηλής ενέργειας έχει σχεδιαστεί με γνώμονα την παροχή αισθητά μειωμένης ενεργειακής κατανάλωσης και κόστους ενώ παράλληλα διατηρώντας ανάλογο εύρος επικοινωνίας.

Κινητά λειτουργικά συστήματα όπως τα iOS, Android, Windows Phone και BlackBerry, καθώς και τα λειτουργικά συστήματα MacOS, Linux, Windows 8 και Windows 10, υποστηρίζουν το Bluetooth χαμηλής ενέργειας. Η Bluetooth SIG προβλέπει ότι μέχρι το 2018, περισσότερες από το 90% των έξυπνων κινητών τηλεφώνων που υποστηρίζουν Bluetooth, θα υποστηρίζουν και το Bluetooth χαμηλής ενέργειας.

Bluetooth Beacons

Τα Bluetooth beacons είναι συσκευές εκπομπής. Ανήκουν στην κατηγορία των συσκευών Bluetooth χαμηλής ενέργειας που εκπέμπουν την πληροφορία αναγνώρισής τους σε φορητές ηλεκτρονικές συσκευές γύρω τους. Η τεχνολογία επιτρέπει σε έξυπνα κινητά τηλέφωνα, ταμπλέτες και άλλες συσκευές να εκτελέσουν ενέργειες όταν είναι κοντά σε κάποιο beacon.

Τα Bluetooth beacons χρησιμοποιούν αισθητήρες εγγύτητας μέσω Bluetooth χαμηλής ενέργειας ώστε να μεταδώσουν έναν καθολικά μοναδικό αναγνωριστικό κώδικα ο οποίος λαμβάνεται από μία συμβατή εφαρμογή ή ένα λειτουργικό σύστημα. Ο αναγνωριστικός κώδικας και αρκετά bytes, που εκπέμπονται μαζί του, μπορούν να χρησιμοποιηθούν για τον προσδιορισμό της φυσικής θέσης της συσκευής, για την παρακολούθηση πελατών, ή για την ενεργοποίηση κάποια ενέργειας που εξαρτάται από κάποια συγκεκριμένη θέση όπως ένα check-in σε μέσα κοινωνικής δικτύωσης ή μία ωθούμενη ειδοποίηση.

Μία εφαρμογή μοιράζει μηνύματα σε ένα συγκεκριμένο σημείο ενδιαφέροντος, για παράδειγμα ένα κατάστημα, μία στάση λεωφορείου, ένα δωμάτιο ή μία ακόμα πιο συγκεκριμένη τοποθεσία όπως ένα έπιπλο ή έναν αυτόματο πωλητή. Αυτή η τεχνολογία είναι παρόμοια με την τεχνολογία geopush που έχει χρησιμοποιηθεί στο παρελθόν και βασίζεται στο GPS, αλλά με σημαντικά μειωμένη επίδραση στην διάρκεια ζωής της μπαταρίας και πολύ μεγαλύτερη ακρίβεια.

Μία άλλη εφαρμογή είναι σε ένα σύστημα στιγματοθέτησης εσωτερικού χώρου που βοηθάει έξυπνα κινητά τηλέφωνα να προσδιορίσουν την θέση τους ή το περιβάλλον τους. Με τη βοήθεια ενός Bluetooth beacon, το λογισμικό ενός έξυπνου κινητού τηλεφώνου μπορεί κατά προσέγγιση, να βρει την σχετική θέση ως προς ένα beacon μέσα σε ένα κατάστημα. Ήδη καταστήματα φυσικού χώρου χρησιμοποιούν beacons για διαφήμιση σε κινητά τηλέφωνα, προτείνοντας ειδικές προσφορές στους πελάτες τους, και μπορούν να ενεργοποιήσουν ηλεκτρονικές πληρωμές σε κινητές συσκευές μέσω συστημάτων POS.

Τα Bluetooth beacons διαφέρουν από άλλες τεχνολογίες που ασχολούνται με τον περιβάλλοντα χώρο επειδή η συσκευή-πομπός (beacon) είναι απλά ένας πομπός μία διαδρομής προς τη συσκευή-δέκτη (κινητό τηλέφωνο ή άλλη συσκευή), και απαιτείται μία συγκεκριμένη εφαρμογή εγκατεστημένη στη συσκευή ώστε να αλληλεπιδράσει με το beacon. Αυτό διασφαλίζει ότι μονάχα η εγκατεστημένη εφαρμογή μπορεί να παρακολουθήσει τη δραστηριότητα χρηστών, και συνεπώς δεν μπορεί να γίνει χρήση παρά τη θέλησή τους ενώ βρίσκονται σε έναν χώρο όπου υπάρχουν πομποί.

Ιστορία των Beacons

Η ανάπτυξη της ραδιοτεχνολογίας “short-link”, που αργότερα ονομάστηκε Bluetooth, άρχισε το 1989 από τον Dr. Nils Rydbeck, CTO στην εταιρία Ericsson Mobile, στην πόλη Lund, και από τον Dr. Johan Ullman. Ο σκοπός ήταν να αναπτυχθούν ασύρματα ακουστικά, με βάση τις δύο εφευρέσεις του Johan Ullman, SE 89020986, κατοχυρωμένη στις 12-06-1989, και SE 9202239, κατοχυρωμένη στις 24-07-1992. Από τη δημιουργία του μέχρι σήμερα, το Bluetooth στάνταρ έχει περάσει πολλές γενιές με κάθε μία να προσθέτει διαφορετικά χαρακτηριστικά. Το Bluetooth 1.2 επέτρεψε πιο γρήγορες ταχύτητες μέχρι και 700 Kbps. Το Bluetooth 2.0 βελτίωσε αυτές τις ταχύτητες στα 3Mbps. Το Bluetooth 2.1 βελτίωσε την ταχύτητα ζευγαρώματος συσκευών και ζητήματα ασφάλειας. Το Bluetooth 3.0 βελτίωσε την ταχύτητα μεταφοράς για μία ακόμη φορά, φτάνοντας τα 24 Mbps. Το 2010 κυκλοφόρησε το Bluetooth 4.0 (χαμηλής ενέργειας) με κεντρική εστίαση στη μείωση της κατανάλωσης ενέργειας. Πριν από το Bluetooth 4.0 οι πλειοψηφία των συνδέσεων μέσω Bluetooth ήταν αμφίδρομες, με τις δύο συσκευές να “ακούν” και να “μιλούν” μεταξύ τους. Παρ’ότι αυτή η λειτουργία είναι ακόμα δυνατή στο Bluetooth 4.0, η μονομερής επικοινωνία είναι επίσης δυνατή. Αυτή η μονομερής επικοινωνία επιτρέπει μία συσκευή Bluetooth να εκπέμπει πληροφορία αλλά να μην λαμβάνει τίποτα. Αυτές οι συσκευές ονομάστηκαν “Φάροι” (Beacons) και δεν απαιτούν ζευγάρωμα συσκευών όπως οι προηγούμενες συσκευές Bluetooth και έχουν ανοίξει το δρόμο για νέες χρήσιμες εφαρμογές.

Beacons και Πλοήγηση Εσωτερικού Χώρου

Προς το παρόν η στιγματοθέτηση εσωτερικού χώρου με χρήση beacons χωρίζεται σε τρεις κατηγορίες. Εφαρμογές με αρκετά beacons ανά δωμάτιο, εφαρμογές με ένα beacon ανά δωμάτιο, και εφαρμογές με μερικά beacons ανά κτίριο. Η πλοήγηση εσωτερικού χώρου με χρήση bluetooth είναι ακόμα στα πρώτα της βήματα αλλά έχουν γίνει αρκετές προσπάθειες για την εύρεση μιας λειτουργικής λύσης.

Με χρήση αρκετών beacons ανά δωμάτιο μπορεί να εκτιμηθεί η θέση κάποιου χρήστη με τη μέθοδο trilateration με ακρίβεια περίπου 2 μέτρων. Τα Bluetooth beacons είναι ικανά να εκπέμπουν την ένδειξη έντασης του ληφθέντος σήματος (RSSI) μαζί με άλλα δεδομένα. Αυτή η πληροφορία υπολογίζεται από τον κατασκευαστή του beacon και αντιπροσωπεύει την ένταση του σήματος του beacon σε μία γνωστή απόσταση, συνήθως στο ένα μέτρο. Χρησιμοποιώντας τη γνωστή ένταση σήματος του beacon και την ένταση που αναφέρει το beacon στη συσκευή με την οποία επικοινωνεί, γίνεται μία εκτίμηση για την απόσταση ανάμεσα στη συσκευή και το beacon. Ωστόσο αυτή η εκτίμηση δεν είναι καθόλου αξιόπιστη, έτσι για μεγαλύτερη ακρίβεια, άλλες μέθοδοι παρακολούθησης θέσης και στιγματοθέτησης προτιμώνται. Από την κυκλοφορία τους το 2010 μέχρι σήμερα, αρκετές μελέτες έχουν χρησιμοποιήσει Bluetooth beacons για παρακολούθηση θέσης. Αρκετές μέθοδοι έχουν δοκιμαστεί ώστε να βρεθεί ο βέλτιστος τρόπος συνδυασμού των τιμών RSSI και οδηγεί σε μεγαλύτερη ακρίβεια. Τα νευρωνικά δίκτυα έχουν προταθεί σαν ένας αποτελεσματικός τρόπος αντιμετώπισης του σφάλματος. Τέλος, μία προσέγγιση με χρήση στιγμάτων έχει επίσης δοκιμαστεί, όπου η εκτίμηση της θέσης του χρήστη γίνεται με τη χρήση ενός χάρτη πυκνότητας.

Πρωτόκολλα

iBeacon

Το iBeacon είναι ένα πρωτόκολλο ανεπτυγμένο από την Apple το 2013. Αρχικά χρησιμοποιήθηκε από την εταιρία σε 254 καταστήματά της στις ΗΠΑ με σκοπό τόσο την απλοποίηση των πληρωμών όσο και για διαφημιστικούς λόγους. Αρκετοί κατασκευαστές έκτοτε δημιούργησαν Bluetooth beacons συμβατά με αυτό το πρωτόκολλο. Από το Μάιο του 2014, συσκευές που υλοποιούν το iBeacon πρωτόκολλο κυκλοφορούν στην αγορά σε μεγάλο εύρος τιμών από 5$ μέχρι περισσότερο από 30$. Κάθε μία από αυτές τις συσκευές έχουν ποικίλες εργοστασιακές ρυθμίσεις σε ό,τι αφορά στην ένταση και στη συχνότητα εκπομπής.

AltBeacon

Το πρωτόκολλο AltBeacon είναι μία εναλλακτική ανοικτού κώδικα για το iBeacon και έχει δημιουργηθεί από την Radius Networks. Το AltBeacon δεν υποστηρίζεται από πολλούς κατασκευαστές beacons μέχρι στιγμής.

URIBeacon

Το πρωτόκολλο URIBeacon είναι διαφορετικό από τα πρωτόκολλα iBeacon και AltBeacon. Μία συσκευή URIBeacon, αντί να εκπέμπει κάποιον αναγνωριστικό κώδικα, εκπέμπει ένα URL το οποίο οι συσκευές μπορούν να χρησιμοποιήσουν αμέσως. Αξίζει να σημειωθεί ότι το URIBeacon είναι ένα Google project.

Eddystone

Το πρωτόκολλο Eddystone δημιουργήθηκε από την Google. Υποστηρίζει τρεις διαφορετικούς τύπους πακέτων: Eddystone-UID, Eddystone-URL, Eddystone-TLM. Το Eddystone-UID λειτουργεί με αρκετά όμοιο τρόπο με το iBeacon της Apple, ωστόσο υποστηρίζει παραπάνω τηλεμετρικά δεδομένα με το Eddystone-TLM. Η τηλεμετρική πληροφορία μεταδίδεται μαζί με τα δεδομένα UID. Οι πληροφορίες που μεταδίδονται συμπεριλαμβάνουν μέτρηση της τάσης της μπαταρίας, μέτρηση της θερμοκρασίας του beacon, τον αριθμό των πακέτων που αποστάλθηκαν από την τελευταία εκκίνηση, και το χρόνο λειτουργίας του beacon.

Estimote

Η Estimote,Inc. είναι μία τεχνολογική start-up που δημιουργεί analytics βασισμένα σε αισθητήρες και πλατφόρμες αλληλεπίδρασης. Η κύρια εστίαση της εταιρείας είναι τα καταστήματα φυσικού χώρου, καθώς πάνω από το 95% των συναλλαγών πραγματοποιούνται ακόμα σε αυτά. Παράλληλα πάνω από τους μισούς καταναλωτές που επισκέπτονται τα καταστήματα αυτά έχουν έξυπνα κινητά τηλέφωνα, ένας αριθμός που αυξάνεται συνεχώς.

Το όραμα της εταιρείας είναι ότι οι μελλοντικές εφαρμογές δεν θα εγκαθίστανται στα έξυπνα κινητά τηλέφωνα, αλλά στους διάφορους προορισμούς στον πραγματικό κόσμο όπως εστιατόρια, στάσεις λεωφορείων, μουσεία ή πάρκα. Οι φυσικές τοποθεσίες θα αλληλεπιδρούν ασταμάτητα και θα επικοινωνούν με τους ανθρώπους μέσω βολικών διαύλων όπως τα έξυπνα κινητά τηλέφωνα, οθόνες καταστημάτων και φουτουριστικές βιτρίνες.

Περιγραφή Προβλήματος

Η στιγματοθέτηση και η πλοήγηση σε εσωτερικούς χώρους παραμένει ένα ανοιχτό πρόβλημα. Ενώ γίνονται συνεχώς προσπάθειες, είναι εύκολο να παρατηρήσουμε ότι υπάρχουν ακόμα τεχνολογικοί περιορισμοί. Ωστόσο τα δεδομένα αλλάζουν καθημερινά τόσο με την προσθήκη νέων συσκευών όσο και με τη συνεχή βελτίωση των παλαιότερων. Σε αυτή τη διπλωματική αντιμετωπίζονται δύο προβλήματα, τα οποία παρότι συσχετίζονται επειδή και τα δύο ασχολούνται με τη θέση, είναι διακριτά και δημιουργούν διαφορετικές σχεδιαστικές προκλήσεις.

Στο κομμάτι της στιγματοθέτησης κεντρικό ζήτημα παραμένει η ακρίβεια και η εύρεση αποδοτικών μεθόδων για την ελάττωση του σφάλματος. Η ακρίβεια έχει πρωτίστως άρρηκτη σχέση με τις συσκευές beacons και την απόκριση αυτών σε πραγματικό χρόνο. Παράλληλα δεν υπάρχουν ευριστικές μέθοδοι που να έχουν ξεχωρίσει για την ακρίβειά υπολογισμού της θέσης. Συνεπώς παρατηρούνται ποικίλες υλοποιήσεις και δοκιμές διαφόρων τεχνικών. Η δυσκολία στην ακρίβεια αυξάνεται όταν η απαιτούμενη παρακολούθηση της έξυπνης κινητής συσκευής γίνεται κατά την κίνησή της σε γρήγορο χρόνο. Σε αυτή την περίπτωση δημιουργούνται νέα ζητήματα που αφορούν την ταχύτητα με την οποία κινείται ο χρήστης με τη συσκευή του και η γρήγορη ενημέρωση της θέσης του στην οθόνη της συσκευής.

Στο κομμάτι της πλοήγησης υπάρχουν αρκετές οδοί που μπορούν να ακολουθηθούν κάτι που εξαρτάται σε μεγάλο βαθμό από τον τρόπο της στιγματοθέτησης και την διαχείριση των δεδομένων. Η αποφυγή εμποδίων είναι ένα κεντρικό ζήτημα που απαιτεί χαρτογράφηση υψηλής ακρίβειας. Μία συχνή λύση είναι η αναπαράσταση με κάποιο bitmap ή κάποιο γράφο. Και εδώ δυσκολίες παρουσιάζονται στις υλοποιήσεις για κίνηση σε γρήγορο χρόνο, όπου θέλουμε να παρακολουθούμε τον χρήστη και να ανανεώνουμε την πληροφορία στην οθόνη του σχετικά με την προτεινόμενη διαδρομή. Επιπλέον πρέπει να εξετάζεται η απόκλιση από την τρέχουσα διαδρομή ώστε να γίνεται επαναπροσδιορισμός νέας.

Αντικείμενο Διπλωματικής

Σε αυτή τη διπλωματική θα σχεδιάσουμε μία εφαρμογή android που θα πληροί υψηλές προδιαγραφές σχεδιασμού, αξιοποιώντας σύγχρονες τεχνολογίες του οικοσυστήματος Android, και θα έχει προοπτικές επέκτασης. Θα επιχειρήσουμε να βελτιώσουμε την ακρίβεια της στιγματοθέτησης με τη χρήση ενός πλέγματος σημείων στο χώρο. Σε κάθε σημείο θα πάρουμε μετρήσεις RSSI από τα beacon και έτσι θα ορίσουμε το γνωστό για την εφαρμογή μας περιβάλλον. Λαμβάνοντας συνεχώς πακέτα από πέντε beacons θα δημιουργούμε ένα σύνολο τιμών RSSI που θα περιγράφει τη θέση μας στο χώρο, ο οποίος αναπαρίσταται από ένα σύστημα πέντε μεταβλητών που προκύπτουν από τις μετρήσεις από τα πέντε beacons. Έτσι δυναμικά θα συγκρίνουμε τις τρέχουσες τιμές που λαμβάνουμε στη συσκευή μας, με τις γνωστές τιμές RSSI στους κόμβους-σημεία του περιβάλλοντος, προσδιορίζοντας έτσι τη θέση της συσκευής ως προς τα γνωστά σημεία.

Το σενάριο που υλοποιούμε είναι ένα σενάριο άμεσης εκκένωσης του δωματίου, συνεπώς προτείνουμε πάντα το βέλτιστο μονοπάτι από τη θέση μας προς την έξοδο του χώρου. Σε μία μεγαλύτερη υλοποίηση θα μπορούσε να γίνει χαρτογράφηση συνεχόμενων δωματίων σε ένα κτίριο ώστε να ολοκληρώνεται η εκκένωση του κτιρίου, κάτι που θα μπορούσε να γίνει με το σχεδιασμό που περιγράφεται σε αυτή τη διπλωματική δεδομένης της παροχής περισσότερων beacons για την κάλυψη περισσότερων και μεγαλύτερων χώρων.

Σχεδιασμός Εφαρμογής

Σύνοψη Σχεδιασμού

Ο σχεδιασμός της εφαρμογής που περιγράφεται σε αυτή τη διπλωματική εργασία μπορεί να διακριθεί στα παρακάτω στάδια, τα οποία αποτέλεσαν και χρονολογικά τα βήματα της δημιουργίας. Πριν προχωρήσουμε σε αυτά, είναι πρέπον να αναφερθεί ότι μία αντίστοιχη υλοποίηση για έναν γνωστό μικρό χώρο, θα μπορούσε να γίνει με πολύ απλούστερο σχεδιασμό. Ωστόσο, η μελέτη που ακολουθεί έγινε με γνώμονα την επεκτασιμότητα και την καλή εικόνα του πρότζεκτ. Έτσι σχεδιάστηκαν και αξιοποιήθηκαν στοιχεία και τεχνολογίες που ίσως δεν είναι απαραίτητες σε αυτή τη μορφή της εφαρμογής αλλά καθιστούν εύκολη και ξεκάθαρη την μελλοντική ανάπτυξή της.

Μελέτη beacons και estimote SDK

Σε αυτό το αρχικό στάδιο γίνεται η απαραίτητη μελέτη και κατανόηση της λειτουργίας των beacons και του estimote SDK που παρέχεται από τον κατασκευαστή. Υλοποιήθηκαν όλα τα διαθέσιμα παραδείγματα που παρέχονται για εξοικείωση με τα beacons και το SDK τους. Αφότου ολοκληρώθηκε η διαδικασία, απομονώθηκαν οι μέθοδοι που επιστρέφουν χρήσιμες, για τη δική μας εφαρμογή, πληροφορίες. Ενδεικτικά χρησιμοποιήθηκαν μέθοδοι που επιστρέφουν αναγνωριστικές πληροφορίες για το κάθε beacon, όπως οι τιμές major & minor, καθώς και μέθοδοι που επιστρέφουν το RSSI, όπως η computeAccuracy.

Διαμόρφωση δεδομένων εισόδου

Σε αυτό το στάδιο γίνεται απομόνωση της πληροφορίας που θέλουμε από τα πακέτα των beacons και σχεδιασμός των οντοτήτων, σαν κλάσεις java, που θα περιγράφουν αυτή την πληροφορία. Έτσι προκύπτει αρχικά η κλάση Node η οποία θα ορίζει έναν κόμβο στο χώρο και θα εμπεριέχει τις τιμές του RSSI, όπως αυτές επιστρέφονται από την computeAccuracy από το κάθε ένα beacon. Με αυτό τον τρόπο ορίζεται στο σύστημά μας η θέση του κάθε αντικειμένου της κλάσης Node στο χώρο που περιγράφεται από τα RSSI των πέντε beacons. Είναι δηλαδή σαν να έχουμε ένα δημιουργήσει ένα σύστημα συντεταγμένων με τα πέντε beacons σε σταθερές θέσεις και την έξυπνη κινητή συσκευή να λαμβάνει τις τιμές RSSI και να προσδιορίζει τη θέση της με αυτό τον τρόπο.

Βάση Δεδομένων

Σε αυτό το στάδιο μελετάμε τις επιλογές μας για την αποθήκευση των δεδομένων μας. Απορρίπτοντας την επιλογή των “καρφωτών τιμών” προχωράμε σε σχεδιασμό μίας βάσης δεδομένων που θα περιέχει τους απαιτούμενους πίνακες για την αποθήκευση χρήσιμων δεδομένων της εφαρμογής. Έτσι επιλέγουμε την τεχνολογία της Room Database που συνιστάται για χρήση στο IDE μας, Android Studio. Αποφασίζουμε ότι, σε αυτό το στάδιο, αρκεί ένας πίνακας που θα περιγράφει τους το περιεχόμενο των κόμβων, στιγμιοτύπων της κλάσης Node, που αποθηκεύει τις μετρήσεις από τον κάθε κόμβο.

Επικοινωνία με τη Βάση Δεδομένων

Σε αυτό το στάδιο μελετάμε την Room Database και γίνεται λήψη αποφάσεων σχετικά με το αρχιτεκτονικό μοντέλο που θα περιγράφει το διαχωρισμό του backend από το frontend. Σχεδιάζουμε τα απαιτούμενα αντικείμενα (κλάσεις και διαπροσωπείες) για την επικοινωνία με τη βάση καθώς και του τρόπου που θα μεταδίδεται η πληροφορία στο UI. Έτσι καταλήγουμε σε σημαντικές οντότητες για την εφαρμογή μας, όπως οι διαπροσωπείες DAO (data access objects) που υλοποιούνται από τις αντίστοιχες κλάσεις repository. Τέλος σχεδιάζουμε την κλάση ViewModel που θα λειτουργεί σαν διαμεσολαβητής με το thread του UI και θα επιστρέφει σε αυτό μεθόδους των κλάσεων repository.

Οθόνες εφαρμογής

Έχοντας ολοκληρώσει ένα βασικό μέρος του σχεδιασμού, σε αυτό το στάδιο προχωράμε σε σχεδιασμό των οθονών της εφαρμογής (activities), ως κλάσεις java και ως xml αρχεία.

Αποφασίζουμε τον χάρτη της εφαρμογής και τα σενάρια χρήσης αυτής. Δηλαδή πως ο χρήστης πλοηγείται στην εφαρμογή και καταλήγει σε κάθε οθόνη.

Μετρήσεις στους κόμβους

Έχοντας πλέον σχεδιάσει που θα βρίσκονται τα δεδομένα εισόδου, προχωράμε σε σχεδιασμό των μεθόδων για τη διαχείριση και αποθήκευση των δεδομένων εισόδου σε κάθε κόμβο-σημείο του χάρτη που αναπαριστά τον χώρο μας. Έτσι σχεδιάζουμε μία νέα λειτουργία που σε μία νέα οθόνη θα μας δείχνει όλες τις μετρήσεις για όλους τους κόμβους σε μία λίστα και θα μπορούμε να προσθέσουμε νέα μέτρηση. Όταν προσθέτουμε νέα μέτρηση επιλέγουμε τον κόμβο για στον οποίο μετράμε και μία μέθοδος μαζεύει τις μετρήσεις από τα beacons σε εκείνο το σημείο και δημιουργεί ένα νέο στιγμιότυπο της κλάσης Node και το αποθηκεύει στον αντίστοιχο πίνακα της βάσης. Αφότου προστεθεί στη βάση, θα εμφανίζεται στη λίστα των μετρήσεων, αφού η λίστα κοιτάει στη βάση δεδομένων.

Στιγματοθέτηση

Αφότου έχουμε πληθύνει την βάση μας με αρκετές μετρήσεις σε διαφορετικούς κόμβους, σε αυτό το στάδιο, σχεδιάζουμε τις μεθόδους που αφορούν στον υπολογισμό της θέσης της συσκευής. Έτσι υπολογίζουμε την τρέχουσα θέση ως το σύνολο των RSSI των πέντε beacons. Στη συνέχεια υπολογίζουμε τις διαφορές της τρέχουσας θέσης με τις μετρήσεις που υπάρχουν στον πίνακα των Nodes στη βάση δεδομένων. Αυτό το ορίζουμε όμοια με τον τύπο της απόστασης σε ένα καρτεσιανό σύστημα μεταβλητών, δηλαδή ως τη ρίζα των διαφορών των μετρήσεων από κάθε beacon υψωμένων στο τετράγωνο. Από αυτές τις αποστάσεις κρατάμε τη μικρότερη και ορίζουμε το Node που της ανήκει ως τον κοντινότερο κόμβο στην τρέχουσα θέση.

Σχεδιασμός στίγματος στην οθόνη

Έχοντας υπολογίσει το id του κοντινότερου κόμβου, σε αυτό το στάδιο αντιμετωπίζουμε ένα νέο πρόβλημα. Αυτό είναι η σχεδίαση στην οθόνη, τόσο του στίγματος όσο και των κόμβων. Έτσι αρχικά γίνεται σχεδιασμός για την μετάδοση της πληροφορίας της θέσης από την τρέχουσα δραστηριότητα στην κλάση DrawView που περιέχει τον καμβά που θα ζωγραφίσει το στίγμα στην οθόνη. Στη συνέχεια, σχεδιάζουμε έναν νέο πίνακα στη βάση δεδομένων όπου θα κρατάμε τις συντεταγμένες των κόμβων στο χάρτη. Αυτό το κάνουμε κυρίως για καλύτερη οργάνωση του κώδικα, μιας και δεν αποτελεί πληροφορία που αλλάζει κατά την χρήση της εφαρμογής. Συνεπώς οι συντεταγμένες αρχικοποιούνται στη βάση κατά την εκκίνηση της εφαρμογής από την κλάση NodeRoomDatabase. Έχοντας συγκεντρώσει όλα τα δεδομένα που χρειαζόμαστε, προχωράμε στις απαραίτητες κλήσεις draw για σχεδίαση των κόμβων και έπειτα για σχεδίαση της θέσης. Να σημειωθεί ότι δεν περιμένουμε τη θέση των κόμβων να αλλάξει κατά την χρήση της εφαρμογής. Αντιθέτως η θέση αλλάζει δυναμικά και οφείλουμε να ανανεώνουμε τον καμβά σε κάθε αλλαγή.

Αλγόριθμος Dijkstra

Για τον υπολογισμό και τη χάραξη του συντομότερου μονοπατιού προς την έξοδο επιλέγουμε τον αλγόριθμο του Dijkstra. Αυτό είναι απαραίτητο επειδή οι αποστάσεις των κόμβων μας δεν είναι πάντα ίσες μεταξύ τους. Αυτή η σχεδιαστική παραδοχή, δίνει περισσότερες επιλογές για μελλοντικές επεκτάσεις της εφαρμογής. Για την υλοποίηση του Dijkstra ορίζουμε τις απαραίτητες κλάσεις που περιγράφουν αναπόσπαστες έννοιες όπως οι Vertex, Edge, Graph και σε μία διαφορετική κλάση προσθέτουμε τον αλγόριθμο που τις αξιοποιεί. Έτσι καταλήγουμε σε μεθόδους που, αρχικά, υπολογίζουν όλες τις αποστάσεις από τον τρέχοντα κόμβο και στη συνέχεια επιστρέφουν το συντομότερο μονοπάτι.

Σχεδιασμός μονοπατιού στην οθόνη

Αυτή η πληροφορία μεταδίδεται στη συνέχεια στην κλάση DrawView όπου γίνεται η σχεδίαση στον καμβά και, κατ' επέκταση, στην οθόνη. Όμοια με την σχεδίαση της θέσης, το μονοπάτι αλλάζει δυναμικά κατά τη χρήση και οδηγεί σε νέα σχεδίαση στον καμβά.

Εργαλεία και Γλώσσες Προγραμματισμού

Πριν προχωρήσουμε στην αναλυτική περιγραφή κάθε τμήματος του σχεδιασμού γίνεται σύντομη περιγραφή των εργαλείων και των γλωσσών προγραμματισμού που χρησιμοποιήθηκαν.

Android Studio IDE

Για την υλοποίηση της εφαρμογής που περιγράφεται σε αυτή τη διπλωματική χρησιμοποιήθηκε το προγραμματιστικό περιβάλλον Android Studio.

Το Android Studio είναι το επίσημο προγραμματιστικό περιβάλλον για το λειτουργικό Android της Google και βασίζεται στην περιβάλλον IntelliJ της JetBrains. Η πλατφόρμα είναι διαθέσιμη για Windows, macOS και Linux και αποτελεί αντικαταστάτη της πλατφόρμας Eclipse Android Development Tools (ADT) η οποία νωρίτερα ήταν το πιο δημοφιλές IDE για ανάπτυξη εφαρμογών Android.

Github

Με σκοπό την καλύτερη οργάνωση της διπλωματικής χρησιμοποιήθηκε το Github για έλεγχο των εκδόσεων του κώδικα αλλά και λόγω της ανάγκης εργασίας από σταθερό και φορητό υπολογιστή.

Το Github είναι μία διαδικτυακή υπηρεσία ελέγχου εκδόσεων κώδικα με χρήση Git. Παρέχει όλες τις λειτουργίες κατανεμημένου ελέγχου εκδόσεων και διαχείρισης πηγαίου κώδικα που διαθέτει το Git, ενώ παράλληλα προσθέτει κάποια ακόμη. Παρέχει έλεγχο πρόσβασης και αρκετές λειτουργίες που βοηθούν την συνεργασία ανάμεσα σε προγραμματιστές όπως παρακολούθηση bug, αίτηση νέων λειτουργιών, οργάνωση εργασιών και δημιουργία σελίδας wiki για κάθε πρότζεκτ.

Trello

Για την καλύτερη οργάνωση και με σκοπό την τήρησης του χρονοδιαγράμματος της διπλωματικής χρησιμοποιήθηκε το Trello.

Το Trello είναι μία διαδικτυακή εφαρμογή για διαχείριση πρότζεκτ ανεπτυγμένη αρχικά από την εταιρεία Fog Creek Software το 2011, ενώ στη συνέχεια ανεξαρτητοποιήθηκε ως ξεχωριστή εταιρεία το 2014. Τελικά πουλήθηκε στην εταιρεία Atlassian το Γενάρη του 2017.

Java

Η προγραμματιστική γλώσσα που χαρακτηρίζει το μεγαλύτερο μέρος της εφαρμογής είναι η Java. Η Java εξυπηρέτησε τον αντικειμενοστραφή σχεδιασμό της εφαρμογής ενώ είναι η γλώσσα γραφής του estimote SDK και χρησιμοποιείται για ανάπτυξη εφαρμογών στο Android Studio.

Η Java είναι μία προγραμματιστική γλώσσα γενικού σκοπού που είναι μη-σειριακή, βασισμένη σε κλάσεις, αντικειμενοστραφής και συγκεκριμένα σχεδιασμένη για να έχει τις ελάχιστες δυνατές εξαρτήσεις κατά την υλοποίηση. Κεντρική λογική είναι το “write once, run anywhere” που αποδίδεται σημαίνει ότι οι εφαρμογές java αφότου συνταχθούν από τον προγραμματιστή δεν χρειάζεται να συνταχθούν ξανά πριν τη χρήση σε οποιαδήποτε πλατφόρμα.

Xml

Η γλώσσα xml χρησιμοποιήθηκε για τον σχεδιασμό των οθονών στο Android Studio και συγκεκριμένα για την περιγραφή της θέσης κάθε στοιχείου που εμφανίζεται σε κάθε οθόνη.

Η xml (Extensible Markup Language) είναι μία Markup γλώσσα η οποία ορίζει ένα σύνολο κανόνων για την κωδικοποίηση αρχείων κειμένου σε μία μορφή που είναι ευανάγνωστη τόσο από ανθρώπους όσο και από μηχανές. Οι σχεδιαστικοί στόχοι της xml δίνουν έμφαση στην απλότητα, στη γενικότητα και στην ευχρηστία στο διαδίκτυο.

Εμπειρία Χρήσης

Σε αυτό το κεφάλαιο θα αναφερθούμε στη δομή της εφαρμογής μας από την πλευρά του χρήστη, δηλαδή τις οθόνες στις οποίες μπορεί να πλοηγηθεί ο χρήστης. Θα παρουσιάσουμε τα σενάρια χρήσης της εφαρμογής και στη συνέχεια τον χάρτη οθονών της εφαρμογής. Τέλος θα γίνει περιγραφή της κάθε οθόνης ξεχωριστά.

Σενάρια Χρήσης

Ο τρόπος χρήσης της εφαρμογής καθοδηγείται από τον σχεδιασμό της. Έτσι σε αυτή την διπλωματική εργασία αναπτύσσουμε μία εφαρμογή που απαιτεί αρχικά κάποια προεργασία από τον προγραμματιστή και από εκεί και πέρα, ο χρήστης πρέπει να ολοκληρώσει το σετάρισμα έτσι ώστε η εφαρμογή να μπορέσει να τον καθοδηγήσει στο χώρο.

Ο προγραμματιστής έχει περάσει στην εφαρμογή την κάτοψη του χώρου και έχει ορίσει το πλέγμα των κόμβων πάνω στην κάτοψη. Επιπλέον έχει ενημερώσει τη βάση δεδομένων με τις συντεταγμένες αυτών των κόμβων.

Ο χρήστης, με αυτόν τον τρόπο, λαμβάνει μία εφαρμογή προετοιμασμένη για τον χώρο που θέλει να τη χρησιμοποιήσει. Από την αρχική οθόνη του δίνονται δύο επιλογές, οι οποίες συνοψίζουν τα δύο σενάρια χρήσης.

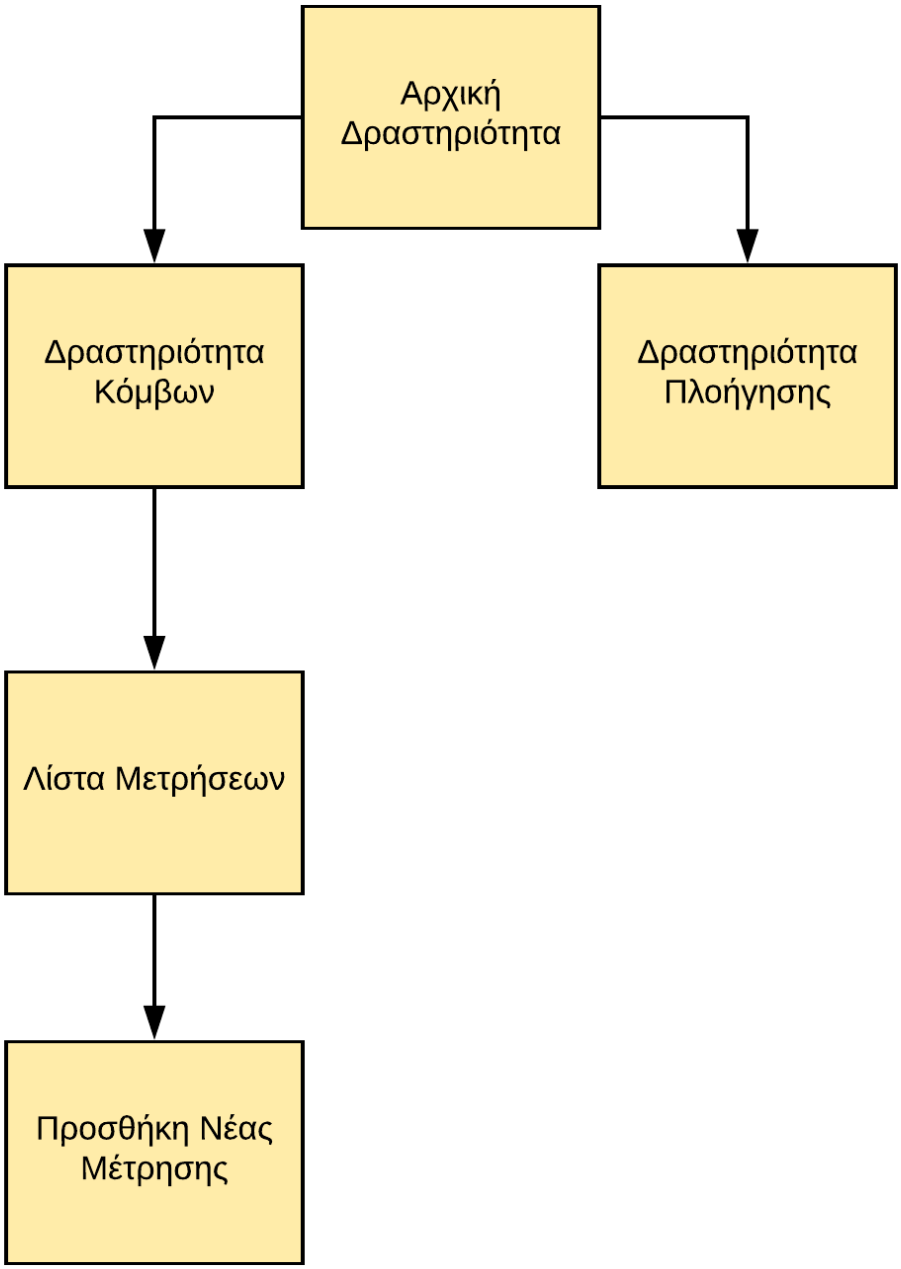
Πρώτα ο χρήστης πρέπει να πραγματοποιήσει μετρήσεις στους κόμβους, κάτι που μπορεί να καταφέρει μέσω των οθονών που αφορούν στις μετρήσεις. Από αυτές τις οθόνες θα μπορέσει να δει την κάτοψη του χώρου και τους κόμβους πάνω της. Στη συνέχεια θα μπορεί να δει τις υπάρχουσες μετρήσεις στους κόμβους αλλά και να πραγματοποιήσει νέα μέτρηση σε νέο κόμβο. Έτσι ο χρήστης επιλέγει τον κόμβο στον οποίο θέλει να πραγματοποιήσει νέα μέτρηση και μετακινείται στην αντίστοιχη θέση μαζί με την έξυπνη κινητή συσκευή του. Κατόπιν πραγματοποιεί τη μέτρηση και αυτή αποθηκεύεται στη βάση δεδομένων. Όμοια πραγματοποιεί μετρήσεις για όλους τους κόμβους στο χώρο και έτσι ολοκληρώνει την προετοιμασία της εφαρμογής.

Έχοντας ολοκληρώσει την προετοιμασία της εφαρμογής, ο χρήστης πλέον εκκινεί την εφαρμογή με σκοπό της πλοήγηση προς την έξοδο του δωματίου, όπως θα γινόταν σε ένα σενάριο έκτακτης εκκένωσης χώρου. Αυτό γίνεται μέσω της οθόνης της δραστηριότητας πλοήγησης.

Σε μετέπειτα επέκταση της εφαρμογής θα ήταν σκόπιμο να υπάρχουν διαφορετικοί ρόλοι χρηστών. Έτσι θα έχουμε ένα “Διαχειριστή” ο οποίος θα ευθύνεται για την προετοιμασία της εφαρμογής και άλλους χρήστες ως “Επισκέπτες” που χρησιμοποιούν την εφαρμογή αποκλειστικά για την πλοήγηση.

Χάρτης Εφαρμογής

Αρχικά παρουσιάζεται μία οπτική απεικόνιση των οθονών σαν ένα διάγραμμα ροής. Εδώ μπορούμε να δούμε με ευκολία την εφαρμογή των σεναρίων χρήσης και την ροή χρήσης της εφαρμογής.



Αρχική Δραστηριότητα

Η αρχική δραστηριότητα θα περιλαμβάνει ένα εισαγωγικό μήνυμα για τον χρήστη. Παράλληλα θα παρέχει δύο κουμπιά τα οποία θα οδηγούν στην οθόνη “Δραστηριότητα Κόμβων” και στην οθόνη “Δραστηριότητα Πλοήγησης”.

Δραστηριότητα Κόμβων

Η δραστηριότητα κόμβων θα παρουσιάζει στον χρήστη-διαχειριστή την κάτοψη του χώρου και πάνω σε αυτή, τους κόμβους στις συντεταγμένες οθόνης που έχει ορίσει ο προγραμματιστής στη βάση δεδομένων. Παράλληλα θα παρέχει κουμπί για μετάβαση στην οθόνη “Λίστα Μετρήσεων”.

Λίστα Μετρήσεων

Αυτή η οθόνη θα παρουσιάζει λίστα των μετρήσεων όπως αυτές έχουν αποθηκευτεί στη βάση δεδομένων προηγουμένως. Παράλληλα θα παρέχει κουμπί για την προσθήκη νέας μέτρησης με μεταφορά στην οθόνη “Προσθήκη Νέας Μέτρησης”.

Προσθήκη Νέας Μέτρησης

Αυτή την οθόνη θα επιτρέπει στο χρήστη να επιλέξει έναν κόμβο από μία λίστα και στη συνέχεια μέσω ενός κουμπιού θα καλεί μεθόδους για την πραγματοποίηση της μέτρησης στη συγκεκριμένη θέση και την αποθήκευση αυτής της μέτρησης στη βάση δεδομένων.

Δραστηριότητα Πλοήγησης

Τέλος, η δραστηριότητα πλοήγησης αξιοποιεί όλη την προετοιμασία της εφαρμογής και χρησιμοποιεί τρέχουσες μετρήσεις από τα beacons για να προσδιορίσει τη θέση του χρήστη στο χώρο. Έπειτα πάνω στην κάτοψη του χώρου σχεδιάζει δυναμικά τη θέση του χρήστη καθώς αυτός κινείται και επιπλέον σχεδιάζει το συντομότερο μονοπάτι προς την έξοδο του χώρου όπως αυτό υπολογίζεται από τον αλγόριθμο του Dijkstra.

Εισαγωγή Δεδομένων

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τον τρόπο με τον οποίο η εφαρμογή δέχεται τα δεδομένα εισόδου από τα beacons στον περιβάλλοντα χώρο. Για να επιτευχθεί αυτός ο σκοπός θα δούμε αρχικά πως γίνεται η επικοινωνία με τα beacons μέσω του Estimote SDK σε θεωρητικό επίπεδο, και στη συνέχεια θα παρουσιάσουμε τις δομές δεδομένων που σχεδιάζουμε για την απεικόνιση αυτών των δεδομένων στην εφαρμογή μας.

Estimote SDK

Αρχικά είναι απαραίτητο να μιλήσουμε για τα αναγνωριστικά χαρακτηριστικά ενός beacon, τα οποία είναι τρία:

- Το UUID, ένας αλφαριθμητικός κωδικός οποίος χρησιμοποιείται για την ομαδοποίηση beacons.
- Το Major, ένας αριθμητικός κωδικός που χρησιμοποιείται σαν “προσωπικό” στοιχείο του beacon.
- Και το Minor, ένας ακόμη αριθμητικός κωδικός όμοιος τους Major.

Οι παραπάνω τιμές στο σύνολό τους απαρτίζουν το ID του beacon. Από τις εργοστασιακές ρυθμίσεις, τα beacons έρχονται με έναν μοναδικό συνδυασμό. Βεβαίως αυτές οι τιμές μπορούν να αλλάξουν από τον ιδιοκτήτη της συσκευής, κάτι που μερικές φορές είναι απαραίτητο αλλά παράλληλη μπορεί να οδηγήσει σε ταυτονυμία δύο συσκευών και συνεπώς την σύγχυση κάποιου λογισμικού.

Έχοντας κατανοήσει τα παραπάνω, ξεκινάμε να μελετάμε τι θα χρειαστούμε ώστε να μπορεί η εφαρμογή μας να αναγνωρίσει τα beacons που διαθέτουμε. Είναι σημαντικό να σημειωθεί ότι δεν αρκεί στις περισσότερες περιπτώσεις να μπορεί μία εφαρμογή να εντοπίσει γενικά beacons. Συχνά είναι απαραίτητο να ξέρει η εφαρμογή ποιο beacon συναντά κάθε φορά.

Το Estimote SDK έχει τις μεθόδους και τις οντότητες που θα εκτελέσουν το έργο για εμάς.

- Αρχικά θα ορίσουμε μία περιοχή ανίχνευσης ως ένα στιγμιότυπο της κλάσης BeaconRegion. Σε αυτήν θα ορίσουμε το UUID της περιοχής να ισούται με το “B9407F30-F5F8-466E-AFF9-25556B57FE6D", ενώ ορίζουμε ως null τις τιμές Major και Minor. Αυτό σημαίνει ότι η περιοχή μας θα αναγνωρίζει όλα τα beacons με το παραπάνω UUID ανεξαρτήτως των τιμών Major και Minor που διαθέτουν. Θα μπορούσαμε αν θέλαμε να γίνουμε πιο συγκεκριμένοι και να ορίσουμε και μία τιμή Major, την οποία θα φροντίσουμε να έχουν όλες οι συσκευές μας, αλλά στην παρούσα εφαρμογή κρίνεται περιττό.
- Επίσης στην εφαρμογή μας θα χρειαστούμε ένα στιγμιότυπο της κλάσης BeaconManager. Ο beacon manager είναι μία οντότητα που διαχειρίζεται διάφορες λειτουργίες του Estimote SDK, αλλά στην περίπτωσή μας τον χρειαζόμαστε για την πιο βασική. Θα καλέσουμε την μέθοδό του “setRangingListener”, η οποία αρχικοποιείται με ένα στιγμιότυπο της κλάσης beaconManager.BeaconRangingListener.
- Εκεί ορίζουμε την πιο σημαντική για εμάς μέθοδο, την onBeaconsDiscovered, η οποία σαν όρισμα θα πάρει την περιοχή που προσδιορίσαμε νωρίτερα. Αυτή η μέθοδος θα καλείται κάθε ένα δευτερόλεπτο, συχνότητα ορισμένη από το SDK. Κάθε φορά που η εφαρμογή μας θα εντοπίζει ένα τουλάχιστον beacon, η μέθοδος αυτή θα μας επιστρέφει μία λίστα από τα beacons που εντόπισε και μάλιστα θα τα έχει ταξινομήσει με κριτήριο την απόσταση.

Έτσι καταλήγουμε με μία λίστα από αντικείμενα της κλάσης Beacon τα οποία θα είναι κάποια ή όλα από τα beacons μας.

Διαχείριση Δεδομένων

Από τα αντικείμενα της λίστας που επιστρέφει συνεχώς η OnBeaconsDiscovered μας ενδιαφέρουν δύο πράγματα.

- Ο κωδικός Minor τον οποίο μπορούμε να πάρουμε με την μέθοδο getMinor(). Συγκρίνοντας τον κωδικό του κάθε αντικειμένου με τους γνωστούς κωδικούς των beacons μας μπορούμε να προσδιορίσουμε σε ποιον αντιστοιχεί το συγκεκριμένο αντικείμενο.
- Το αποτέλεσμα της μεθόδου computeAccuracy() με όρισμα το αντικείμενο. Αυτή η μέθοδος επιστρέφει μία εκτίμηση της απόστασης του χρήστη από το beacon που αντιστοιχεί στο αντικείμενο.

Έχοντας αυτά τα δύο δεδομένα μπορούμε να δημιουργήσουμε ένα νέο στιγμιότυπο της κλάσης Node. Η κλάση Node, όπως αναφέραμε προηγουμένως στην σύνοψη, περιέχει μετρήσεις που λαμβάνει ένας συγκεκριμένος κόμβος από τα beacons στον περιβάλλοντα χώρο. Οπότε το μόνο που απομένει είναι βάλουμε τα αποτελέσματα των μεθόδων comuteAccuracy στην προκαθορισμένη επιθυμητή σειρά των beacons, χρησιμοποιώντας τους κωδικούς Minor.

Βάση Δεδομένων

Σε αυτό το κεφάλαιο θα αναφερθούμε στο σχεδιασμό της βάσης δεδομένων που χρησιμοποιούμε σε αυτή την διπλωματική εργασία. Θα μιλήσουμε για την Room Database που θα υλοποιήσουμε μέσω της Room Persistence Library, και παράλληλα θα μιλήσουμε για τις οντότητες που θα χρησιμοποιήσουμε για να έχουμε μία σωστή επικοινωνία με τη βάση δεδομένων.

Room Database

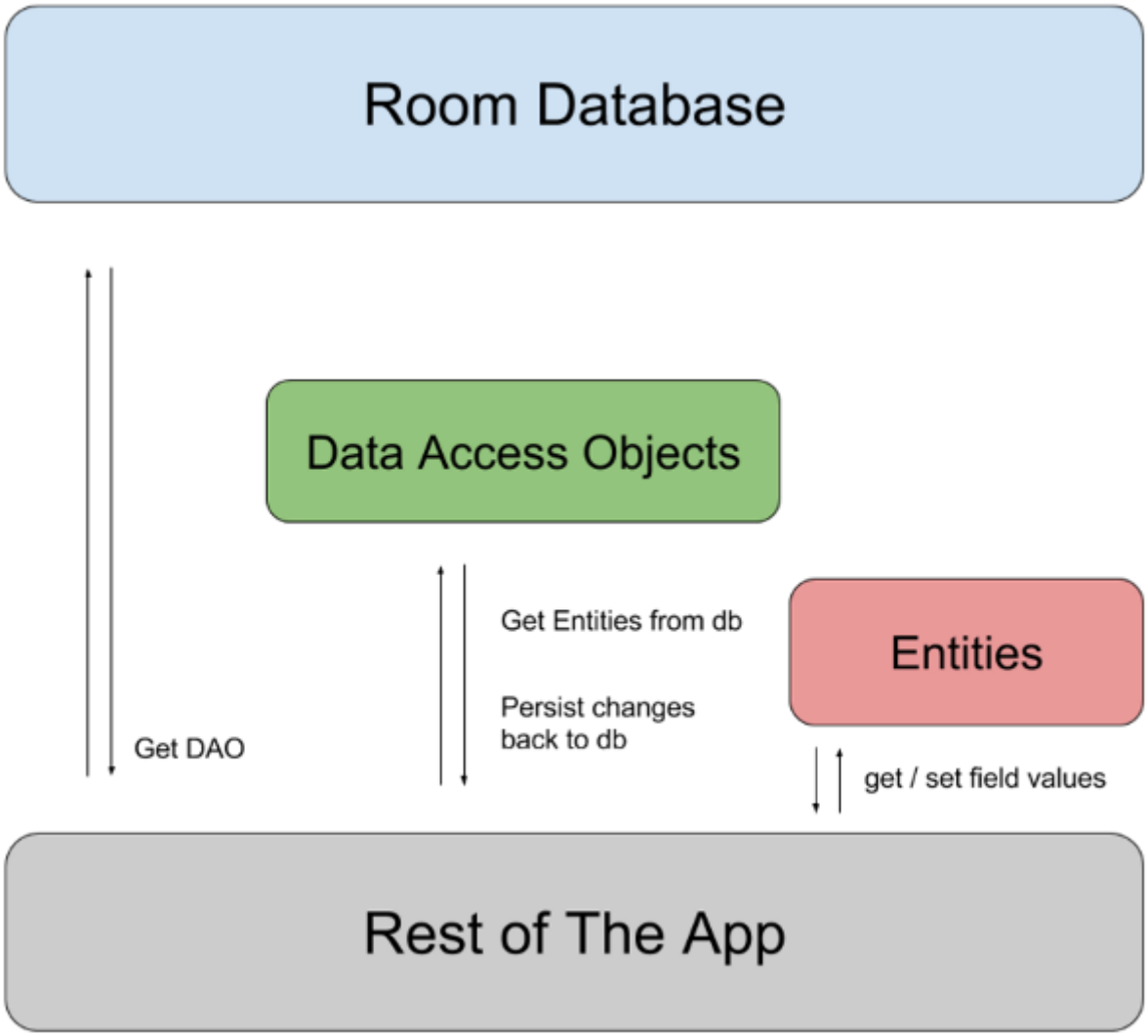
Η Room Persistence Library είναι μία βιβλιοθήκη της Java που χρησιμοποιείται πολύ συχνά σε εφαρμογές Android. Η Room παρέχει ένα αφηρημένο στρώμα πάνω από την SQLite ώστε να επιτρέπει την εύκολη πρόσβαση στη βάση δεδομένων διατηρώντας όλες τις δυνατότητες της SQLite.

Εφαρμογές με μη τετριμμένες ποσότητες από δομημένα δεδομένα μπορούν να επωφεληθούν σημαντικά από την τοπική αποθήκευση δεδομένων. Η πιο συχνή περίπτωση χρήσης είναι η αποθήκευση σχετικών δεδομένων στην cache. Με αυτόν τον τρόπο, όταν η συσκευή δεν μπορεί να έχει πρόσβαση στο διαδίκτυο, ο χρήστης μπορεί να έχει πρόσβαση στα δεδομένα που είναι αποθηκευμένα στη βάση δεδομένων. Οποιεσδήποτε αλλαγές πραγματοποιηθούν από τον χρήστη θα συγχρονιστούν αργότερα στον εξυπηρετητή όταν η συσκευή αποκτήσει ξανά πρόσβαση στο διαδίκτυο.

Υπάρχουν τρία κύρια μέρη στην Room:

- Βάση Δεδομένων: Περιέχει τον κάτοχο της βάσης δεδομένων και υπηρετεί ως το κύριο σημείο πρόσβασης για την σύνδεση με τα αποθηκευμένα σχεσιακά δεδομένα της εφαρμογής. Η κλάση που σημειώνεται με @Database πρέπει να πληροί τις παρακάτω προϋποθέσεις:
    - Πρέπει να είναι μία αφηρημένη κλάση που θα επεκτείνει την κλάση RoomDatabase.
    - Πρέπει να περιλαμβάνει τη λίστα από τις Οντότητες που σχετίζονται με τη βάση δεδομένων μέσα στην σημείωση.
    - Πρέπει να περιέχει μία αφηρημένη μέθοδο με μηδέν παραμέτρους που θα επιστρέφει τις κλάσεις που είναι σημειωμένες με @Dao.
- Οντότητα: Αντιπροσωπεύει έναν πίνακα μέσα στη βάση δεδομένων.
- DAO: Περιέχει τις μεθόδους που χρησιμοποιούνται για την πρόσβαση στη βάση δεδομένων.

Παρακάτω παρέχεται ένα διάγραμμα που περιγράφει τη βασική δομή μίας υλοποίησης της Room.



Πριν προχωρήσουμε στην περιγραφή των επιμέρους οντοτήτων, να αναφερθεί ότι παρότι η λογική των κλάσεων στην Java ωθεί τους προγραμματιστές στην δημιουργία πολλών στιγμιοτύπων των κλάσεων, όταν υλοποιείται μία κλάση με σημείωση @Database που επεκτείνει την κλάση RoomDatabase θέλουμε να είναι μοναδική για την εφαρμογή μας. Αυτό στην δική μας υλοποίηση το καταφέρνουμε εφαρμόζοντας το μοντέλο Singleton, μία λογική που δημιουργεί το πρώτο στιγμιότυπο μίας κλάσης και στη συνέχεια μπλοκάρει τη δημιουργία νέων στιγμιοτύπων.

Data Access Objects

Όπως αναφέρθηκε και νωρίτερα, τα αντικείμενα πρόσβασης δεδομένων, γνωστά ως DAO, επιτρέπουν την σωστή πρόσβαση στα δεδομένα της εφαρμογής μας όταν χρησιμοποιούμε την βιβλιοθήκη Room. Υλοποιώντας την πρόσβαση στη βάση δεδομένων με DAO αντί απευθείας με queries, καταφέρνουμε να διαχωρίσουμε διαφορετικά τμήματα της αρχιτεκτονικής της βάσης δεδομένων.

Ένα DAO μπορεί να είναι είτε μία διαπροσωπεία είτε μία αφηρημένη κλάση. Αν είναι αφηρημένη κλάση, μπορεί να έχει έναν κατασκευαστή που θα παίρνει ένα στιγμιότυπο της κλάσης RoomDatabase ως το μοναδικό του όρισμα. Η Room δημιουργεί κάθε κλάση/διαπροσωπεία DAO κατά την διάρκεια της σύνταξης της εφαρμογής.

Repositories

Τέλος θα αναφερθούμε στα Repositories, ένα κομμάτι του αρχιτεκτονικού μοντέλου της βάσης δεδομένων μας που δεν είναι απαραίτητο για την Room.

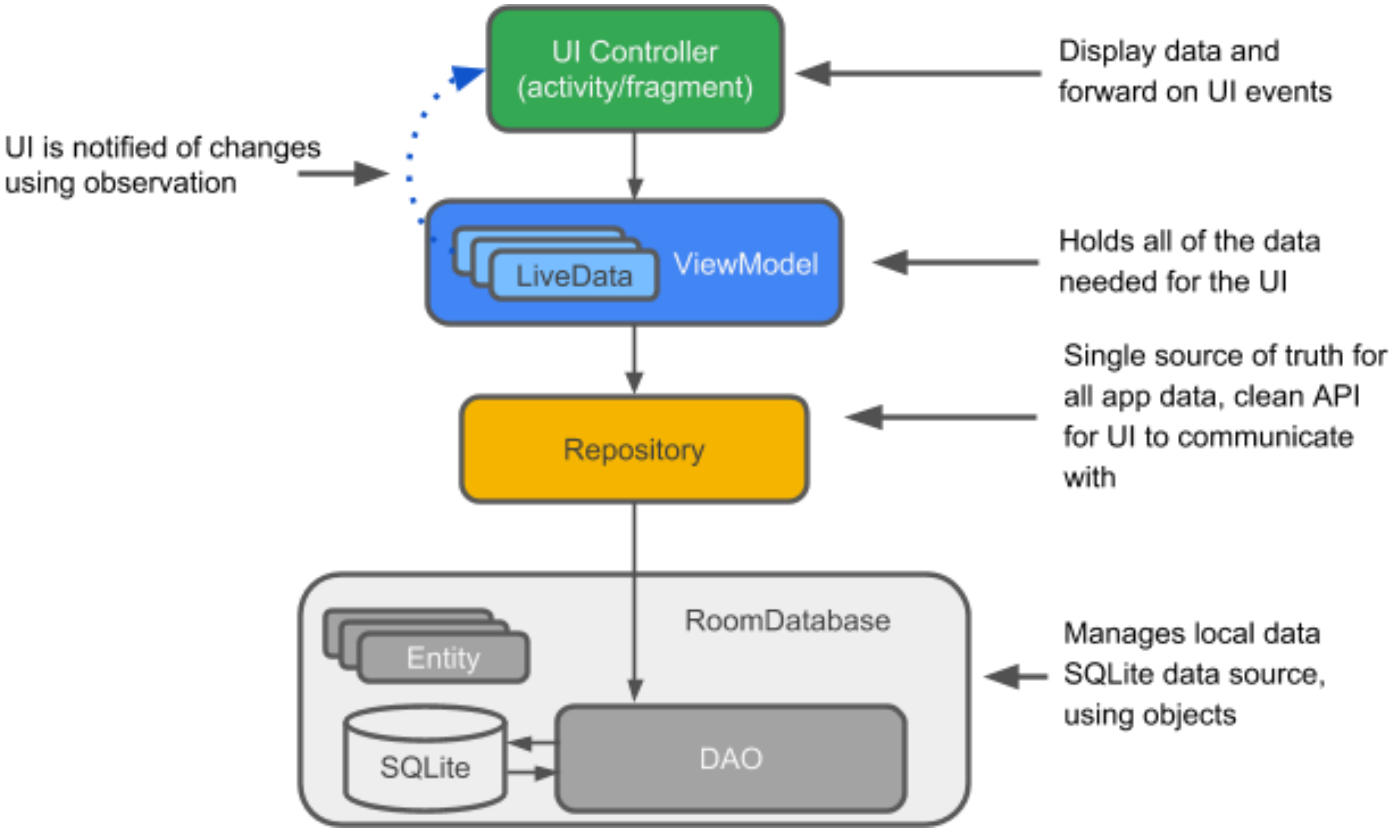
Τα Repositories χρησιμοποιούνται σαν ένα επιπλέον στρώμα μετά τα DAO, δηλαδή ανάμεσα στη βάση δεδομένων και την εφαρμογή μας. Ο λόγος που τα καθιστά απαραίτητα σε πολλές εφαρμογές είναι η ανάγκη για πολλαπλές πηγές δεδομένων. Για παράδειγμα αν πέρα από τη βάση που σχεδιάζουμε με τη βιβλιοθήκη Room, έχουμε αργότερα ανάγκη για παράλληλη λήψη όμοιων δεδομένων από κάποιο web service, θα μπορούμε να προσθέσουμε τη λογική λήψης δεδομένων του web service στο αντίστοιχο repository, χωρίς αυτό να επηρεάσει την υλοποίηση που θα έχει γίνει για την Room.

Συνεπώς, στη συγκεκριμένη διπλωματική εργασία, σχεδιάζουμε τα repositories σαν μέρος του αρχιτεκτονικού μας μοντέλου με κύριο γνώμονα την επεκτασιμότητα και όχι κάποια άμεση ανάγκη, μιας και τα δεδομένα μας θα προέρχονται αποκλειστικά από τη βάση δεδομένων της Room.

Αρχιτεκτονικό Μοντέλο

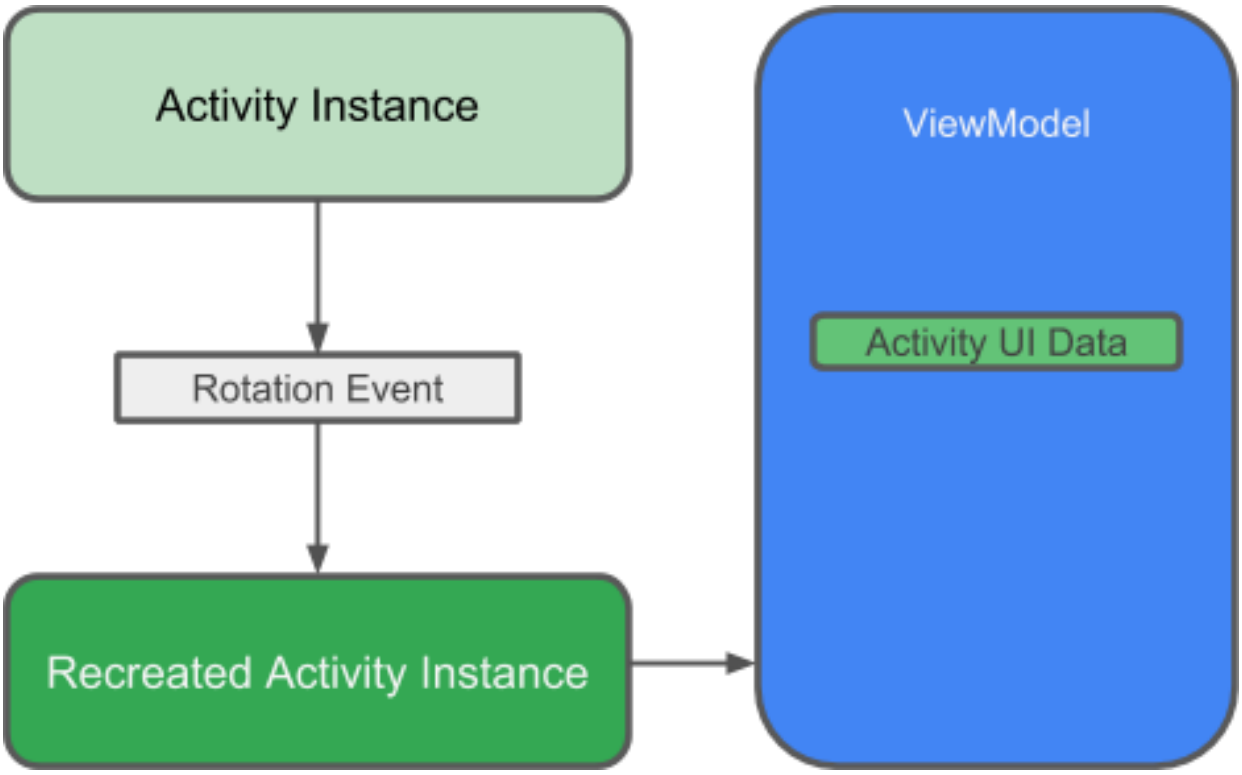
Σε αυτό το κεφάλαιο θα δούμε το αρχιτεκτονικό μοντέλο που περιγράφει τη δομή της βάσης μας αλλά και την επικοινωνία της με την εφαρμογή. Επιπλέον θα σχεδιάσουμε το ViewModel της εφαρμογής, μία οντότητα που επιτρέπει την αδιάκοπη επικοινωνία του UI με τα δεδομένα, όπως αυτά έρχονται από τη βάση δεδομένων μέσω της λογικής που ορίσαμε προηγουμένως (DAOs & Repositories).

Αρχικά ας δούμε μία οπτικοποίηση της δομής που ακολουθούμε, όπως αυτή παρουσιάζεται στο παρακάτω διάγραμμα ροής:



ViewModel

Το ViewModel, σαν τμήμα του Android, δημιουργήθηκε ως λύση σε ένα πρόβλημα, όπου τα τρέχοντα δεδομένα καταστρεφόντουσαν κατά την περιστροφή της οθόνης της συσκευής. Αυτό έχει να κάνει με τον κύκλο ζωής των δραστηριοτήτων μίας εφαρμογής Android, καθώς το λειτουργικό “σκοτώνει” και ξαναδημιουργεί κάθε δραστηριότητα όταν ο χρήστης πλοηγείται από τη μία εφαρμογή στην άλλη. Στο παρακάτω διάγραμμα ροής βλέπουμε πώς το τμήμα ViewModel λύνει το παραπάνω πρόβλημα:



Ο ρόλος του ViewModel είναι να παρέχει δεδομένα στο UI και να επιβιώνει όταν γίνονται αλλαγές στις ρυθμίσεις του Android. Το ViewModel λειτουργεί σαν ένα κέντρο επικοινωνίας μεταξύ των repositories και του UI. Επιπλέον μπορεί να εξυπηρετήσει τη μεταφορά δεδομένων μεταξύ δραστηριοτήτων.

Το ViewModel είναι μέρος της βιβλιοθήκης Lifecycle, δηλαδή του κύκλου ζωής των δραστηριοτήτων και γι αυτό κατά την υλοποίησή της, η κλάση ViewModel επεκτείνει πάντοτε την κλάση AndroidViewModel. Έτσι ενα ViewModel κρατάει τα δεδομένα της εφαρμογής που είναι απαραίτητα για το UI με έναν τρόπο που λαμβάνει υπόψη τον κύκλο ζωής και επιβιώνει όλες τις πιθανές αλλαγές στις ρυθμίσεις μίας συσκευής Android, όπως η περιστροφή οθόνης.

Τέλος, ο διαχωρισμός των δεδομένων του UI από τις κλάσεις των δραστηριοτήτων επιτρέπει στην ακολούθηση της αρχής της μοναδικής ευθύνης: Οι δραστηριότητες της εφαρμογής είναι υπεύθυνες για την σχεδίαση των δεδομένων στην οθόνη, ενώ το ViewModel αναλαμβάνει να κρατάει και να επεξεργάζεται όλα τα δεδομένα που είναι απαραίτητα για το UI.

LiveData

Ολοκληρώνοντας το κεφάλαιο της βάσης δεδομένων θα αναφερθούμε σε μία ακόμα κλάση της βιβλιοθήκης Lifecycle, την LiveData.

Όταν τα δεδομένα αλλάζουν κατά τη χρήση του εφαρμογής, συνήθως απαιτείται η πραγματοποίηση κάποιας ενέργειας, όπως η παρουσίαση νέων δεδομένων στο UI. Αυτό σημαίνει ότι πρέπει να παρακολουθούμε τα δεδομένα έτσι ώστε, όταν αλλάξουν, να εκτελέσουμε την επιθυμητή ενέργεια. Ανάλογα με το πως αποθηκεύονται τα δεδομένα της εφαρμογής, το παραπάνω πρόβλημα δεν έχει πάντα εύκολη λύση. Η παρακολούθηση αλλαγών σε δεδομένα που ανήκουν σε διαφορετικά τμήματα της εφαρμογής μπορεί να δημιουργήσει μεγάλα μονοπάτια εξάρτησης μεταξύ των τμημάτων και αυτό δυσκολεύει σε μεγάλο βαθμό τις δοκιμές και την επίλυση προβλημάτων.

Αυτό το πρόβλημα λύνει η κλάση LiveData της βιβλιοθήκης Lifecycle, για παρακολούθηση δεδομένων. Σε συνδυασμό με τη βιβλιοθήκη Room, αρκεί να επιστρέφουμε δεδομένα τύπου LiveData και η Room θα δημιουργήσει τον απαιτούμενο κώδικα.

Έχοντας λοιπόν επιλέξει τα δεδομένα που θέλουμε να έχουμε ως LiveData, αρκεί να προσθέσουμε έναν Observer στην δραστηριότητα που θέλουμε και να ορίσουμε ποια data θα παρακαλουθεί. Έτσι όταν αλλάξουν τα δεδομένα, μέσω της μεθόδου onChanged() του Observer, μπορούμε να εκτελούμε τις όποιες ενέργειες επιθυμούμε και έπειτα να ανανεώνουμε το UI.

Μετρήσεις Κόμβων

Σε αυτό το κεφάλαιο θα αναφερθούμε στη λογική του πλέγματος κόμβων και θα περιγράψουμε τον τρόπο με τον οποίο ολοκληρώνονται οι μετρήσεις στους κόμβους. Τέλος θα σχολιάσουμε την αξιοπιστία και την ακρίβεια των μετρήσεων.

Πλέγμα Κόμβων

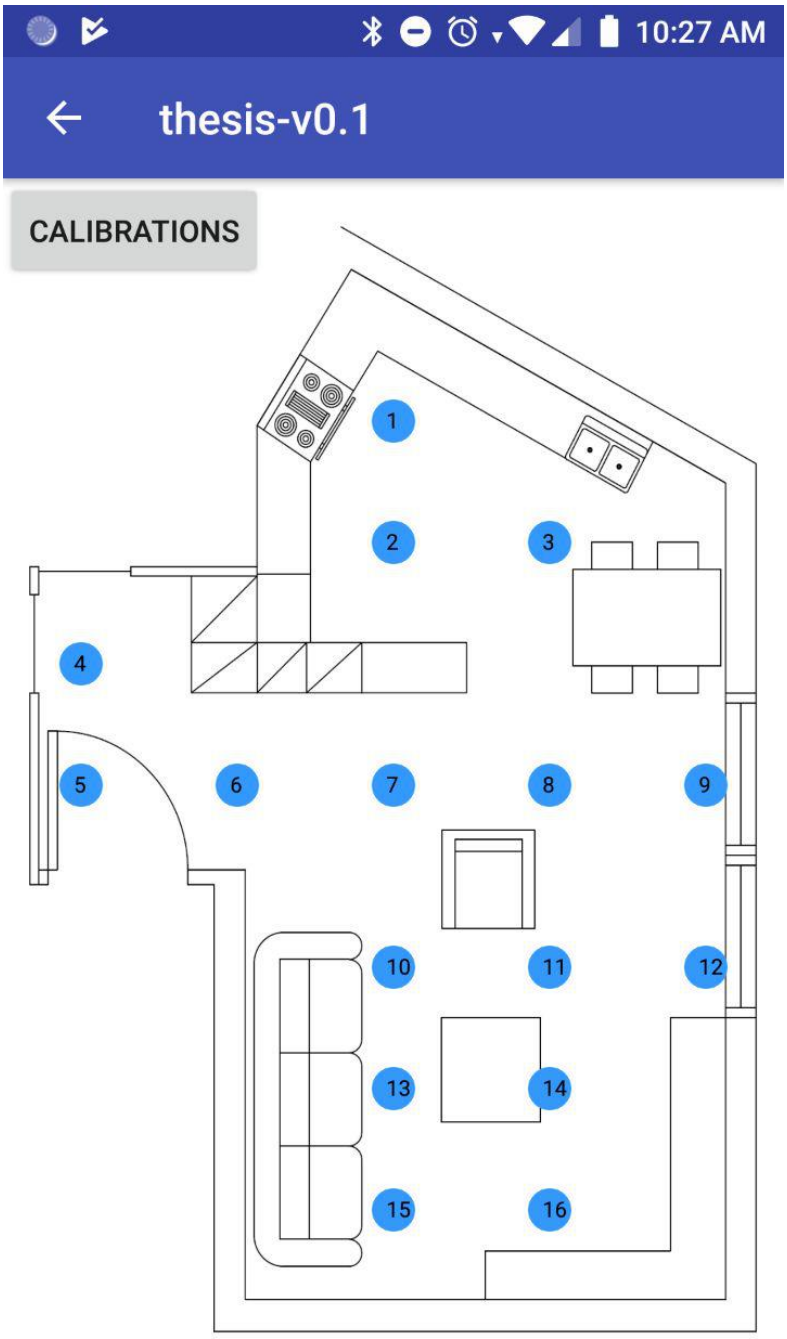
Οι συμβατικές μέθοδοι στιγματοθέτησης σε εσωτερικούς χώρους, όπως είδαμε και στην εισαγωγή, οδηγούνε συχνά σε μεγάλα σφάλματα. Ο λόγος είναι ότι προσπαθούν να υπολογίσουν τη θέση μετρώντας δυναμικά την ισχύ του σήματος και με χρήση trilateration, κάτι που απαιτεί μεγάλη ακρίβεια. Ως αποτέλεσμα το σφάλμα συχνά ξεπερνά τα 2 μέτρα κάτι το οποίο πρέπει να θεωρείται μεγάλο για αποτελεσματική στιηματοθέτηση και πλοήγηση σε εσωτερικούς χώρους.

Με γνώμονα τα παραπάνω, στην παρούσα διπλωματική, επιχειρούμε μία διαφορετική προσέγγιση που προσδοκούμε να δώσει μεγαλύτερη ακρίβεια. Αυτή είναι η εφαρμογή ενός πλέγματος εικονικών κόμβων που αντιστοιχούν βεβαίως σε πιθανές θέσεις στον πραγματικό χώρο. Αυτό μας δίνει δύο σχεδιαστικά πλεονεκτήματα:

- Κατ'αρχάς μπορούμε να κάνουμε το πλέγμα μας όσο πυκνό θέλουμε επιχειρώντας να πετύχουμε μεγαλύτερη ακρίβεια. Βέβαια, αυτό γίνεται με κόστος καθώς θα πρέπει να γίνουν μετρήσεις σε κάθε κόμβο ξεχωριστά. Παράλληλα δεν μπορούμε να εξαλείψουμε το σφάλμα των μετρήσεων κάτι που σημαίνει ότι η μεγάλη αύξηση της πυκνότητας οδηγεί σε αποτελέσματα γνωστά με σφάλμα αντίστοιχο των προσεγγίσεων με trilateration.
- Επιπλέον δεν μας ενδιαφέρουν ούτε η θέση των beacons ούτε η ποσότητά τους. Για το trilateration απαιτούνται τρία beacons και δεδομένων των θέσεών τους στον περιβάλλοντα χώρο υπολογίζεται η εκτίμηση της θέσης του χρήστη. Στη δική μας περίπτωση από κάθε θέση του πλέγματος μετράμε τα RSSI από κάθε beacon στον περιβάλλοντα χώρο και αποθηκεύουμε αυτές τις μετρήσεις ώστε να τις συγκρίνουμε αργότερα με τις μετρήσεις που αναλογούν στην τρέχουσα θέση του χρήστη και έτσι να προσδιορίσουμε ποιος είναι ο κοντινότερος κόμβος και να υποθέσουμε ότι εκεί βρίσκεται ο χρήστης.

Έτσι καταλήξαμε στην δημιουργία ενός πλέγματος που οι αποστάσεις των κόμβων είναι ένα μέτρο στις περισσότερες περιπτώσεις, ενώ σε άλλες είναι ενάμιση. Οι διαφοροποίηση των αποστάσεων είναι σχεδιαστική επιλογή έτσι ώστε εφαρμογή να είναι πιο έτοιμη για σενάρια πραγματικού κόσμου όπου οι αποστάσεις δεν θα είναι πάντα ίσες. Όπως θα δούμε και στο κεφάλαιο της πλοήγησης, σχεδιάζουμε την υλοποίηση ενός αλγορίθμου Dijkstra, αντί κάποιο άλλου αλγορίθμου τύπου DFS/BFS που θα μας έδινε καλύτερη πολυπλοκότητα αλλά θα απαιτούσε ίσες αποστάσεις μεταξύ των κόμβων.

Παρακάτω μπορείτε να δείτε την αναπαράσταση του πλέγματος πάνω από την κάτοψη του χώρου όπως φαίνεται στην αντίστοιχη οθόνη της εφαρμογής μας. Να σημειωθεί ότι η σχεδίαση των κόμβων γίνεται συναρτησιακά με βάση τις συντεταγμένες των κόμβων που έχουν αποθηκευτεί στη βάση δεδομένων.



Έχοντας ορίσει τις θέσεις των κόμβων στη βάση, μπορούμε να προχωρήσουμε στις μετρήσεις. Για την επίτευξη του στόχου θα προσθέσουμε δύο νέες δραστηριότητες όπως είδαμε και στο χάρτη οθονών της εφαρμογής.

Στη δραστηριότητα “Λίστα Μετρήσεων” σχεδιάζουμε μία RecyclerView, μία κλάση που επεκτείνει την κλάση ViewGroup και υλοποιεί τις διαπροσωπείες ScrollingView, NestedScrollingChild2. Έτσι μπορούμε να παρουσιάσουμε με κομψό τρόπο τα αποτελέσματα που φέρνουμε από τη βάση δεδομένων όσα κι αν είναι αυτά.

Την δραστηριότητά μας θα συνοδεύσουν η κλάση NodeListAdapter, που εκτελεί την κλήση των δεδομένων από τη βάση και αναθέτει κάθε μέτρηση σε ένα στοιχείο της λίστας, και τα xml αρχεία content_main και recyclervuew_item, που ορίζουν τον τρόπο σχεδίασης των στοιχείων της λίστας και της λίστας αυτής καθεαυτής στο UI.

Τέλος στη δραστηριότητα “Προσθήκη Νέας Μέτρησης” επιτρέπουμε στο χρήστη να επιλέξει τον κόμβο για τον οποίο προσθέτει τη μέτρηση. Έπειτα μέσω ενός κουμπιού, εκκινεί η διαδικασία της μέτρησης.

Σε αυτή την δραστηριότητα ενεργοποιούμε τις λειτουργίες εισαγωγής δεδομένων όπως περιγράφηκαν νωρίτερα, και έτσι επιτρέπουμε τη λήψη πακέτων από τα beacons στον περιβάλλοντα χώρο. Λαμβάνουμε όλες τις μετρήσεις που χρειαζόμαστε και ελέγχουμε αν αυτές είναι null δηλαδή αν δεν βρέθηκε κανένα beacon στην περιοχή. Στην περίπτωση αυτή επιστρέφουμε ανάλογο μήνυμα στο χρήστη. Εάν η λίστα δεν είναι άδεια τότε επιστρέφουμε τις τιμές των μετρήσεων στην προηγούμενη δραστηριότητα ως Context και εκείνη με αυτές τις τιμές δημιουργεί ένα νέο στιγμιότυπο της κλάσης Node και το προσθέτει στη βάση δεδομένων στον αντίστοιχο πίνακα, χρησιμοποιώντας μεθόδους που ορίσαμε στο αντίστοιχο repository.

Στιγματοθέτηση

Για την στιγματοθέτηση στην εφαρμογή μας απαιτούνται δύο δεδομένα:

- Η τρέχουσα μέτρηση που πραγματοποιείται από τη συσκευή σε πραγματικό χρόνο και μας επιστρέφει ένα στιγμιότυπο της κλάσης Node με default id 99, και αφορά στην τρέχουσα θέση του χρήστη.
- Και το σύνολο των μετρήσεων που έχουν προηγηθεί και βρίσκονται αποθηκευμένες στον πίνακα node_table της βάσης δεδομένων.

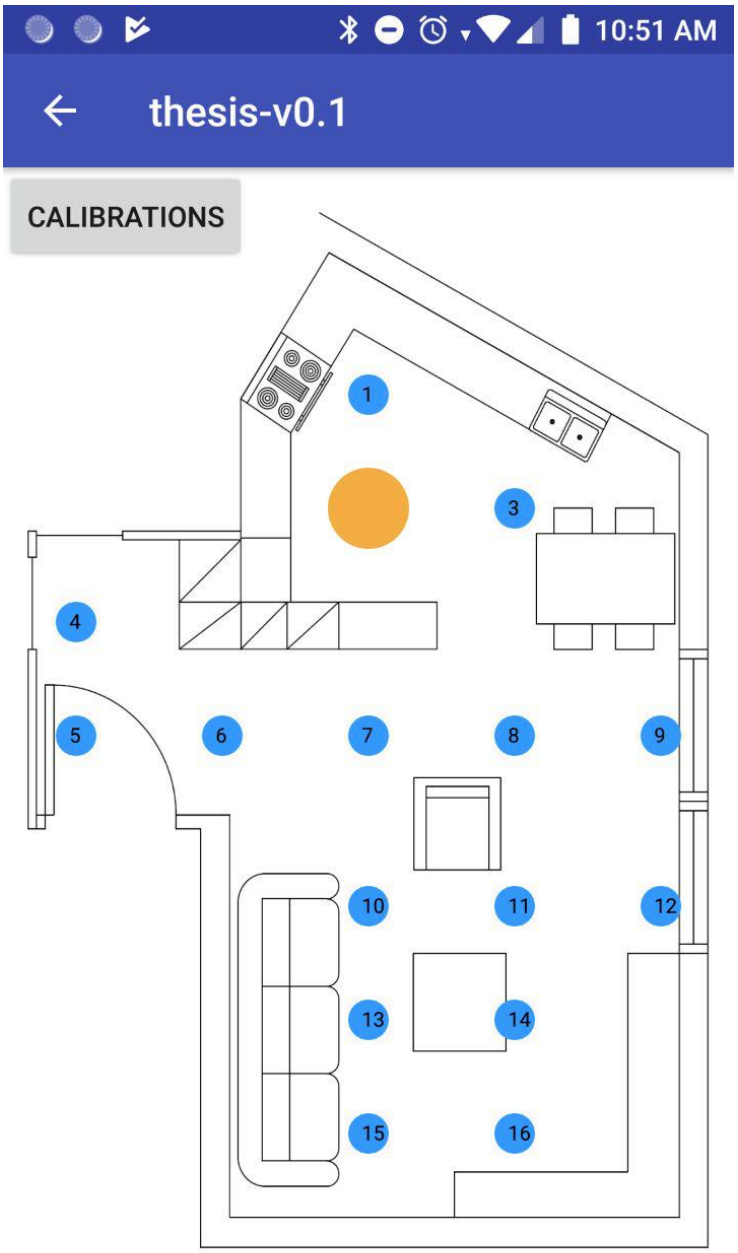
Τόσο η στιγματοθέτηση όσο και η πλοήγηση είναι λειτουργίες που πραγματοποιούνται στην δραστηριότητα Πλοήγησης. Σχετικά με την στιγματοθέτηση οι ανάγκες μας είναι ανάλογες με την δραστηριότητα Προσθήκης Νέας Μέτρησης.

Χρειαζόμαστε να ανοίξουμε την επικοινωνία με τις συσκευές beacons επιτρέποντας στην εφαρμογή μας να χρησιμοποιήσει το Bluetooth της κινητής συσκευής και όπως περιγράψαμε στο κεφάλαιο της Εισαγωγής Δεδομένων, ορίζουμε μία νέα περιοχή BeaconRegion που θα αναγνωρίζει τα beacons μας και έναν νέο beaconManager.

Έπειτα διαμορφώνουμε τα δεδομένα μας ώστε να τα φέρουμε στη σωστή σειρά. Αυτό το καταφέρνουμε με χρήση μίας δομής Switch όπου χρησιμοποιούμε τη μέθοδο getMinor και αντιστοιχίζουμε τις ποσότητες computeAccuracy του κάθε beacon σε αντίστοιχες τοπικές μεταβλητές. Τέλος δημιουργούμε νέο στιγμιότυπο της κλάσης Node με default id 99 και τις παραπάνω μετρήσεις. Έτσι καταλήγουμε με ένα αντικείμενο που περιγράφει την θέση του χρήστη σε κάθε στιγμή, καθώς η μέθοδος onBeaconsDiscovered επιστρέφει λίστα με beacons που ανακαλύπτει με συχνότητα ενός δευτερολέπτου. Από κάθε νέα λίστα παίρνουμε ένα νέο αντικείμενο που περιγράφει τη θέση.

Στη συνέχεια περνάμε το Node της τρέχουσας θέσης σε μία συνάρτηση που υπολογίζει τον κοντινότερο κόμβο. Όπως περιγράψαμε και στη σύνοψη, υπολογίζουμε την απόσταση των μετρήσεων της τρέχουσας θέσης από τις μετρήσεις κάθε αντικειμένου Node που υπάρχει στη βάση δεδομένων στον αντίστοιχο πίνακα. Αυτή η απόσταση ορίζεται όμοια με την καρτεσιανή απόσταση, δηλαδή ως τη ρίζα του αθροίσματος των διαφορών των μετρήσεων ανά beacon υψωμένων στο τετράγωνο.

Παρακάτω μπορείτε να δείτε την αναπαράσταση της στιγματοθέτησης όπως αυτή φαίνεται στην αντίστοιχη οθόνη της εφαρμογής μας.



Στο συγκεκριμένο στιγμιότυπο της εφαρμογής, ο χρήστης εντοπίζεται στη θέση 2, δηλαδή οι μετρήσεις για την τρέχουσα θέση ήταν πιο κοντά σε μετρήσεις που αναλογούν στον κόμβο 2.

Πλοήγηση

Η λειτουργία πλοήγησης εκτελείται στην δραστηριότητα πλοήγησης όπως και η λειτουργία στιγματοθέτησης.

Έχοντας ολοκληρώσει τη στιγματοθέτηση έχουμε πλέον την αρχική μας θέση για την πλοήγηση και ασφαλώς στο σενάριο που υλοποιούμε γνωρίζουμε πάντα και την τελική θέση-στόχο. Όπως φαίνεται και στο πλέγμα κόμβων η έξοδος είναι ο κόμβος 5.

Γνωρίζοντας τα παραπάνω απομένουν δύο πράγματα για την υλοποίηση της σχεδίασης:

- Να γνωστοποιήσουμε το σύνολο των κόμβων μας στην εφαρμογή δηλαδή να βρούμε έναν τρόπο να περιγράψουμε τις θέσεις των κόμβων και τις σχέσεις μεταξύ τους, δηλαδή αν συνδέονται δύο κόμβοι και πόση είναι η απόσταση ανάμεσά τους. Αυτό θα το καταφέρουμε αναπαριστώντας τους κόμβους μας ως μέλη ενός γράφου και ορίζοντας τις ακμές που περιγράφουν τη σύνδεση μεταξύ δύο κόμβων και την απόστασή τους.
- Να υπολογίσουμε το συντομότερο μονοπάτι από την αφετηρία προς την έξοδο, υλοποιώντας των αλγόριθμο του Dijkstra.

Για το πρώτο μέρος προσθέτουμε νέες κλάσεις που περιγράφουν τις έννοιες της κορυφής, της ακμής, και του γράφου. Έτσι έχουμε:

- Vertex, κλάση που ορίζεται από από ένα αναγνωριστικό id και τις συντεταγμένες x και y που περιγράφουν τη θέση της κορυφής στο χάρτη.
- Edge, κλάση που ορίζεται από ένα αναγνωριστικό id, ένα Vertex αφετηρίας, ένα Vertex προορισμού και το βάρος της ακμής που αντιπροσωπεύει την απόσταση μεταξύ των δύο κορυφών.
- Graph, κλάση που ορίζεται από μία λίστα αντικειμένων τύπου Vertex και μία λίστα αντικειμένων τύπου Edge.

Έτσι κατά την εκκίνηση της δραστηριότητας Πλοήγησης και μόλις έχουμε πάρει από τη βάση δεδομένων τις συντεταγμένες των κόμβων μας από τον πίνακα coordinates_table, αρχικοποιούμε το γράφο μας. Έτσι προσθέτουμε αρχικά τις κορυφές με βάση τον πίνακα της βάσης δεδομένων σε μία λίστα, και στη συνέχεια προσθέτουμε τις ακμές, όπου εμείς ορίζουμε τις δυνατές διαδρομές και τις αποστάσεις όπως έχουμε μετρήσει στο φυσικό χώρο, σε μία δεύτερη λίστα. Τέλος δημιουργούμε το στιγμιότυπο της κλάσης Graph με τις δύο λίστες σαν όρισμα.

Για το δεύτερο μέρος, υλοποιούμε τον αλγόριθμο του Dijkstra ως μία νέα κλάση που θα περιέχει όλες τις απαιτούμενες λειτουργίες. Έτσι έχουμε την κλάση DijkstraAlgorithm που έχει σαν όρισμα ένα αντικείμενο τύπου Graph και από αυτό ορίζει τις private μεταβλητές που αφορούν τις λίστες των κορυφών και τον ακμών.

Η χρήση του αλγορίθμου γίνεται σε δύο στάδια:

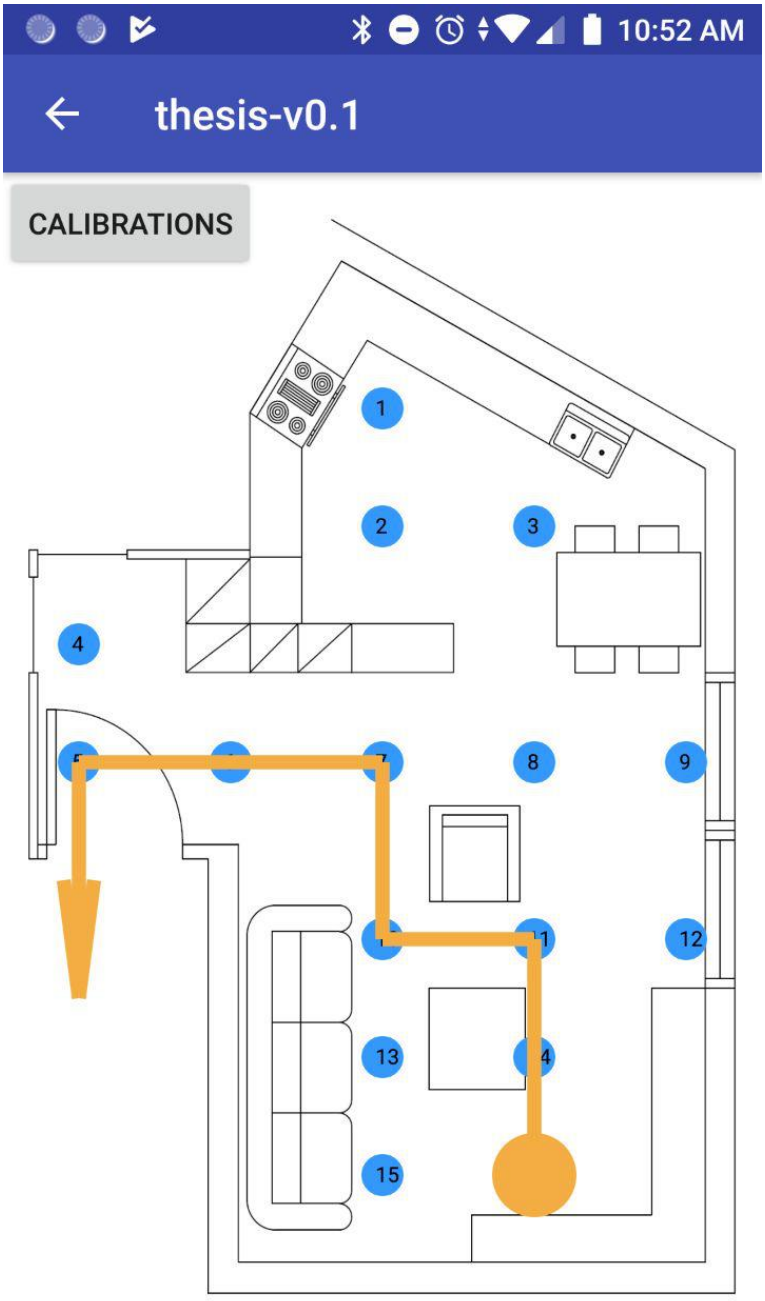
- Πρώτα καλούμε την μέθοδο DijkstraAlgorithm.execute με όρισμα τον κόμβο της τρέχουσας θέσης όπως αυτός έχει προσδιοριστεί από την στιγματοθέτηση που έχει προηγηθεί. Αυτή η μέθοδος θα εκτελέσει όλα τα βήματα ενός αλγορίθμου Dijkstra και θα υπολογίσει γεμίσει έναν πίνακα αποστάσεων με τις ελάχιστες τιμές απόστασης που αναλογούν στον κόμβο εκκίνησης προς κάθε άλλο συνεκτικό κόμβο του γράφου.
- Και έπειτα καλούμε την μέθοδο DijkstraAlgorithm.getPath με όρισμα τον κόμβο-στόχο, στην περίπτωσή μας τον κόμβο 5. Η μέθοδος αυτή θα υπολογίσει το συντομότερο μονοπάτι από τον κόμβο αφετηρίας προς τον τελικό κόμβο αξιοποιώντας των πίνακα αποστάσεων που έχει υπολογιστεί προηγουμένως. Το αποτέλεσμα έρχεται στη μορφή συνδεδεμένης λίστας LinkedList και το αποθηκεύουμε με σε μία τοπική μεταβλητή ως το επιλεγμένο μονοπάτι.

Έχοντας το συντομότερο μονοπάτι, ολοκληρώνεται το υπολογιστικό μέρος της δραστηριότητας Πλοήγησης και απομένει μονάχα η αναπαράσταση αυτού στην οθόνη.

Θα μπορούσαμε να αποφύγουμε την πλήρη υλοποίηση του αλγορίθμου του Dijkstra αφού στο σενάριό μας δεν ενδιαφερόμαστε για όλες τις αποστάσεις και όλα τα μονοπάτια. Θα μπορούσαμε να υλοποιήσουμε μία παραλλαγή του αλγορίθμου όπου θα υπολογίζουμε μονάχα την απόσταση από τον κόμβο 5, όπου βρίσκεται η έξοδος του χώρου.

Ωστόσο για λόγους πληρότητας και κρίθηκε σκόπιμο να γίνει πλήρης υλοποίηση σε περίπτωση που αλλάξει ο σκοπός της εφαρμογής στο μέλλον.

Παρακάτω μπορείτε να δείτε την αναπαράσταση της πλοήγησης όπως αυτή φαίνεται στην αντίστοιχη οθόνη της εφαρμογής μας.



Στο συγκεκριμένο στιγμιότυπο έχει γίνει στιγματοθέτηση και η θέση έχει υπολογιστεί κοντά στον κόμβο 16. Έπειτα η εφαρμογή υπολογίζει όλες τις αποστάσεις από τον κόμβο 16 προς τους άλλους κόμβους και ζητά από τον Dijkstra το συντομότερο μονοπάτι προς τον κόμβο 5, την έξοδο.

Σχεδίαση στην Οθόνη

Η σχεδίαση στην οθόνη γίνεται μέσω της κλάσης DrawView που επεκτείνει την κλάση View. Κύριο χαρακτηριστικό αυτής της κλάσης είναι η συνάρτηση onDraw που λαμβάνει ως όρισμα έναν καμβά. Κάνοντας override την συνάρτηση onDraw που ευθύνεται για τη σχεδίαση στην οθόνη, έχουμε τη δυνατότητα να σχεδιάσουμε τους κόμβους, το στίγμα και το μονοπάτι μας.

Κατά την απαιτούμενη διαδικασία δημιουργούμε νέες μπογιές οι οποίες είναι στιγμιότυπα της κλάσης Paint και ορίζουν χαρακτηριστικά όπως το χρώμα ή το πάχος της χάραξης καθώς και το μέγεθος των χαρακτήρων, εφόσον η μπογιά γράφει κάποια γραμματοσειρά.

Στην εφαρμογή μας χρησιμοποιούμε την κλάση DrawView και τον καμβά της για να σχεδιάσουμε όλους τους κόμβους, σαν κύκλους με κείμενο μέσα τους, καθώς και το στίγμα της θέσης του χρήστη αλλά και το προτεινόμενο μονοπάτι.

Μέσω της δραστηριότητα Πλοήγησης παρατηρούμε τα δεδομένα που μας ενδιαφέρουν και όταν αλλάξουν οι τιμές τους ενημερώνουμε τον καμβά μέσω συναρτήσεων setters που αλλάζουν τις private τιμές που χρησιμοποιεί ο καμβάς. Έπειτα καλούμε τη συνάρτηση της drawView invalidate() που ανοίξει στην κλάση View και ενημερώνει το UIthread ότι η υπάρχουσα κατάσταση στο UI δεν είναι πλέον έγκυρη. Έτσι το UI ανανεώνεται και οι αλλαγές σχεδιάζονται στην οθόνη.

Υλοποίηση Εφαρμογής

Βάση Δεδομένων

Databases

Κλάση NodeRoomDatabase

Η κλάση NodeRoomDatabase είναι μία αφηρημένη κλάση και αποτελεί την αφετηρία της βάσης δεδομένων μας με τη χρήση της βιβλιοθήκης Room. Έτσι αρχικά μπορούμε να παρατηρήσουμε ότι έχει σημειωθεί με @Database και δίπλα ορίζονται οι πίνακες που περιέχονται στη βάση ως οντότητες. Αυτές είναι η κλάση Node και η κλάση Coordinate.

Στη συνέχεια μπορούμε να δούμε ότι σε αυτή την κλάση ορίζεται το μοναδικό στιγμιότυπο της βάσης δεδομένων που περιγράφεται σε αυτή την κλάση. Αυτό όπως έχουμε ήδη παρουσιάσει κατά τον σχεδιασμό της εφαρμογής, το καταφέρνουμε με εφαρμογή του μοντέλου Singleton, εξασφαλίζοντας ότι υπάρχει μονάχα ένα στιγμιότυπο της βάσης δεδομένων.

Έπειτα ορίζουμε τη συνάρτηση PopulateDbAsync, που επεκτείνει την κλάση AsyncTask με την οποία πετυχαίνουμε το γέμισμα της βάσης δεδομένων κατά την εκκίνηση της εφαρμογής. Επιπλέον επειδή είναι ασύγχρονη διεργασία αφαιρούμε βάρος από το κεντρικό υπολογιστικό thread. Μέσα στην κλάση συνδέουμε το στιγμιότυπο της βάσης δεδομένων μας με τις διαπροσωπείες Dao που αφορούν στους πίνακες των Node και των Coordinate αντικειμένων μας.

Τέλος, στη συνάρτηση doInBackground την οποία κάνουμε @Override, αρχικά διαγράφουμε όλα τα αντικείμενα του πίνακα αντικειμένων τύπου Coordinate και στη συνέχεια προσθέτουμε τις συντεταγμένες των κόμβων που ανήκουν στο πλέγμα των κόμβων του χώρου μας, όπως τις έχουμε μετρήσει στον πραγματικό χώρο, ως νέα αντικείμενα τύπου Coordinate.

Κάτι αντίστοιχο δεν συμβαίνει για τον πίνακα των αντικειμένων τύπου αφού οι μετρήσεις που γίνονται από τους κόμβους προστίθενται από τον χρήστη.

Οντότητες Βάσης Δεδομένων

Κλάση Node

Η κλάση Node αποτελεί μία οντότητα της βάσης δεδομένων και σημειώνεται με @Entity που ακολουθείται από το όνομα που δίνουμε στον πίνακα της βάσης δεδομένων. Ακολουθούν οι private μεταβλητές της κλάσης που αποτελούν ουσιαστικά της στήλες του πίνακα node_table. Έχουμε σημειώσει με @PrimaryKey το πρωτεύον κλειδί του πίνακα το οποίο επίσης γίνεται auto-generate και με @NonNull τη μεταβλητή mNumber που αποτελεί το αναγνωριστικό στοιχείο για την κάθε μέτρηση και συνεπώς δεν μπορεί να είναι null.

Ακολουθεί ο constructor που ορίζει τις απαιτούμενες παραμέτρους κατά τη δημιουργία ενός νέου στιγμιοτύπου της κλάσης Node.

Τέλος έχουμε τους απαιτούμενους public getters και setters που μας επιτρέπουν πρόσβαση στις τιμές των private μεταβλητών ενός αντικειμένου τύπου Node.

Κλάση Coordinate

Η κλάση Coordinate αποτελεί την δεύτερη οντότητα της βάσης δεδομένων και σημειώνεται με @Entity που ακολουθείται από το όνομα που δίνουμε στον πίνακα της βάσης δεδομένων. Ακολουθούν οι private μεταβλητές της κλάσης που αποτελούν ουσιαστικά της στήλες του πίνακα coordinate_table. Έχουμε σημειώσει με @PrimaryKey το πρωτεύον κλειδί του πίνακα που αποτελεί και το αναγνωριστικό μίας συντεταγμένης.

Ακολουθεί ο constructor που ορίζει τις απαιτούμενες παραμέτρους κατά τη δημιουργία ενός νέου στιγμιοτύπου της κλάσης Coordinate.

Τέλος έχουμε τους απαιτούμενους public getters και setters που μας επιτρέπουν πρόσβαση στις τιμές των private μεταβλητών ενός αντικειμένου τύπου Coordinate.

DAOs

Διαπροσωπεία NodeDao

Η διαπροσωπεία NodeDao αποτελεί ένα αντικείμενο πρόσβασης δεδομένων που σχετίζεται με τον πίνακα node_table και τα αντικείμενα τύπου Node. Όπως ορίζεται από τη βιβλιοθήκη Room, έχουμε σημειώσει την διαπροσωπεία με @Dao.

Ο ρόλος των Dao είναι να επιτελούν τις λειτουργίες CRUD της βάσης δεδομένων. Έτσι έχουμε:

- τη μέθοδο insert που σημειώνεται με @Insert και δέχεται ως όρισμα ένα αντικείμενο τύπου Node και η Room αναλαμβάνει την προσθήκη του νέου αντικειμένου τύπου Node στον πίνακα node_table της βάσης δεδομένων.
- Τη μέθοδο deleteAll που σημειώνεται με @Query("DELETE FROM node_table") και η Room αναλαμβάνει την διαγραφή όλων των αντικειμένων από τον πίνακα node_table.
- Και τη μέθοδο getAllNodes που σημειώνεται με @Query("SELECT * from node_table ORDER BY mId ASC") και η Room επιστρέφει ένα αντικείμενο τύπου LiveData<List<Node>> που ουσιαστικά περιέχει μία λίστα με όλα τα αντικείμενα του πίνακα node_table αλλά μέσα σε ένα παρατηρήσιμο αντικείμενο τύπου LiveData.

Διαπροσωπεία CoordinateDao

Η διαπροσωπεία CoordinateDao αποτελεί ένα αντικείμενο πρόσβασης δεδομένων που σχετίζεται με τον πίνακα coordinate_table και τα αντικείμενα τύπου Coordinate. Όπως ορίζεται από τη βιβλιοθήκη Room, έχουμε σημειώσει την διαπροσωπεία με @Dao.

Ο ρόλος των Dao είναι να επιτελούν τις λειτουργίες CRUD της βάσης δεδομένων. Έτσι έχουμε:

- τη μέθοδο insert που σημειώνεται με @Insert και δέχεται ως όρισμα ένα αντικείμενο τύπου Coordinate και η Room αναλαμβάνει την προσθήκη του νέου αντικειμένου τύπου Coordinate στον πίνακα coordinate_table της βάσης δεδομένων.
- Τη μέθοδο deleteAll που σημειώνεται με @Query("DELETE FROM coordinate_table") και η Room αναλαμβάνει την διαγραφή όλων των αντικειμένων από τον πίνακα coordinate_table.
- Και τη μέθοδο getAllCoordinates που σημειώνεται με @Query("SELECT * from coordinate_table ORDER BY mId ASC") και η Room επιστρέφει ένα αντικείμενο τύπου LiveData<List<Coordinate>> που ουσιαστικά περιέχει μία λίστα με όλα τα αντικείμενα του πίνακα coordinate_table αλλά μέσα σε ένα παρατηρήσιμο αντικείμενο τύπου LiveData.

Repositories

Κλάση NodeRepository

Η κλάση NodeRepository αποτελεί μία οντότητα η οποία εκτελεί τις μεθόδους του αντίστοιχου DAO. Έτσι αρχικά σε αυτή την κλάση βλέπουμε τον εντοπισμό του στιγμιότυπου της βάσης δεδομένων μέσω της μεθόδου getDatabase που υλοποιεί και τη λογική του μοντέλου Singleton.

Στη συνέχεια μέσω του στιγμιοτύπου της βάσης παίρνουμε το nodeDao και το συνδέουμε με την τοπική μεταβλητή τύπου NodeDao και από αυτό παίρνουμε και το αποτέλεσμα της μεθόδου getAllNodes(). Αυτό αποθηκεύεται στην τοπική μεταβλητή τύπου LiveData<List<Node>> και δημιουργούμε και ένα getter για τη συγκεκριμένη λίστα.

Τέλος υλοποιούμε την κλάση insertAsyncTask που επεκτεί την κλάση AsyncTask και μας επιτρέπει να εκτελέσουμε την μέθοδο insert του DAO σε συνάρτηση doInBackground αφαιρώντας βάρος από το κύριο thread.

Κλάση CoordinateRepository

Η κλάση CoordinateRepository αποτελεί μία οντότητα η οποία εκτελεί τις μεθόδους του αντίστοιχου DAO. Έτσι αρχικά σε αυτή την κλάση βλέπουμε τον εντοπισμό του στιγμιότυπου της βάσης δεδομένων μέσω της μεθόδου getDatabase που υλοποιεί και τη λογική του μοντέλου Singleton.

Στη συνέχεια μέσω του στιγμιοτύπου της βάσης παίρνουμε το coordinateDao και το συνδέουμε με την τοπική μεταβλητή τύπου CoordinateDao και από αυτό παίρνουμε και το αποτέλεσμα της μεθόδου getAllCoordinates(). Αυτό αποθηκεύεται στην τοπική μεταβλητή τύπου LiveData<List<Coordinate>> και δημιουργούμε και ένα getter για τη συγκεκριμένη λίστα.

Τέλος υλοποιούμε την κλάση insertAsyncTask που επεκτεί την κλάση AsyncTask και μας επιτρέπει να εκτελέσουμε την μέθοδο insert του DAO σε συνάρτηση doInBackground αφαιρώντας βάρος από το κύριο thread.

Μεταφορά Δεδομένων

ViewModels

Κλάση NodeViewModel

Η κλάση NodeViewModel αποτελεί το μοναδικό ViewModel της εφαρμογής μας. Παρατηρούμε ότι το ViewModel μας διατηρεί δικά του στιγμιότυπα των δύο repositories, NodeRepository και CoordinateRepository. Παράλληλα διατηρεί και δύο δικές του μεταβλητές τύπου LiveData<List<Node>> και LiveData<List<Coordinate>> που προκύπτουν από τα δύο παραπάνω repositories που περιέχουν στις κλάσεις τους τα αντίστοιχα στοιχεία.

Έτσι χρησιμοποιώντας τους getters που έχουμε ορίσει στα repositories γίνεται ανάθεση τιμών στις δύο μεταβλητές τύπου LiveData. Με τη σειρά του το ViewModel μας παρέχει getters για τα δύο αυτά στοιχεία.

Τέλος το ViewModel μας υλοποιεί τις δύο μεθόδους insert, μία για Node και μία για Coordinate, μέσω των μεθόδων insert που έχουν υλοποιηθεί στα αντίστοιχα repositories.

Δραστηριότητες

Κλάση MainActivity

Η κλάση MainActivity αποτελεί την αρχική μας δραστηριότητα και συνεπώς την αρχική οθόνη που θα δει ο χρήστης όταν ανοίξει την εφαρμογή.

Αρχικά, όπως και σε κάθε δραστηριότητα, στην συνάρτηση onCreate συνδέουμε την δραστηριότητά μας με το αντίστοιχο αρχείο xml που είναι υπεύθυνο για τα στοιχεία που εμφανίζονται στην οθόνη του χρήστη. Η δραστηριότητα MainActivity συνδέεται με το αρχείο activity_main.xml.

Επιπλέον συνδέουμε τις τοπικές μεταβλητές που αφορούν δύο κουμπιά με τα δύο κουμπιά που αναφέρονται στο xml, το bnToCalibration και το bnToNavigation.

Όπως υποδεικνύει και η ονομασία τους, τα κουμπιά αυτά με το πάτημά τους ξεκινάνε ένα νέο Intent που εκκινεί μία νέα δραστηριότητα. Το bnToCalibration θα μας οδηγήσει στην δραστηριότητα Κόμβων ενώ το bnToNavigation θα μας οδηγήσει στη δραστηριότητα Πλοήγησης.

Κλάση CalibrationActivity

Η κλάση CalibrationActivity αποτελεί την δραστηριότητα Κόμβων και σε αυτή ο χρήστης-διαχειριστής μπορεί να δει την κάτοψη του χώρου και το πλέγμα των κόμβων σχεδιασμένο πάνω από αυτήν.

Αρχικά, όπως και σε κάθε δραστηριότητα, στην συνάρτηση onCreate συνδέουμε την δραστηριότητά μας με το αντίστοιχο αρχείο xml που είναι υπεύθυνο για τα στοιχεία που εμφανίζονται στην οθόνη του χρήστη. Η δραστηριότητα CalibrationActivity συνδέεται με το αρχείο activity_calibration.xml.

Έπειτα συνδέουμε το custom View που έχουμε συμπεριλάβει στο xml με την τοπική μεταβλητή τύπου DrawNodesView, μέσω του id του “dvDrawNodes”. Το ίδιο κάνουμε και για το κουμπί του xml με id bnToNodeList. Όπως υποδεικνύει και η ονομασία του, το κουμπί bnToNodeList μας οδηγεί στην δραστηριότητα της Λίστας Μετρήσεων.

Τέλος σε αυτή τη δραστηριότητα καλούμε για πρώτη φορά το ViewModel μας το οποίο εντοπίζουμε μέσω της κλάσης ViewModelProviders. Από το NodeViewModel ζητάμε τη λίστα με τις συντεταγμένες των κόμβων και τις παρατηρούμε. Έτσι μόλις έρθουν από τη βάση, χρησιμοποιούμε τον setter της DrawNodesView και περνάμε τη λίστα στο custom View. Ολοκληρώνουμε τη διαδικασία καλώντας την μέθοδο invalidate() της DraNodesView ώστε να ενημερώσουμε το UIthread ότι η οθόνη δεν είναι πλέον έγκυρη και πρέπει να την σχεδιάσει ξανά.

Κλάση NodeListActivity

Η κλάση NodeListActivity αποτελεί τη δραστηριότητα της Λίστας Μετρήσεων. Εδώ ο χρήστης βλέπει μία λίστα από τα αντικείμενα Node, όπως αυτά έρχονται από τη βάση δεδομένων μέσω του NodeViewModel.

Αρχικά, όπως και σε κάθε δραστηριότητα, στην συνάρτηση onCreate συνδέουμε την δραστηριότητά μας με το αντίστοιχο αρχείο xml που είναι υπεύθυνο για τα στοιχεία που εμφανίζονται στην οθόνη του χρήστη. Η δραστηριότητα NodeListActivity συνδέεται με το αρχείο activity_node_list.xml.

Επιπλέον στο xml αυτής της δραστηριότητα έχουμε μία View τύπου RecyclerView και ένα κουμπί FloatingActionButton. Όμοια με πριν συνδέουμε τα στοιχεία αυτά του xml με τις τοπικές μεταβλητές μέσω των id τους, recyclerview και fab.

Το κουμπί fab μας οδηγεί στην δραστηριότητα Προσθήκης Νέας Μέτρησης.

Επίσης δημιουργούμε ένα στιγμιότυπο της κλάσης NodeListAdapter και το συνδέουμε με την recyclerView μέσω του setter της.

Σε αυτή τη δραστηριότητα καλούμε ξανά το ViewModel μας το οποίο εντοπίζουμε μέσω της κλάσης ViewModelProviders. Από το NodeViewModel ζητάμε τη λίστα με τις μετρήσεις στους κόμβους και τις παρατηρούμε. Έτσι μόλις έρθουν από τη βάση, χρησιμοποιούμε τον setter της NodelistAdapter και περνάμε τη λίστα στο View recyclerView. Από εκεί η recyclerView θα προσθέσει ένα ένα τα αντικείμενα της λίστας των μετρήσεων στη λίστα που βλέπει ο χρήστης στην οθόνη του.

Τέλος, έχουμε τη συνάρτηση onActivityResult που ελέγχει το αποτέλεσμα της επόμενης δραστηριότητας, NewNodeActivity, κοιτώντας τον κωδικό που επιστρέφει η δραστηριότητα μόλις ολοκληρωθεί. Εάν το resultCode είναι RESULT_OK, τότε από το Context του Intent παίρνει τις νέες μετρήσεις και δημιουργεί ένα νέο στιγμιότυπο της κλάσης Node, το οποίο στη συνέχεια προσθέτει στον αντίστοιχο πίνακα node_table της βάσης δεδομένων μέσω της μεθόδου insert που περιέχει το NodeViewModel.

Κλάση NewNodeActivity

Η κλάση NewNodeActivity αποτελεί τη δραστηριότητα Προσθήκης Νέας Μέτρησης. Εδώ ο χρήστης μπορεί να προσθέσει μία νέα μέτρηση στη βάση δεδομένων.

Αρχικά, όπως και σε κάθε δραστηριότητα, στην συνάρτηση onCreate συνδέουμε την δραστηριότητά μας με το αντίστοιχο αρχείο xml που είναι υπεύθυνο για τα στοιχεία που εμφανίζονται στην οθόνη του χρήστη. Η δραστηριότητα NewNodeActivity συνδέεται με το αρχείο activity_new_node.xml.

Σε αυτή τη δραστηριότητα δουλεύουμε με τις μετρήσεις των beacons οπότε ορίζουμε ένα νέο region τύπου BeaconRegion που περιορίζει την ανίχνευση της συσκευής μας για beacons με ένα συγκεκριμένο UUID.

Επίσης ορίζουμε έναν νέο στιγμιότυπο της κλάσης BeaconManager και μέσω αυτού χρησιμοποιούμε τη συνάρτηση onBeaconsDiscovered που επιστρέφει μία λίστα με τα αντικείμενα τύπου Beacon που εντοπίστηκαν με βάση τους περιορισμούς που ορίζει το region μας. Εκεί βλέπουμε ότι παίρνουμε την μέτρηση RSSI του από κάθε beacon και το αντιστοιχίζουμε σε μεταβλητές με συγκεκριμένη σειρά. Αυτή είναι η σειρά με την οποία θέλουμε να δώσουμε τις μετρήσεις ως ορίσματα του νέου αντικειμένου τύπου Node που θα προσθέσουμε στη βάση.

Επίσης έχουμε ένα στοιχείο Spinner στο xml μας, με id spinner1, το οποίο είναι περιέχει όλα τα id των κόμβων ώστε να επιλέξει ο χρήστη για ποιον κόμβο θα εκτελέσει τη μέτρηση.

Επιπλέον συνδέουμε το κουμπί του xml με την τοπική μεταβλητή μέσω του id του, bnCalibrate. Με το πάτημα του κουμπιού δημιουργείται ένα νέο Intent και, εφόσον η λίστα των beacons που ανακαλύφθηκαν στην περιοχή δεν είναι άδεια, προσθέτουμε στο Context του Intent την τιμή του spinner1 και τις μετρήσεις RSSI με την επιθυμητή σειρά. Αυτό τερματίζει και την δραστηριότητα μεταφέροντας το χρήστη πίσω στη δραστηριότητα της Λίστας Μετρήσεων όπου θα δει τη νέα μέτρηση, εφόσον το resultCode ήταν θετικό.

Κλάση NavigationActivity

Η τελευταία κλάση αυτής της ενότητας είναι η NavigationActivity και αποτελεί τη δραστηριότητα Πλοήγησης.

Αρχικά, όπως και σε κάθε δραστηριότητα, στην συνάρτηση onCreate συνδέουμε την δραστηριότητά μας με το αντίστοιχο αρχείο xml που είναι υπεύθυνο για τα στοιχεία που εμφανίζονται στην οθόνη του χρήστη. Η δραστηριότητα NavigationActivity συνδέεται με το αρχείο activity_navigation.xml.

Έπειτα συνδέουμε το custom View που έχουμε συμπεριλάβει στο xml με την τοπική μεταβλητή τύπου DrawView, μέσω του id του “dvNodes”.

Σε αυτή τη δραστηριότητα καλούμε ξανά το ViewModel μας το οποίο εντοπίζουμε μέσω της κλάσης ViewModelProviders. Από το NodeViewModel ζητάμε, αρχικά, τη λίστα με τις μετρήσεις στους κόμβους και τις παρατηρούμε. Έτσι μόλις έρθουν τα δεδομένα από τη βάση, ανανεώνουμε την τοπική μεταβλητή τύπου List nodeList με τα δεδομένα της βάσης. Από το NodeViewModel επίσης ζητάμε τη λίστα με τις συντεταγμένες των κόμβων και την παρατηρούμε. Έτσι μόλις έρθουν τα δεδομένα από τη βάση ενημερώνουμε την τοπική μεταβλητή τύπου List coordinateList με τα δεδομένα της βάσης. Επίσης τότε καλούμε τη συνάρτηση initializeGraph η οποία αρχικοποιεί το γράφο που περιγράφει τον περιβάλλοντα χώρο, ενώ παράλληλα ενημερώνουμε την custom View DrawView μέσω του setter της και καλούμε τη μέθοδο invalidate() έτσι ώστε το UIthread να σχεδιάσει ξανά την οθόνη με τα νέα δεδομένα.

Σε αυτή τη δραστηριότητα, όπως και στη δραστηριότητα Προσθήκης Νέας Μέτρησης, δουλεύουμε με τις μετρήσεις των beacons οπότε ορίζουμε ένα νέο region τύπου BeaconRegion που περιορίζει την ανίχνευση της συσκευής μας για beacons με ένα συγκεκριμένο UUID.

Επίσης ορίζουμε έναν νέο στιγμιότυπο της κλάσης BeaconManager και μέσω αυτού χρησιμοποιούμε τη συνάρτηση onBeaconsDiscovered που επιστρέφει μία λίστα με τα αντικείμενα τύπου Beacon που εντοπίστηκαν με βάση τους περιορισμούς που ορίζει το region μας. Εκεί βλέπουμε ότι παίρνουμε την μέτρηση RSSI του από κάθε beacon και το αντιστοιχίζουμε σε μεταβλητές με συγκεκριμένη σειρά. Αυτή είναι η σειρά με την οποία θέλουμε να δώσουμε τις μετρήσεις ως ορίσματα του νέου αντικειμένου myNode τύπου Node που περιγράφει την τρέχουσα θέση του χρήστη.

Έπειτα καλούμε την συνάρτηση calculateColsestNode με όρισμα τις το αντικείμενο myNode και εκείνη μας επιστρέφει το id του κοντινότερου κόμβου. Αυτή η πληροφορία μεταδίδεται στην DrawView μέσω ενός setter και ανανεώνει τη θέση του χρήστη στην οθόνη.

Επιπλέον αφότου έχει προσδιοριστεί η θέση του χρήστη γίνεται κλήση της συνάρτησης calculateShortestPath με όρισμα τον κόμβο που έχει οριστεί ως η τρέχουσα θέση του χρήστη. Το συντομότερο μονοπάτι μεταδίδεται στην DrawView μέσω ενός setter και ανανεώνει το προτεινόμενο μονοπάτι στην οθόνη. Προφανώς γίνεται κλήση της invalidate() για την DrawView ώστε να γίνει νέα σχεδίαση στην οθόνη.

Τροποποίηση Δεδομένων

Adapters

Κλάση NodeListAdapter

Η κλάση NodeListAdapter είναι μία κλάση που επεκτείνει την κλάση RecyclerView.Adapter.

Ο ρόλος της είναι να διατηρεί μία λίστα με τις μετρήσεις και να ειδοποιεί την RecyclerView όταν αλλάζει το περιεχόμενο της λίστας. Λειτουργεί σαν διαμεσολαβητής ανάμεσα στην δραστηριότητα NodeListActivity και την RecyclerView που συμπεριλαμβάνεται στο xml που είναι υπεύθυνο για την σχεδίαση στην οθόνη.

Διεπαφή Χρήστη

Views

Κλάση DrawView

Η DrawView είναι μία κλάση που επεκτείνει την κλάση View. Χρησιμοποιείται από τη δραστηριότητα NavigationActivity ώστε να πραγματοποιείται δυναμικά χάραξη πάνω στην οθόνη.

Ορίζουμε διάφορες μπογιές τύπου Paint που περιέχουν διαφορετικές οδηγίες σχεδίασης όπως το χρώμα ή το μέγεθος της γραμματοσειράς.

Έπειτα ξεκινάμε να σχεδιάζουμε το πλέγμα των κόμβων με βάση τις συντεταγμένες που έρχονται από τη βάση δεδομένων μέσω της δραστηριότητας NavigationActivity. Το πλέγμα απαρτίζεται από ένα σύνολο κύκλων και αριθμών μέσα τους που σχεδιάζουμε με κλήσεις των συναρτήσεων drawCircle και drawText της κλάσης canvas.

Στη συνέχεια, εφόσον η δραστηριότητα NavigationActivity έχει ενημερώσει τη DrawView με τα απαραίτητα δεδομένα γίνεται σχεδίαση της θέσης του χρήστη αλλά και του προτεινόμενου μονοπατιού. Εδώ χρησιμοιποιούμε κλήσεις των συναρτήσεων drawCircle και drawLine της κλάσης Canvas.

Τέλος βλέπουμε αρκετούς setter που εξυπηρετούν την σωστή ενημέρωση των private μεταβλητών της DrawView από την Navigation Activity.

Κλάση DrawNodesView

Η DrawNodesView είναι μία κλάση που επεκτείνει την κλάση View. Χρησιμοποιείται από τη δραστηριότητα CalibrationActivity ώστε να πραγματοποιείται δυναμικά χάραξη πάνω στην οθόνη.

Ορίζουμε διάφορες μπογιές τύπου Paint που περιέχουν διαφορετικές οδηγίες σχεδίασης όπως το χρώμα ή το μέγεθος της γραμματοσειράς.

Έπειτα ξεκινάμε να σχεδιάζουμε το πλέγμα των κόμβων με βάση τις συντεταγμένες που έρχονται από τη βάση δεδομένων μέσω της δραστηριότητας CalibrationActivity . Το πλέγμα απαρτίζεται από ένα σύνολο κύκλων και αριθμών μέσα τους που σχεδιάζουμε με κλήσεις των συναρτήσεων drawCircle και drawText της κλάσης canvas.

Τέλος βλέπουμε τον setter της λιστας mCoordinates που εξυπηρετεί την σωστή ενημέρωση της private μεταβλητής mCoordinates της DrawView από την CalibrationActivity.

Αλγόριθμοι

Οντότητες Αλγορίθμων

Κλάση Vertex

Η κλάση Vertex αποτελεί μία απλή οντότητα που περιγράφει μία κορυφή ενός γράφου. Τα πεδία της περιγράφουν τις συντεταγμένες x και y πέρα από το id.

Στον κώδικα μπορείτε να δείτε τον constructor της κλάσης, δηλαδή τον τρόπο με τον οποίο δημιουργείται ένα νέο αντικείμενο της κλάσης.

Επιπλέον υπάρχουν και οι getters ώστε να γίνεται σωστά η πρόσβαση στις private μεταβλητές της κλάσης.

Κλάση Edge

Η κλάση Edge αποτελεί μία απλή οντότητα που περιγράφει μία ακμή ενός γράφου. Πέρα από το id της, περιλαμβάνει δύο πεδία τύπου Vertex που είναι οι κορυφές στα δύο άκρα της ακμής, και τέλος το βάρος της ακμής που αντιπροσωπεύει την απόσταση μεταξύ των δύο κορυφών στα άκρα της ακμής.

Στον κώδικα μπορείτε να δείτε τον constructor της κλάσης, δηλαδή τον τρόπο με τον οποίο δημιουργείται ένα νέο αντικείμενο της κλάσης.

Επιπλέον υπάρχουν και οι getters ώστε να γίνεται σωστά η πρόσβαση στις private μεταβλητές της κλάσης.

Κλάση Graph

Η κλάση Graph αποτελεί την τελευταία οντότητα σε αυτή την ενότητα και περιγράφει έναν γράφο. Περιλαμβάνει σαν πεδία της δύο λίστες, μία λίστα από αντικείμενα τύπου Vertex και μία λίστα από αντικείμενα τύπου Edge.

Στον κώδικα μπορείτε να δείτε τον constructor της κλάσης, δηλαδή τον τρόπο με τον οποίο δημιουργείται ένα νέο αντικείμενο της κλάσης.

Dijkstra

Κλάση DijkstraAlgorithm

Η κλάση DijkstraAlgorithm είναι η κλάση που περιέχει την λογική του αλγορίθμου του Dijkstra. Παίρνει σαν όρισμα έναν γράφο και από αυτόν αρχικοποιεί τις λίστες των αντικειμένων τύπου Vertex και τύπου Edge.

Η χρήση της κλάσης γίνεται από την δραστηριότητα NavigationActivity με σκοπό τον υπολογισμό του συντομότερου μονοπατιού από την τρέχουσα θέση του χρήστη προς την έξοδο. Αυτό γίνεται με κλήση δύο μεθόδων της κλάσης DijkstraAlgorithm:

- Execute: με όρισμα ένα αντικείμενο Vertex που αντιπροσωπεύει την αφετηρία και στο σενάριό μας την τρέχουσα θέση του χρήστη στον χώρο, δηλαδή τον κόμβο στον οποίο βρίσκεται πιο κοντά. Ξεκινώντας από αυτόν τον κόμβο ο αλγόριθμος αρχικοποιεί τα HashSets settledNodes και unSettledNodes καθώς και τα HashMaps distance και predecessors. Αρχικά ορίζει την απόσταση του κόμβου αφετηρίας μέχρι τον εαυτό του ως

μηδενική. Παράλληλα προσθέτει τον κόμβο αφετηρίας στο σετ των unSettledNodes. Τέλος μέχρι το σετ unSettledNodes να αδειάσει, η μέθοδος εκτελεί σε επανάληψη τα παρακάτω βήματα:

- Διαλέγει τον κοντινότερο κόμβο από το σετ unSettledNodes.
- Τον προσθέτει στο σετ SettledNodes και τον αφαιρεί από το σετ unSettledNodes.
- Καλεί την συνάρτηση findMinimalDistances με όρισμα τον κόμβο που εξετάζει. Η συνάρτηση αυτή εξετάζει τους γειτονικούς κόμβους. Εάν η καταγεγραμμένη συντομότερη απόσταση στο HashMap distance σε κάποιο γειτονικό κόμβο είναι μικρότερη από αυτή που είναι καταγεγραμμένη για τον κόμβο που εξετάζουμε προσθέτοντας και τη μεταξύ τους απόσταση, τότε ενημερώνεται το HashMap distance με τη νέα συντομότερη απόσταση προς το συγκεκριμένο κόμβο και ο κόμβος προστίθεται στο σετ unSettledNodes.

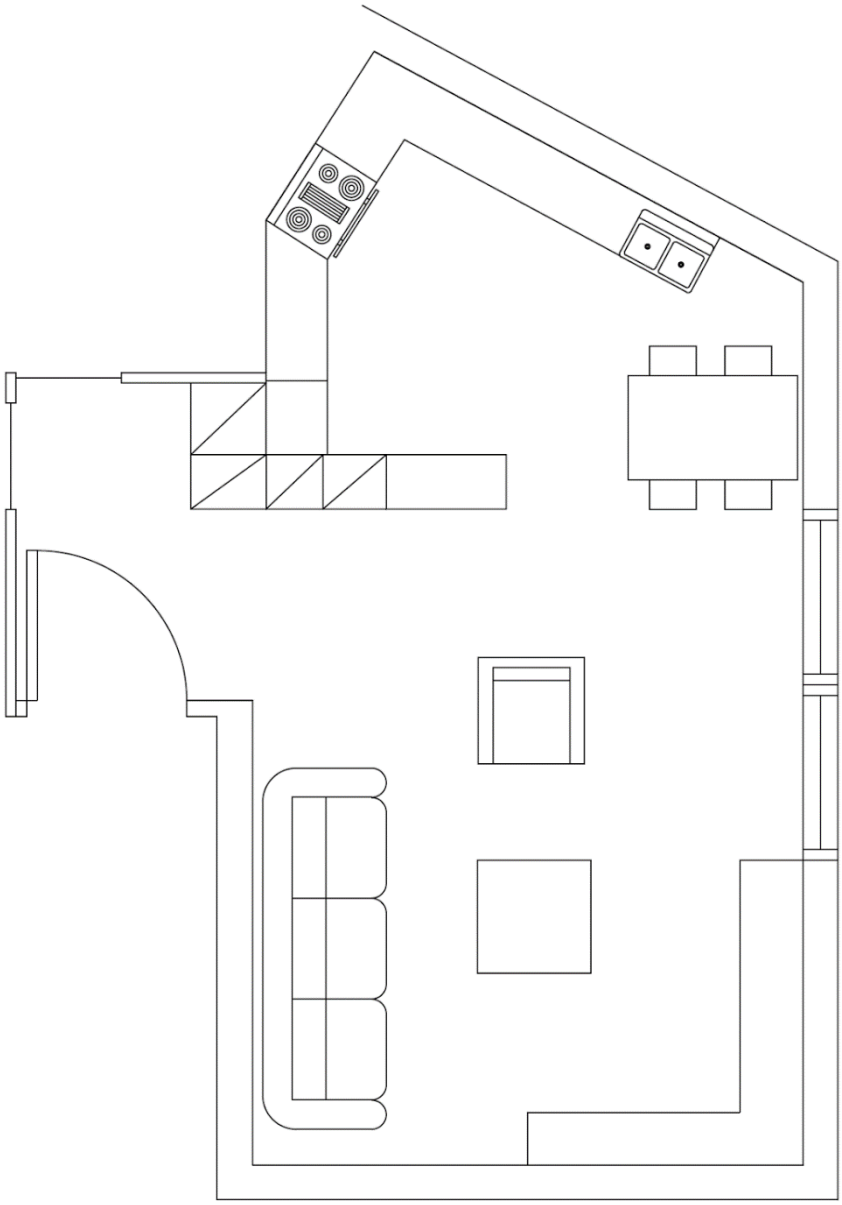
Έτσι μετά το πέρας της execute το HashMap distance περιέχει τις συντομότερες αποστάσεις από τον κόμβο αφετηρίας προς κάθε άλλο κόμβο του γράφου και το HashMap predecessors περιέχει τους προγόνους που οδήγησαν σε αυτές τις συντομότερες αποστάσεις.

- getPath: με όρισμα ένα αντικείμενο Vertex που αντιπροσωπεύει τον προορισμό και στο σενάριό μας την έξοδο, δηλαδή τον κόμβο 5. Η μέθοδος αυτή απλά διαβάζει το HashMap predecessors και επιστρέφει σε μορφή LinkedList το συντομότερο μονοπάτι.

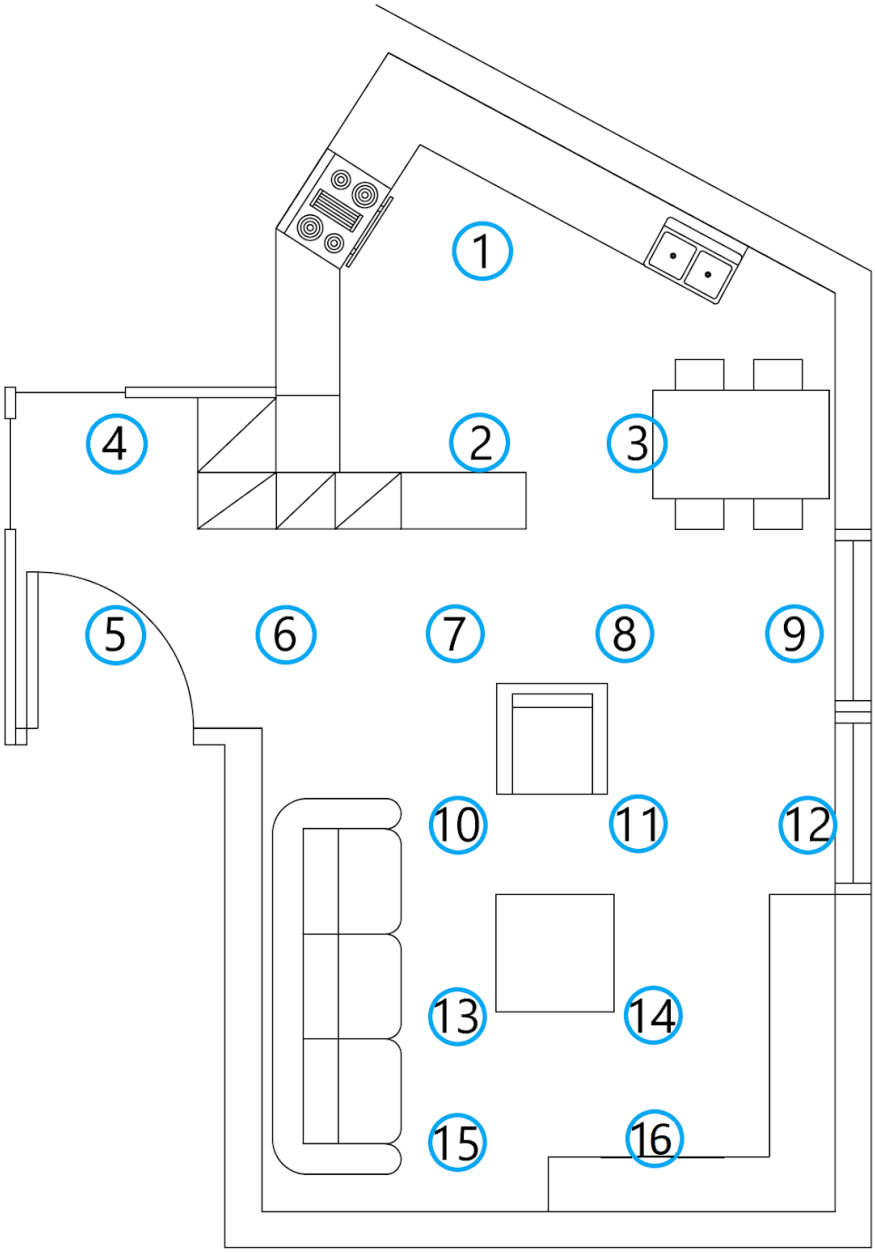
Προετοιμασία Εφαρμογής

Χαρτογράφηση του χώρου και προσθήκη πλέγματος κόμβων

Το βήμα αυτό πραγματοποιείται από τον προγραμματιστή ή παρέχεται σε αυτόν από τον διαχειριστή της εφαρμογής. Στη δική μας περίπτωση έγινε αρχικά η αποτύπωση του χώρου στο εργαλείο ArchiCad παίρνοντας το παρακάτω αποτέλεσμα:



Στη συνέχεια έγινε υπόθεση των θέσεων των κόμβων που εν τέλει απαρτίζουν το πλέγμα:


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Αρχικοποίηση Βάσης Δεδομένων

Αυτό το βήμα γίνεται επίσης από τον προγραμματιστή καθώς οι τιμές των συντεταγμένων των κόμβων πρέπει να περαστούν στην κλάση NodeRoomDatabase.

Έτσι στη προαναφερθείσα κλάση έχουμε τις παρακάτω γραμμές κώδικα που αρχικά διαγράφουν όλες τις εγγραφές του πίνακα coordinate_table και στη συνέχεια τον γεμίζουν με τις νέες εγγραφές:

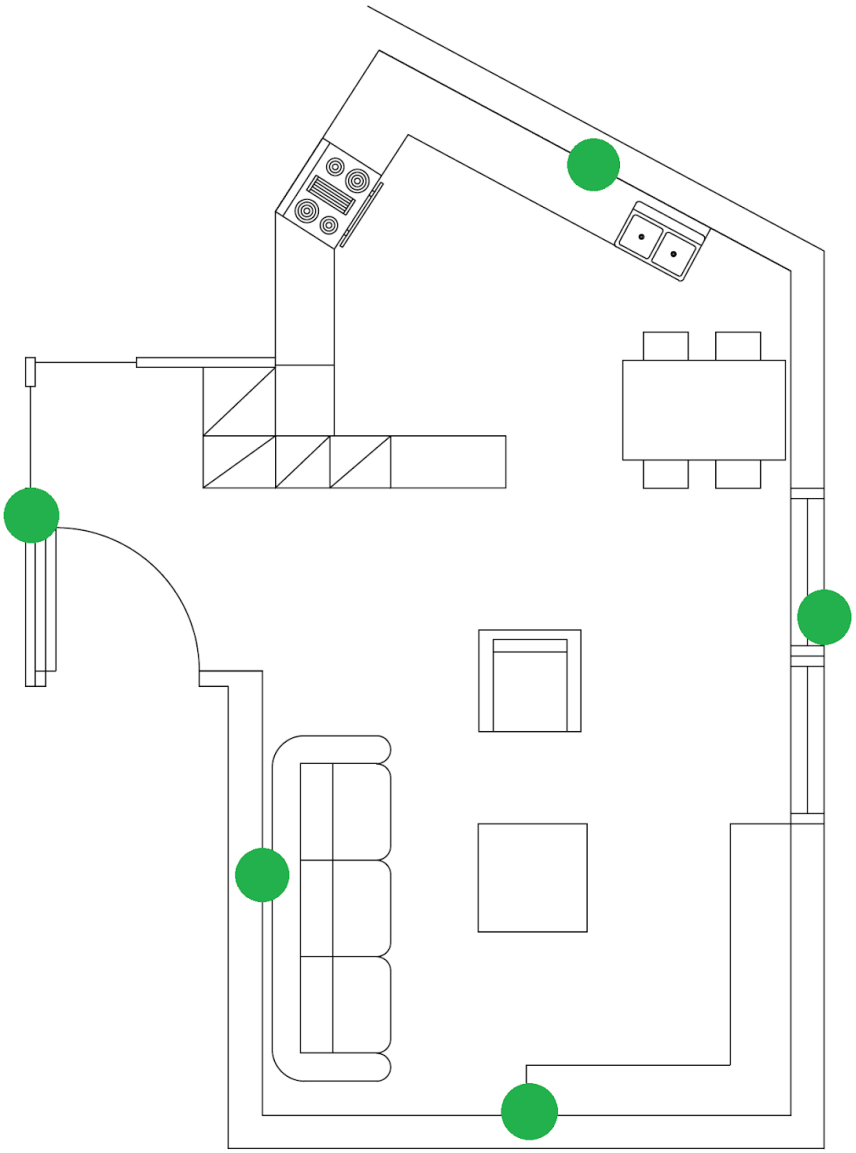
```
mCoordinateDao.deleteAll();
mCoordinateDao.insert(new Coordinate(1,5,2));
mCoordinateDao.insert(new Coordinate(2,5,3));
mCoordinateDao.insert(new Coordinate(3,7,3));
mCoordinateDao.insert(new Coordinate(4,1,4));
mCoordinateDao.insert(new Coordinate(5,1,5));
mCoordinateDao.insert(new Coordinate(6,3,5));
mCoordinateDao.insert(new Coordinate(7,5,5));
mCoordinateDao.insert(new Coordinate(8,7,5));
mCoordinateDao.insert(new Coordinate(9,9,5));
mCoordinateDao.insert(new Coordinate(10,5,6));
mCoordinateDao.insert(new Coordinate(11,7,6));
mCoordinateDao.insert(new Coordinate(12,9,6));
mCoordinateDao.insert(new Coordinate(13,5,7));
mCoordinateDao.insert(new Coordinate(14,7,7));
mCoordinateDao.insert(new Coordinate(15,5,8));
mCoordinateDao.insert(new Coordinate(16,7,8));
```

Αυτές είναι συντεταγμένες έχουν μετρηθεί και ώστε να αντιπροσωπεύουν όσο καλύτερα γίνεται τις αποστάσεις στο φυσικό χώρο.

Τοποθέτηση των Beacons στο χώρο

Αυτό το βήμα εκτελείται από τον διαχειριστή της εφαρμογής καθώς αφορά τον φυσικό χώρο όπου θα χρησιμοποιείται η εφαρμογή.

Η τοποθέτηση των beacons στην εφαρμογή μας δεν χρειάζεται να γίνει σε συγκεκριμένες θέση αρκεί να καλύπτουν όσο καλύτερα γίνεται το χώρο. Στην περίπτωσή μας τα beacons τοποθετήθηκαν σε ψηλά σημεία στους τοίχους έτσι ώστε να μειώσουμε τις παρεμβολές μεταξύ της συσκευής και του κάθε beacon. Η τοποθέτηση φαίνεται στο παρακάτω διάγραμμα:



Μετρήσεις αποστάσεων από κάθε κόμβο

Αυτό το τελικό βήμα γίνεται λογικά από τον διαχειριστή της εφαρμογής. Ο χρήστης-διαχειριστής καλείται να μεταβεί σε κάθε κόμβο όπως αυτοί φαίνονται στην ακριβή κάτοψη που βρίσκεται στην εφαρμογή, και να πραγματοποιήσει τουλάχιστον μία μέτρηση ανά κόμβο.

Σε αυτό το σημείο να αναφέρουμε ότι είναι σημαντικό πλεονέκτημα της υλοποίησης με πλέγμα κόμβων που επιλέξαμε, το ότι μπορούμε να καταχωρήσουμε όσες μετρήσεις θέλουμε από τον κάθε κόμβο. Έτσι μπορούμε να αντιμετωπίσουμε την αστάθεια των μετρήσεων RSSI. Συγκεκριμένα, όπως θα αναφερθεί και παρακάτω στις προτάσεις επέκτασης, έχοντας ένα σετ μετρήσεων για κάποιον κόμβο μπορούμε να τις μελετήσουμε και να "κόψουμε" τις μετρήσεις που αποκλίνουν από το σύνολο. Αυτό σημαίνει ότι με όσες περισσότερες μετρήσεις πραγματοποιήσουμε οδηγούμαστε τελικά σε ελάττωση του σφάλματος.

Αποτελέσματα Εφαρμογής και Συμπεράσματα

Σε αυτό το κεφάλαιο θα εξετάσουμε την ποιότητα των αποτελεσμάτων που προσφέρει η προσέγγιση που κάναμε στο ζήτημα της στιγματοθέτησης καθώς και τις επιδόσεις της εφαρμογής.

Εφαρμόζοντας το πλέγμα των κόμβων επιχειρήσαμε να εξετάσουμε με διαφορετικό τρόπο το ζήτημα της στιγματοθέτησης. Έχουμε δει και στην εισαγωγή ότι ίσως η πιο δημοφιλής μέθοδος είναι αυτή του trilateration. Αυτό γίνεται για ευκολία στην εφαρμογή και οικονομία στο πλήθος των beacons.

Ασφαλώς όσο η τεχνολογία προχωράει θα έχουμε την ευκαιρία να δουλέψουμε το πρόβλημα με μεγαλύτερη ακρίβεια, αλλά τη δεδομένη στιγμή, διαθέτοντας το συγκεκριμένο hardware η ποιότητα των μετρήσεων RSSI που λάβαμε κατά τις πρώτες δοκιμές ήταν αρκετά αναξιόπιστες και η μέθοδος trilateration φάνηκε ότι θα πετύχαινε το σωστό αποτέλεσμα λιγότερες φορές από ότι κάποιο λανθασμένο αποτέλεσμα. Αυτό αποδίδεται κυρίως στην μέθοδο computeAccuracy του EstimoteSDK η οποία παρ' ότι επιστρέφει μία μέτρηση σε μέτρα, δεν προβλέπεται να χρησιμοποιείται για υπολογισμό απόστασης. Η ίδια η εταιρία έχει ονομάσει τα beacons αυτής της σειράς "proximity beacons", όνομα που υποδηλώνει την προβλεπόμενη λειτουργία του. Τα proximity beacons λειτουργούν καλύτερα με εφαρμογές που θέλουν να εκμεταλλευτούν συμβάντα εισόδου και εξόδου, όπως για παράδειγμα την προώθηση διαφημίσεων όταν ένας πελάτης εισέρχεται σε ένα κατάστημα ή ένα αποχαιρετιστήριο μήνυμα όταν βγαίνει.

Συνεπώς αποφασίσαμε να μην διαχειριστούμε τα αποτελέσματα των κλήσεων της computeAccuracy σαν αποστάσεις και αντί αυτού να τα αντιμετωπίσουμε σαν συντεταγμένες. Έτσι οργανώσαμε τα δεδομένα αυτά σαν μετρήσεις σε κόμβους και δημιουργήσαμε μία συλλογή με αρκετές μετρήσεις από κάθε κόμβο. Επιλέξαμε να κάνουμε περισσότερες μετρήσεις ανά κόμβο γιατί με αυτό τον τρόπο καταφέρνουμε να μειώσουμε σημαντικά την επίδραση μίας κακής μέτρησης από κάποιο κόμβο, η οποία μπορεί να δείχνει λάθος μετρήσεις λόγω τυχαίων παραγόντων όπως ότι η χρήστης μπορεί να έχει γυρισμένη την πλάτη του σε κάποιο beacon και έτσι να απορροφάται η ισχύς του σήματος ακόμη και πλήρως.

Παράλληλα δεν χρειάζεται απαραιτήτως να κόψουμε τις μετρήσεις που θεωρούμε λάθος καθώς απλά περιγράφουν μία διαφορετική ανάγνωση του περιβάλλοντα χώρου από την κινητή συσκευή. Αυτό αποτελεί πολύ σημαντικό προβάδισμα της προσέγγισής μας έναντι άλλων προσεγγίσεων όπως το trilateration.

Εξετάζοντας την τρέχουσα θέση είναι πολύ πιθανό και συχνό φαινόμενο να πάρουμε μία συλλογή μετρήσεων που ενώ δεν είναι η πιο δημοφιλής για τη συγκεκριμένη θέση-κόμβο μπορεί να μας υποδείξει τη σωστή θέση, καθώς η εφαρμογή μας συγκρίνει τη συλλογή μετρήσεων της τρέχουσας θέσης με όλες τις συλλογές μετρήσεων στη βάση.

Δεδομένου του παραπάνω μπορούμε να συμπεράνουμε και το εξής. Όσο πιο πολύπλοκες γίνονται οι συλλογές μετρήσεων που αποθηκεύουμε στη βάση δεδομένων, τόσο μεγαλύτερη διαφορετικότητα θα έχουμε.
Σε αυτό σημείο είναι σημαντικό να ξεκαθαρίσουμε ότι η εφαρμογή που περιγράφεται σε αυτή τη διπλωματική μπορεί να λειτουργήσει και μονάχα με ένα beacon. Βέβαια αυτό θα οδηγούσε πιθανότατα σε κακά αποτελέσματα, καθώς όπως παρατηρούμε από τις μετρήσεις μας, είναι πολύ πιθανό μετρώντας από τον ίδιο κόμβο να πάρουμε μετρήσεις όπου το ίδιο beacon επιστρέφει τιμές ή επιστρέφει null. Παράλληλα αυξάνοντας τον αριθμό των beacon που χρησιμοποιούμε σε ένα χώρο όπου τα εύρη τους επικαλύπτονται, δημιουργούμε μετρήσεις με μεγαλύτερη λεπτομέρεια και έχουμε πολλές πιθανότητες να διατηρούμε στη βάση μας μοναδικές μετρήσεις, τουλάχιστον από διαφορετικούς κόμβους.

Έτσι, παρ’ ότι πάντα υπάρχει η πιθανότητα να μην έχουμε σωστό υπολογισμό της θέσης, μπορούμε να δούμε ότι η μέθοδός μας μπορεί να αντιμετωπίσει προβλήματα που οι άλλες μέθοδοι δεν μπορούν, δουλεύοντας με μετρήσεις που άλλες μέθοδοι θα προσπαθούσαν να “κόψουν”.

Τέλος στο ζήτημα της ακρίβειας παίζουν μεγάλο ρόλο και οι αποστάσεις ανάμεσα στους κόμβους που έχουμε στο πλέγμα μας. Όσο μεγαλύτερες οι αποστάσεις τόσο καλύτερες πιθανότητες έχουμε να υπολογίσουμε τον σωστό κόμβο. Ωστόσο οι μεγάλες αποστάσεις δεν είναι επιθυμητές αφού θέλουμε να πετύχουμε και την ακριβέστερη αναπαράσταση της θέσης του χρήστη στο χώρο. Έτσι κρατήσαμε τις αποστάσεις σε ένα με ενάμιση μέτρο ώστε να μπορούμε να ανταγωνιστούμε το σφάλμα των μεθόδων με trilateration που συχνά ξεπερνά τα δύο μέτρα.

Προτάσεις Επέκτασης

Σε αυτό το κεφάλαιο θα γίνει μία σύντομη αναφορά γύρω από ζητήματα που επιδέχονται επέκταση και θα μπορούσαν να βελτιώσουν τα αποτελέσματα της εφαρμογής.

Μία αρχική σκέψη, όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, είναι να διερευνηθούν τα όρια για τα οποία έχουμε όσο πιο πυκνό πλέγμα κόμβων γίνεται, διατηρώντας υψηλά ποσοστά ευστοχίας στην πρόβλεψη της θέσης. Αυτό θα οδηγούσε σε καλύτερη ακρίβεια στην αναπαράσταση της σωστής θέσης, δηλαδή δεδομένου του σωστού υπολογισμού του πλησιέστερου κόμβου, ένα πυκνότερο πλέγμα θα μας έδινε σαν αποτέλεσμα έναν κόμβο που βρίσκεται πιο κοντά στην πραγματική θέση του χρήστη.

Αυτό δημιουργεί ένα δευτερεύον ζήτημα προς διερεύνηση, πώς μπορούμε να κάνουμε τις μετρήσεις στους κόμβους αποδοτικά όταν το πλήθος των κόμβων αυξάνεται σημαντικά, είτε επειδή θέλουμε να δουλέψουμε με ένα πυκνότερο πλέγμα είτε επειδή θέλουμε να καλύψουμε έναν μεγαλύτερο χώρο. Έχοντας πλέον μπει στην εποχή των ρομπότ, θα ήταν ενδιαφέρον ο προγραμματισμός ενός ρομπότ το οποίο θα μπορεί να χαρτογραφήσει το χώρο και παράλληλα, ξέροντας τις συντεταγμένες των κόμβων, να πραγματοποιεί μεγάλο αριθμό μετρήσεων ανά θέση.

Μία άλλη προφανής επέκταση θα ήταν η αντικατάσταση των beacons που διαθέτουμε με beacons νεότερης γενιάς. Οι νέες συσκευές που κυκλοφορούν στο εμπόριο προσφέρουν περισσότερες λειτουργίες όπως η χαρτογράφηση του χώρου, ή ακόμα και η παροχή συντεταγμένων x,y απευθείας από το SDK. Βέβαια καθώς η τεχνολογία είναι ακόμη στα πρώτα της βήματα δεν μπορούμε να ξέρουμε με τι ακρίβεια μπορούν να δίνονται οι παραπάνω συντεταγμένες θέσης και αν τα αποτελέσματα θα είναι αξιόπιστα.

Στο κομμάτι της υλοποίησης της εφαρμογής Android υπάρχουν ασφαλώς πολλές δυνατότητες επέκτασης καθώς φροντίσαμε να θέσουμε πολύ καλές βάσεις σε όλα τα τμήματα. Για παράδειγμα έχοντας υλοποιήσει τα repositories για το πέρασμα των δεδομένων στο viewModel, θα μπορούσαμε να υλοποιήσουμε μία ιστοσελίδα για τον διαχειριστή της εφαρμογής από όπου θα τροποποιεί τα δεδομένα που αφορούν τις μετρήσεις ή τις συντεταγμένες. Η μόνη προσθήκη που θα γινόταν στην εφαρμογή είναι η τροποποίηση του αντίστοιχου repository ώστε να λαμβάνει δεδομένα και από κάποιο web service.

Έχοντας υλοποιήσει έξυπνα την εφαρμογή μας ακολουθώντας σύγχρονα αρχιτεκτονικά μοντέλα και τεχνικές, έχουμε θέσει τα θεμέλια για εύκολες αλλά σημαντικές επεκτάσεις με τον ελάχιστο δυνατό κόπο.

Παράρτημα

Σε αυτό το κεφάλαιο θα γίνει καταγραφή των αρχείων κώδικα Java και Xml σε αλφαβητική σειρά.

Java

CalibrationActivity

```
package com.zzz.prpp.thesis_v01;

import android.arch.lifecycle.Observer;
import android.arch.lifecycle.ViewModelProviders;
import android.content.Intent;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import java.util.List;


public class CalibrationActivity extends AppCompatActivity {

    private NodeViewModel mNodeViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_calibration);

        final DrawNodesView dw = findViewById(R.id.dvDrawNodes);

        Button btn = findViewById(R.id.bnToNodeList);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(CalibrationActivity.this, NodeListActivity.class));
            }
        });

        mNodeViewModel = ViewModelProviders.of(this).get(NodeViewModel.class);

        mNodeViewModel.getAllCoordinates().observe(this, new Observer<List<Coordinate>>() {
            @Override
            public void onChanged(@Nullable List<Coordinate> coordinates) {
                dw.setCoordinates(coordinates);
                dw.invalidate();
            }
        });
    }
}
```

Coordinate

```
package com.zzz.prpp.thesis_v01;

import android.arch.persistence.room.Entity;
import android.arch.persistence.room.PrimaryKey;

@Entity(tableName="Coordinate_table")
public class Coordinate {

    @PrimaryKey
    private Integer mId;
    private Integer mX;
    private Integer mY;

    //Constructors
    public Coordinate(Integer id, Integer x, Integer y){
        this.mId = id;
        this.mX = x;
        this.mY = y;
    }

    //getters
    public Integer getMId() {return mId;}
    public Integer getMX() {return mX;}
    public Integer getMY() {return mY;}

    //setters
    public void setMId(Integer mId) {this.mId = mId;}
    public void setMX(Integer mX) {this.mX = mX;}
    public void setMY(Integer mY) {this.mY = mY;}
}
```

CoordinateDao

```
package com.zzz.prpp.thesis_v01;

import android.arch.lifecycle.LiveData;
import android.arch.persistence.room.Dao;
import android.arch.persistence.room.Insert;
import android.arch.persistence.room.Query;

import java.util.List;

@Dao
public interface CoordinateDao {

    @Insert
    void insert(Coordinate coordinate);

    @Query("DELETE FROM coordinate_table")
    void deleteAll();

    @Query("SELECT * from coordinate_table ORDER BY mId ASC")
    LiveData<List<Coordinate>> getAllCoordinates();
}
```

CoordinateRepository

```
package com.zzz.prpp.thesis_v01;

import android.app.Application;
import android.arch.lifecycle.LiveData;
import android.os.AsyncTask;

import java.util.List;

public class CoordinateRepository {

    private CoordinateDao mCoordinateDao;
    private LiveData<List<Coordinate>> mAllCoordinates;

    CoordinateRepository(Application application) {
        NodeRoomDatabase db = NodeRoomDatabase.getDatabase(application);
        mCoordinateDao = db.coordinateDao();
        mAllCoordinates = mCoordinateDao.getAllCoordinates();
    }

    LiveData<List<Coordinate>> getmAllCoordinates(){
        return mAllCoordinates;
    }

    public  void insert (Coordinate coordinate) { new
insertAsyncTask(mCoordinateDao).execute(coordinate);}

    private  static class  insertAsyncTask extends AsyncTask<Coordinate, Void, Void> {

        private CoordinateDao mAsyncTaskDao;

        insertAsyncTask(CoordinateDao dao) {
            mAsyncTaskDao = dao;
        }

        @Override
        protected Void doInBackground(final Coordinate... params) {
            mAsyncTaskDao.insert(params[0]);
            return null;
        }
    }
}
```

DijkstraAlgorithm

```
package com.zzz.prpp.thesis_v01;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class DijkstraAlgorithm {

   private final List<Vertex> nodes;
   private final List<Edge> edges;
   private Set<Vertex> settledNodes;
   private Set<Vertex> unSettledNodes;
   private Map<Vertex, Vertex> predecessors;
   private Map<Vertex, Float> distance;

   public DijkstraAlgorithm(Graph graph) {
      // create a copy of the array so that we can operate on this array
      this.nodes = new ArrayList<>(graph.getVertexes());
      this.edges = new ArrayList<>(graph.getEdges());
   }

   public void execute(Vertex source) {
      settledNodes = new HashSet<>();
      unSettledNodes = new HashSet<>();
      distance = new HashMap<>();
      predecessors = new HashMap<>();
      distance.put(source, 0f);
      unSettledNodes.add(source);
      while (unSettledNodes.size() > 0) {
         Vertex node = getMinimum(unSettledNodes);
         settledNodes.add(node);
         unSettledNodes.remove(node);
         findMinimalDistances(node);
      }
   }


private void findMinimalDistances(Vertex node) {
      List<Vertex> adjacentNodes = getNeighbors(node);
      for (Vertex target : adjacentNodes) {
         if (getShortestDistance(target) > getShortestDistance(node)
               + getDistance(node, target)) {
            distance.put(target, getShortestDistance(node)
                  + getDistance(node, target));
            predecessors.put(target, node);
            unSettledNodes.add(target);
         }
      }
```

```
}

private float getDistance(Vertex node, Vertex target) {
   for (Edge edge : edges) {
      if (edge.getSource().equals(node)
            && edge.getDestination().equals(target)) {
         return edge.getWeight();
      }
   }
   throw new RuntimeException("Should not happen");
}

private List<Vertex> getNeighbors(Vertex node) {
   List<Vertex> neighbors = new ArrayList<>();
   for (Edge edge : edges) {
      if (edge.getSource().equals(node)
            && !isSettled(edge.getDestination())) {
         neighbors.add(edge.getDestination());
      }
   }
   return neighbors;
}

private Vertex getMinimum(Set<Vertex> vertexes) {
   Vertex minimum = null;
   for (Vertex vertex : vertexes) {
      if (minimum == null) {
         minimum = vertex;
      } else {
         if (getShortestDistance(vertex) < getShortestDistance(minimum)) {
            minimum = vertex;
         }
      }
   }
   return minimum;
}

private boolean isSettled(Vertex vertex) {
   return settledNodes.contains(vertex);
}

private float getShortestDistance(Vertex destination) {
   float d;
   if (distance.get(destination) == null) d = 100f;
   else d = distance.get(destination);
      return d;
}

/*
 * This method returns the path from the source to the selected target and
 * NULL if no path exists
 */
public LinkedList<Vertex> getPath(Vertex target) {
   LinkedList<Vertex> path = new LinkedList<>();
   Vertex step = target;
```

```
        // check if a path exists
        if (predecessors.get(step) == null) {
            return null;
        }
        path.add(step);
        while (predecessors.get(step) != null) {
            step = predecessors.get(step);
            path.add(step);
        }
        // Put it into the correct order
        Collections.reverse(path);
        return path;
    }

}
```

DrawNodesView

```
package com.zzz.prpp.thesis_v01;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.support.annotation.Nullable;
import android.util.AttributeSet;
import android.view.View;

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

public class DrawNodesView extends View {

    private List<Coordinate> mCoordinates = new ArrayList<>();

    int textColor = Color.parseColor("#000000");
    int solidColor = Color.parseColor("#3399ff");

    public DrawNodesView(Context context) {
        super(context);
    }

    public DrawNodesView(Context context, @Nullable AttributeSet attrs) {
        super(context, attrs);
    }

    public DrawNodesView(Context context, @Nullable AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
    }

    @Override
    public void onDraw(Canvas canvas) {

        setWillNotDraw(false);

        Paint circlePaint = new Paint();
        circlePaint.setColor(solidColor);
        circlePaint.setFlags(Paint.ANTI_ALIAS_FLAG);

        Paint textPaint = new Paint();
        textPaint.setColor(textColor);
        textPaint.setTextSize(30);
        textPaint.setFlags(Paint.ANTI_ALIAS_FLAG);

        float offsetY;
        int radius = 30;
        int h = this.getHeight();
        int w = this.getWidth();

        if (!mCoordinates.isEmpty()) {
            System.out.println("mCoordinates not empty");
            for (Coordinate c : mCoordinates) {
```

```
            if (c.getMId() > 9) offsetY = 0.5f;
            else offsetY = 0f;
            canvas.drawCircle((float) c.getMX() * w / 10, (float) (c.getMY() + offsetY) * h / 10,
radius, circlePaint);
            canvas.drawText(String.valueOf(c.getMId()), (float) c.getMX() * w / 10 - radius / 3,
(float) (c.getMY() + offsetY) * h / 10 + radius / 3, textPaint);
          }
       }

       System.out.println("I WILL DRAW NOW");
       super.onDraw(canvas);
    }

    void setCoordinates(List<Coordinate> coordinates){
       System.out.println("Coordinates size: "+coordinates.size());
       mCoordinates = coordinates;
    }

}
```

DrawView

```
package com.zzz.prpp.thesis_v01;

import android.arch.lifecycle.LiveData;
import android.arch.lifecycle.ViewModelProviders;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.support.annotation.Nullable;
import android.util.AttributeSet;
import android.view.View;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class DrawView extends View {

    private List<Coordinate> mCoordinates = new ArrayList<>();
    private Integer current_position;
    private LinkedList<Vertex> path = new LinkedList<>();
    private Vertex previousVertex;

    int textColor = Color.parseColor("#000000");
    int solidColor = Color.parseColor("#3399ff");
    int positionColor = Color.parseColor("#f4ad42");

    public DrawView(Context context) {
        super(context);
    }

    public DrawView(Context context, @Nullable AttributeSet attrs) {
        super(context, attrs);
    }

    public DrawView(Context context, @Nullable AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
    }

    @Override
    public void onDraw(Canvas canvas) {

        setWillNotDraw(false);

        Paint circlePaint = new Paint();
        circlePaint.setColor(solidColor);
        circlePaint.setFlags(Paint.ANTI_ALIAS_FLAG);

        Paint textPaint = new Paint();
        textPaint.setColor(textColor);
        textPaint.setTextSize(30);
        textPaint.setFlags(Paint.ANTI_ALIAS_FLAG);

        Paint positionPaint = new Paint();
```

```
        positionPaint.setColor(positionColor);
        positionPaint.setFlags(Paint.ANTI_ALIAS_FLAG);

        Paint pathPaint = new Paint();
        pathPaint.setColor(positionColor);
        pathPaint.setFlags(Paint.ANTI_ALIAS_FLAG);
        pathPaint.setStrokeWidth(20f);

        float offsetY;
        float previousOffsetY = 0f;
        int radius = 30;
        int h = this.getHeight();
        int w = this.getWidth();
        if (current_position == null) current_position = 2;

        if (!mCoordinates.isEmpty()) {
            for (Coordinate c : mCoordinates) {
                if (c.getMId() > 9) offsetY = 0.5f;
                else offsetY = 0f;
                canvas.drawCircle((float) c.getMX() * w / 10, (float) (c.getMY() + offsetY) * h / 10,
radius, circlePaint);
                canvas.drawText(String.valueOf(c.getMId()), (float) c.getMX() * w / 10 - radius / 3,
(float) (c.getMY() + offsetY) * h / 10 + radius / 3, textPaint);
                if (c.getMId() == current_position)
                    canvas.drawCircle((float) c.getMX() * w / 10, (float) (c.getMY() + offsetY) * h / 10,
radius * 2, positionPaint);
            }
            if (!path.isEmpty()) {
                previousVertex = null;
                for (Vertex v : path) {
                    if (v.getId() > 9) offsetY = 0.5f;
                    else offsetY = 0f;
                    if (previousVertex != null) {
                        canvas.drawLine(previousVertex.getMX() * w / 10, (previousVertex.getMY() +
previousOffsetY) * h / 10, v.getMX() * w / 10, (v.getMY() + offsetY) * h / 10, pathPaint);
                    }
                    previousVertex = v;
                    previousOffsetY = offsetY;
                }
                canvas.drawLine(1f * w / 10,5f * h / 10,1f * w / 10,7f * h / 10,pathPaint);
                canvas.drawLine(0.8f * w / 10,6f * h / 10,1f * w / 10,7f * h / 10,pathPaint);
                canvas.drawLine(1.2f * w / 10,6f * h / 10,1f * w / 10,7f * h / 10,pathPaint);
            }
        }

        super.onDraw(canvas);
    }

    void setCoordinates(List<Coordinate> coordinates){
        mCoordinates = coordinates;
    }

    void increaseCurrentPosition(){
        current_position++;
    }
```

```
    void setCurrentPosition(Integer pos){
        current_position = pos;
    }

    void setShortestPath(LinkedList<Vertex> p){
        path = p;
        invalidate();
    }
}
```

Edge

```
package com.zzz.prpp.thesis_v01;

public class Edge {
    private final Integer id;
    private final Vertex source;
    private final Vertex destination;
    private final float weight;

    public Edge(Integer id, Vertex source, Vertex destination, Float weight) {
        this.id = id;
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }

    public Integer getId() {
        return id;
    }
    public Vertex getDestination() {
        return destination;
    }

    public Vertex getSource() {
        return source;
    }
    public Float getWeight() {
        return weight;
    }

    @Override
    public String toString() {
        return source + " " + destination;
    }
}
```

Graph

```
package com.zzz.prpp.thesis_v01;

import java.util.List;

public class Graph {
   private final List<Vertex> vertexes;
   private final List<Edge> edges;

   public Graph(List<Vertex> vertexes, List<Edge> edges) {
      this.vertexes = vertexes;
      this.edges = edges;
   }

   public List<Vertex> getVertexes() {
      return vertexes;
   }

   public List<Edge> getEdges() {
      return edges;
   }
}
```

MainActivity

```
package com.zzz.prpp.thesis_v01;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btn = findViewById(R.id.bnToCalibration);
        Button btn2 = findViewById(R.id.bnToNavigation);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(MainActivity.this, CalibrationActivity.class));
            }
        });

        btn2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(MainActivity.this, NavigationActivity.class));
            }
        });
    }
}
```

NavigationActivity

```
package com.zzz.prpp.thesis_v01;

import android.arch.lifecycle.Observer;
import android.arch.lifecycle.ViewModelProviders;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import com.estimote.coresdk.common.requirements.SystemRequirementsChecker;
import com.estimote.coresdk.observation.region.beacon.BeaconRegion;
import com.estimote.coresdk.recognition.packets.Beacon;
import com.estimote.coresdk.service.BeaconManager;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.UUID;

import static com.estimote.coresdk.observation.region.RegionUtils.computeAccuracy;
import static junit.framework.Assert.assertNotNull;
import static junit.framework.Assert.assertTrue;

public class NavigationActivity extends AppCompatActivity {

    private NodeViewModel mNodeViewModel;
    List<Node> nodeList = new ArrayList<>();
    List<Coordinate> coordinateList = new ArrayList<>();

    private BeaconManager beaconManager;
    private BeaconRegion region;
    public Integer closestNodeNumber;
    private Integer closestNodeId = 0;
    private Integer exit = 5;

    private List<Vertex> vertices;
    private List<Edge> edges;
    private Graph graph;

    private Double one = 100.0;
    private Double two = 100.0;
    private Double three = 100.0;
    private Double four = 100.0;
    private Double five = 100.0;
    private Double six = 100.0;

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

```
setContentView(R.layout.activity_navigation);

final DrawView vw = findViewById(R.id.dvNodes);

Button btn = findViewById(R.id.bnIncreasePosition);
btn.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      vw.increaseCurrentPosition();
      vw.invalidate();
   }
});

// Get a new or existing ViewModel from the ViewModelProvider.
mNodeViewModel = ViewModelProviders.of(this).get(NodeViewModel.class);

mNodeViewModel.getAllNodes().observe(this, new Observer<List<Node>>() {
   @Override
   public void onChanged(@Nullable List<Node> nodes) {
      nodeList = nodes;
      System.out.println("Nodes size: "+nodes.size());
   }
});

// Add an observer on the LiveData returned by getAllCoordinates.
// The onChanged() method fires when the observed data changes and the activity is
// in the foreground.
mNodeViewModel.getAllCoordinates().observe(this, new Observer<List<Coordinate>>() {
   @Override
   public void onChanged(@Nullable List<Coordinate> coordinates) {
      coordinateList = coordinates;
      if (graph == null) initializeGraph();
      vw.setCoordinates(coordinates);
      vw.invalidate();
   }
});

region = new BeaconRegion("ranged region",
      UUID.fromString("B9407F30-F5F8-466E-AFF9-25556B57FE6D"), null, null);

beaconManager = new BeaconManager(this);
beaconManager.setRangingListener(new BeaconManager.BeaconRangingListener() {
   @Override
   public void onBeaconsDiscovered(BeaconRegion region, List<Beacon> list) {
      if (!list.isEmpty()) {
         clear();
         for (Beacon item : list) {
            switch (item.getMinor()) {
               case 2224:
                  one = computeAccuracy(item);
                  break;
               case 46152:
                  two = computeAccuracy(item);
                  break;
               case 41111:
```

```
                        three = computeAccuracy(item);
                        break;
                    case 48918:
                        four = computeAccuracy(item);
                        break;
                    case 48677:
                        five = computeAccuracy(item);
                        break;
                }
            }
        }
        Node myNode = new Node(99, one, two, three, four, five, null);
        closestNodeNumber = calculateClosestNode(myNode);
        vw.setCurrentPosition(closestNodeNumber);
        if (closestNodeNumber != 0 && calculateShortestPath(closestNodeNumber) != null)
vw.setShortestPath(calculateShortestPath(closestNodeNumber));
        vw.invalidate();
      }
    });


  }

  private Integer calculateClosestNode(Node node) {
    double minDistance = 100.0;
    if (!nodeList.isEmpty()) {
      for (Node nd : nodeList) {
        double distance = Math.sqrt(Math.pow(node.getMOne() - nd.getMOne(), 2) +
Math.pow(node.getMTwo() - nd.getMTwo(), 2) + Math.pow(node.getMThree() - nd.getMThree(),
2) + Math.pow(node.getMFour() - nd.getMFour(), 2) + Math.pow(node.getMFive() - nd.getMFive(),
2));
        if (distance < minDistance) {
          closestNodeId = nd.getMNumber();
        }
      }
    }
    return closestNodeId;
  }

  private void clear() {
    one = 100.0;
    two = 100.0;
    three = 100.0;
    four = 100.0;
    five = 100.0;
    six = 100.0;
  }

  public static double round(double value) {
    BigDecimal bd = new BigDecimal(value);
    bd = bd.setScale(2, RoundingMode.HALF_UP);
    return bd.doubleValue();
  }
```

```
@Override
protected void onResume() {
    super.onResume();
    SystemRequirementsChecker.checkWithDefaultDialogs(this);
    beaconManager.connect(new BeaconManager.ServiceReadyCallback() {
        @Override
        public void onServiceReady() {
            beaconManager.startRanging(region);
        }
    });
}

@Override
protected void onPause() {
    beaconManager.stopRanging(region);
    super.onPause();
}

private LinkedList<Vertex> calculateShortestPath(Integer pos){

    DijkstraAlgorithm dijkstra = new DijkstraAlgorithm(graph);
    dijkstra.execute(vertices.get(pos-1));
    LinkedList<Vertex> path = dijkstra.getPath(vertices.get(exit-1));
    if (path != null && path.size()==1) System.out.println("You are already at the exit");

    return path;
}

private void initializeGraph(){
    vertices = new ArrayList<>();
    edges = new ArrayList<>();

    addVertices();
    addEdges();
    addReverseEdges();

    graph = new Graph(vertices,edges);
}

private void addVertices(){
    for (Coordinate c : coordinateList){
        vertices.add(new Vertex(c.getMId(),(float) c.getMX(),(float) c.getMY()));
    }
}

private void addEdges(){
    addLane(1,1,2,1);
    addLane(2,2,3,1);
    addLane(3,3,8,2);
    addLane(4,4,5,1);
    addLane(5,5,6,1);
    addLane(6,6,7,1);
    addLane(7,7,8,1);
    addLane(8,7,10,1.5f);
    addLane(9,8,9,1);
```

```
        addLane(10,8,11,1.5f);
        addLane(11,9,12,1.5f);
        addLane(12,10,11,1);
        addLane(13,10,13,1);
        addLane(14,11,12,1);
        addLane(15,11,14,1);
        addLane(16,13,15,1);
        addLane(17,14,16,1);
        addLane(18,15,16,1);
    }

    private void addReverseEdges(){
        List<Edge> edgesHelper = new ArrayList<>();
        for (Edge e : edges){
            edgesHelper.add(new Edge(e.getId()+18,e.getDestination(),e.getSource(),e.getWeight()));
        }
        edges.addAll(edgesHelper);
    }

    private void addLane(int laneId, int sourceLocNo, int destLocNo,
                         float duration) {
        Edge lane = new Edge(laneId,vertices.get(sourceLocNo-1), vertices.get(destLocNo-1), duration
);
        //printEdge(lane);
        edges.add(lane);
    }

    private void printVert(){
        System.out.println("These are the vertices in order:");
        for (Vertex v : vertices)
            System.out.println(v.getId());
    }

    private void printEdge(Edge e){
        System.out.println(e.getId()+","+e.getSource().getId()+","+e.getDestination().getId()+","+e.get
Weight());
    }

    private void printEdges(){
        for (Edge e : edges)
            printEdge(e);
    }
}
```

NewNodeActivity

```
package com.zzz.prpp.thesis_v01;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;

import com.estimote.coresdk.common.requirements.SystemRequirementsChecker;
import com.estimote.coresdk.observation.region.beacon.BeaconRegion;
import com.estimote.coresdk.recognition.packets.Beacon;
import com.estimote.coresdk.service.BeaconManager;
import static com.estimote.coresdk.observation.region.RegionUtils.computeAccuracy;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.List;
import java.util.UUID;


public class NewNodeActivity extends AppCompatActivity {

    public static final String EXTRA_REPLY = "com.zzz.prpp.thesis_v01.REPLY";

    private BeaconManager beaconManager;
    private BeaconRegion region;

    private Double one = 100.0;
    private Double two = 100.0;
    private Double three = 100.0;
    private Double four = 100.0;
    private Double five = 100.0;
    private Double six = 100.0;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_new_node);

        region = new BeaconRegion("ranged region",
                UUID.fromString("B9407F30-F5F8-466E-AFF9-25556B57FE6D"), null, null);

        beaconManager = new BeaconManager(this);
        beaconManager.setRangingListener(new BeaconManager.BeaconRangingListener() {
            @Override
            public void onBeaconsDiscovered(BeaconRegion region, List<Beacon> list) {
                if (!list.isEmpty()) {
                    clear();
                    for (Beacon item : list) {
                        switch (item.getMinor()) {
                            case 2224:
                                one = computeAccuracy(item);
```

```
                    break;
                case 46152:
                    two = computeAccuracy(item);
                    break;
                case 41111:
                    three = computeAccuracy(item);
                    break;
                case 48918:
                    four = computeAccuracy(item);
                    break;
                case 48677:
                    five = computeAccuracy(item);
                    break;
            }
        }
    }
  }
});

final Spinner dropdown = findViewById(R.id.spinner1);
Integer[] items = new Integer[]{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
ArrayAdapter<Integer> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, items);
dropdown.setAdapter(adapter);

final Button button = findViewById(R.id.bnCalibrate);
button.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    Intent replyIntent = new Intent();
    Integer selectedNumber = (Integer) dropdown.getSelectedItem();

    if (isEmpty()) {
      setResult(RESULT_CANCELED, replyIntent);
    } else {
      replyIntent.putExtra("node", selectedNumber);
      replyIntent.putExtra("1", round(one));
      replyIntent.putExtra("2", round(two));
      replyIntent.putExtra("3", round(three));
      replyIntent.putExtra("4", round(four));
      replyIntent.putExtra("5", round(five));
      replyIntent.putExtra("6", round(six));
      setResult(RESULT_OK, replyIntent);
    }
    finish();
  }
});


}

@Override
protected void onResume() {
  super.onResume();

  SystemRequirementsChecker.checkWithDefaultDialogs(this);
```

```
        beaconManager.connect(new BeaconManager.ServiceReadyCallback() {
            @Override
            public void onServiceReady() {
                beaconManager.startRanging(region);
            }
        });
    }

    @Override
    protected void onPause() {
        beaconManager.stopRanging(region);
        super.onPause();
    }

    private Boolean isEmpty(){
        return one == 100.0 && two == 100.0 && three == 100.0 && four == 100.0 && five == 100.0
&& six == 100.0;
    }

    private void clear() {
        one = 100.0;
        two = 100.0;
        three = 100.0;
        four = 100.0;
        five = 100.0;
        six = 100.0;
    }

    public static double round(double value) {
        BigDecimal bd = new BigDecimal(value);
        bd = bd.setScale(2, RoundingMode.HALF_UP);
        return bd.doubleValue();
    }
}
```

Node

```
package com.zzz.prpp.thesis_v01;

import android.arch.persistence.room.ColumnInfo;
import android.arch.persistence.room.Entity;

import android.arch.persistence.room.PrimaryKey;

import android.support.annotation.NonNull;

@Entity(tableName="Node_table")
public class Node {

    @PrimaryKey(autoGenerate = true)
    private Integer mId;
    @NonNull
    @ColumnInfo(name = "node")
    private Integer mNumber;
    private Double mOne;
    private Double mTwo;
    private Double mThree;
    private Double mFour;
    private Double mFive;
    private Double mSix;


    //constructors
    public Node(@NonNull Integer number, Double one, Double two, Double three, Double four,
Double five, Double six) {
        this.mNumber = number;
        this.mOne = one;
        this.mTwo = two;
        this.mThree = three;
        this.mFour = four;
        this.mFive = five;
        this.mSix = six;
    }


    public String getNode(){
        if (mOne==null)
            mOne= 10.0;
        if (mTwo==null)
            mTwo= 10.0;
        if (mThree==null)
            mThree= 10.0;
        if (mFour==null)
            mFour= 10.0;
        if (mFive==null)
            mFive= 10.0;
        if (mSix==null)
            mSix= 10.0;

        return mNumber.toString() + "-" + mOne.toString() + "," + mTwo.toString() + "," +
mThree.toString() + "," + mFour.toString() + "," + mFive.toString() + "," + mSix.toString() ; }
```

```
    //getters

    public Integer getMId() {return this.mId;}
    @NonNull
    public Integer getMNumber() {return this.mNumber;}
    public Double getMOne() {return this.mOne;}
    public Double getMTwo() {return this.mTwo;}
    public Double getMThree() {return this.mThree;}
    public Double getMFour() {return this.mFour;}
    public Double getMFive() {return this.mFive;}
    public Double getMSix() {return this.mSix;}

    //setters
    public void setMId(Integer mId) {this.mId = mId;}
    public void setMNumber(@NonNull Integer mNumber) {this.mNumber = mNumber;}
    public void setMOne(Double mOne) {this.mOne = mOne;}
    public void setMTwo(Double mTwo) {this.mTwo = mTwo;}
    public void setMThree(Double mThree) {this.mThree = mThree;}
    public void setMFour(Double mFour) {this.mFour = mFour;}
    public void setMFive(Double mFive) {this.mFive = mFive;}
    public void setMSix(Double mSix) {this.mSix = mSix;}

}
```

NodeDao

```
package com.zzz.prpp.thesis_v01;

import android.arch.lifecycle.LiveData;
import android.arch.persistence.room.Dao;
import android.arch.persistence.room.Insert;
import android.arch.persistence.room.Query;

import java.util.List;

@Dao
public interface NodeDao {

    @Insert
    void insert(Node node);

    @Query("DELETE FROM node_table")
    void deleteAll();

    @Query("SELECT * from node_table ORDER BY mId ASC")
    LiveData<List<Node>> getAllNodes();
}
```

NodeListActivity

```
package com.zzz.prpp.thesis_v01;

import android.arch.lifecycle.Observer;
import android.arch.lifecycle.ViewModelProviders;
import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.design.widget.FloatingActionButton;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

import java.util.List;


public class NodeListActivity extends AppCompatActivity {

    public static final int NEW_NODE_ACTIVITY_REQUEST_CODE = 1;

    private NodeViewModel mNodeViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_node_list);

        RecyclerView recyclerView = findViewById(R.id.recyclerview);
        final NodeListAdapter adapter = new NodeListAdapter(this);
        recyclerView.setAdapter(adapter);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        // Get a new or existing ViewModel from the ViewModelProvider.
        mNodeViewModel = ViewModelProviders.of(this).get(NodeViewModel.class);

        // Add an observer on the LiveData returned by getAllNodes.
        // The onChanged() method fires when the observed data changes and the activity is
        // in the foreground.
        mNodeViewModel.getAllNodes().observe(this, new Observer<List<Node>>() {
            @Override
            public void onChanged(@Nullable final List<Node> nodes) {
                // Update the cached copy of the nodes in the adapter.
                adapter.setNodes(nodes);
            }
        });

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(NodeListActivity.this, NewNodeActivity.class);
```

```
            startActivityForResult(intent, NEW_NODE_ACTIVITY_REQUEST_CODE);
        }
    });
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        return true;
    }
    return super.onOptionsItemSelected(item);
}

public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == NEW_NODE_ACTIVITY_REQUEST_CODE && resultCode ==
RESULT_OK) {
        //Word word = new Word(data.getStringExtra(NewWordActivity.EXTRA_REPLY));
        Node finalNode = new Node(data.getIntExtra("node",
99),data.getDoubleExtra("1",99.0),data.getDoubleExtra("2",99.0),data.getDoubleExtra("3",99.0),dat
a.getDoubleExtra("4",99.0),data.getDoubleExtra("5",99.0),data.getDoubleExtra("6",99.0));
        mNodeViewModel.insert(finalNode);
        Toast.makeText(
                getApplicationContext(),
                R.string.button_calibrate,
                Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(
                getApplicationContext(),
                R.string.empty_not_saved,
                Toast.LENGTH_LONG).show();
    }
}
}
```

NodeListAdapter

```
package com.zzz.prpp.thesis_v01;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import java.util.List;


public class NodeListAdapter extends RecyclerView.Adapter<NodeListAdapter.NodeViewHolder>
{
    class NodeViewHolder extends RecyclerView.ViewHolder {
        private final TextView nodeItemView;

        private NodeViewHolder(View itemView) {
            super(itemView);
            nodeItemView = itemView.findViewById(R.id.textView);
        }
    }

    private final LayoutInflater mInflater;
    private List<Node> mNodes; // Cached copy of nodes

    NodeListAdapter(Context context) { mInflater = LayoutInflater.from(context); }

    @Override
    public NodeViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View itemView = mInflater.inflate(R.layout.recyclerview_item, parent, false);
        return new NodeViewHolder(itemView);
    }

    @Override
    public void onBindViewHolder(NodeViewHolder holder, int position) {
        Node current = mNodes.get(position);
        holder.nodeItemView.setText(current.getNode());
    }

    void setNodes(List<Node> nodes){
        mNodes = nodes;
        notifyDataSetChanged();
    }

    @Override
    public int getItemCount() {
        if (mNodes != null)
            return mNodes.size();
        else return 0;
    }
}
```

NodeRepository

```
package com.zzz.prpp.thesis_v01;

import android.app.Application;
import android.arch.lifecycle.LiveData;
import android.os.AsyncTask;

import java.util.List;

public class NodeRepository {

    private NodeDao mNodeDao;
    private LiveData<List<Node>> mAllNodes;

    NodeRepository(Application application) {
        NodeRoomDatabase db = NodeRoomDatabase.getDatabase(application);
        mNodeDao = db.nodeDao();
        mAllNodes = mNodeDao.getAllNodes();
    }

    LiveData<List<Node>> getAllNodes() {
        return mAllNodes;
    }

    public void insert (Node node) {
        new insertAsyncTask(mNodeDao).execute(node);
    }

    private static class insertAsyncTask extends AsyncTask<Node, Void, Void> {

        private NodeDao mAsyncTaskDao;

        insertAsyncTask(NodeDao dao) {
            mAsyncTaskDao = dao;
        }

        @Override
        protected Void doInBackground(final Node... params) {
            mAsyncTaskDao.insert(params[0]);
            return null;
        }
    }
}
```

NodeRoomDatabase

```
package com.zzz.prpp.thesis_v01;

import android.arch.persistence.db.SupportSQLiteDatabase;
import android.arch.persistence.room.Database;
import android.arch.persistence.room.Room;
import android.arch.persistence.room.RoomDatabase;
import android.content.Context;
import android.os.AsyncTask;
import android.support.annotation.NonNull;

@Database(entities = {Node.class, Coordinate.class}, version = 1, exportSchema = false)
public abstract class NodeRoomDatabase extends RoomDatabase {

   public abstract NodeDao nodeDao();
   public abstract CoordinateDao coordinateDao();

   private static NodeRoomDatabase INSTANCE;


   static NodeRoomDatabase getDatabase(final Context context) {
      if (INSTANCE == null) {
         synchronized (NodeRoomDatabase.class) {
            if (INSTANCE == null) {
               INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                     NodeRoomDatabase.class, "node_database")
                     .fallbackToDestructiveMigration()
                     .addCallback(sRoomDatabaseCallback)
                     .build();

            }
         }
      }
      return INSTANCE;
   }

   private static RoomDatabase.Callback sRoomDatabaseCallback = new
RoomDatabase.Callback(){

      @Override
      public void onOpen (@NonNull SupportSQLiteDatabase db){
         super.onOpen(db);
         // If you want to keep the data through app restarts,
         // comment out the following line.
         new PopulateDbAsync(INSTANCE).execute();
      }
   };

   private static class PopulateDbAsync extends AsyncTask<Void, Void, Void> {

      private final NodeDao mDao;
      private final CoordinateDao mCoordinateDao;

      PopulateDbAsync(NodeRoomDatabase db) {
         mDao = db.nodeDao();
```

```
            mCoordinateDao = db.coordinateDao();
        }

        @Override
        protected Void doInBackground(final Void... params) {

            //Coordinates
            mCoordinateDao.deleteAll();
            mCoordinateDao.insert(new Coordinate(1,5,2));
            mCoordinateDao.insert(new Coordinate(2,5,3));
            mCoordinateDao.insert(new Coordinate(3,7,3));
            mCoordinateDao.insert(new Coordinate(4,1,4));
            mCoordinateDao.insert(new Coordinate(5,1,5));
            mCoordinateDao.insert(new Coordinate(6,3,5));
            mCoordinateDao.insert(new Coordinate(7,5,5));
            mCoordinateDao.insert(new Coordinate(8,7,5));
            mCoordinateDao.insert(new Coordinate(9,9,5));
            mCoordinateDao.insert(new Coordinate(10,5,6));
            mCoordinateDao.insert(new Coordinate(11,7,6));
            mCoordinateDao.insert(new Coordinate(12,9,6));
            mCoordinateDao.insert(new Coordinate(13,5,7));
            mCoordinateDao.insert(new Coordinate(14,7,7));
            mCoordinateDao.insert(new Coordinate(15,5,8));
            mCoordinateDao.insert(new Coordinate(16,7,8));

            //Nodes
            //mDao.deleteAll();

            return null;
        }
    }
}
```

NodeViewModel

```
package com.zzz.prpp.thesis_v01;

import android.app.Application;
import android.arch.lifecycle.AndroidViewModel;
import android.arch.lifecycle.LiveData;

import java.util.List;

public class NodeViewModel extends AndroidViewModel {

    private NodeRepository mRepository;
    private CoordinateRepository mCoordinateRepository;

    private LiveData<List<Node>> mAllNodes;
    private LiveData<List<Coordinate>> mAllCoordinates;

    public NodeViewModel (Application application) {
        super(application);
        mRepository = new NodeRepository(application);
        mCoordinateRepository = new CoordinateRepository(application);
        mAllNodes = mRepository.getAllNodes();
        mAllCoordinates = mCoordinateRepository.getmAllCoordinates();
    }

    LiveData<List<Node>> getAllNodes() { return mAllNodes; }
    LiveData<List<Coordinate>> getAllCoordinates() { return mAllCoordinates;}

    public void insert(Node node) { mRepository.insert(node); }
    public void insert(Coordinate coordinate) { mCoordinateRepository.insert(coordinate);}
}
```

Vertex

```
package com.zzz.prpp.thesis_v01;

public class Vertex {
   final private int id;
   final private float mX;
   final private float mY;


   public Vertex(Integer id, Float x, Float y) {
      this.id = id;
      this.mX = x;
      this.mY = y;
   }
   public Integer getId() {
      return id;
   }

   public Float getMX() {
      return mX;
   }

   public Float getMY() {
      return mY;
   }

}
```

Xml

Activity_calibration

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/map">

    <Button
        android:id="@+id/bnToNodeList"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="110dp"
        android:layout_marginStart="110dp"
        android:text="@string/button_to_node_list"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.zzz.prpp.thesis_v01.DrawNodesView
        android:id="@+id/dvDrawNodes"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>


</android.support.constraint.ConstraintLayout>
```

Activity_main

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/bnToCalibration"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="110dp"
        android:layout_marginEnd="110dp"
        android:layout_marginBottom="40dp"
        android:text="@string/button_to_calibration"
        app:layout_constraintBottom_toTopOf="@+id/bnToNavigation"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/bnToNavigation"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="110dp"
        android:layout_marginEnd="110dp"
        android:layout_marginBottom="40dp"
        android:text="@string/button_to_navigation"
        app:layout_constraintBottom_toTopOf="@+id/tvCenter"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/tvCenter"
        android:layout_width="wrap_content"
        android:layout_height="18dp"
        android:text="@string/welcome"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/tvCenter2"
        app:layout_constraintHorizontal_bias="0.51"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.625" />

    <TextView
        android:id="@+id/tvCenter2"
        android:layout_width="250dp"
        android:layout_height="50dp"
        android:layout_marginBottom="8dp"
        android:text="@string/github"
```

```
        android:textAlignment="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.888" />

</android.support.constraint.ConstraintLayout>
```

Activity_navigation

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@drawable/map">

   <Button
      android:id="@+id/bnIncreasePosition"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_marginBottom="55dp"
      android:layout_marginEnd="110dp"
      android:layout_marginLeft="110dp"
      android:layout_marginRight="110dp"
      android:layout_marginStart="110dp"
      android:text="@string/button_to_calibration"/>

   <com.zzz.prpp.thesis_v01.DrawView
      android:id="@+id/dvNodes"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>

</android.support.constraint.ConstraintLayout>
```

Activity_new_node

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Spinner
        android:id="@+id/spinner1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/btn_dropdown"
        android:spinnerMode="dropdown"/>

    <Button
        android:id="@+id/bnCalibrate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/colorPrimary"
        android:text="@string/button_calibrate"
        android:textColor="@color/buttonLabel" />

</LinearLayout>
```

Activity_node_list

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.zzz.prpp.thesis_v01.NodeListActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_main" />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        android:src="@drawable/ic_add_black_24dp" />

</android.support.design.widget.CoordinatorLayout>
```

Content_main

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.zzz.prpp.thesis_v01.NodeListActivity"
    tools:showIn="@layout/activity_main">
    <!--app:layout_behavior="@string/appbar_scrolling_view_behavior" -->

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/darker_gray"
        tools:listitem="@layout/recyclerview_item" />

</android.support.constraint.ConstraintLayout>
```

Recyclervuew_item

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/textView"
        style="@style/node_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:color/holo_orange_light" />
</LinearLayout>
```

Menu_main

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.zzz.prpp.thesis_v01.NodeListActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```

Colors

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="buttonLabel">#d3d3d3</color>
</resources>
```

Dimens

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <dimen name="fab_margin">16dp</dimen>
    <dimen name="small_padding">6dp</dimen>
    <dimen name="big_padding">16dp</dimen>
</resources>
```

Strings

```
<resources>
    <string name="app_name">thesis-v0.1</string>
    <string name="action_settings">Settings</string>
    <string name="hint_node">Node…</string>
    <string name="button_to_calibration">Calibrations</string>
    <string name="button_to_navigation">Navigate</string>
    <string name="button_calibrate">Calibrate</string>
    <string name="empty_not_saved">Node not saved because it is empty.</string>
    <string name="button_to_node_list">Show Calibrations</string>
    <string name="welcome">Welcome to the thesis application of Nick Androulakis</string>
    <string name="github">Find the code on my github page
https://github.com/NickAndroulakis</string>
</resources>
```

Styles

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>

    <style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar"
/>

    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

    <!-- The default font for RecyclerView items is too small.
    The margin is a simple delimiter between the words. -->
    <style name="node_title">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">26dp</item>
        <item name="android:textSize">24sp</item>
        <item name="android:textStyle">bold</item>
        <item name="android:layout_marginBottom">6dp</item>
        <item name="android:paddingLeft">8dp</item>
    </style>

</resources>
```

Βιβλιογραφία

Chang, N., Rashidzadeh, R., & Ahmadi, M. (2010). Robust indoor positioning using differential wi-fi access points. *IEEE Transactions on Consumer Electronics*, *56*(3), 1860–1867.

Chiou, Y.-S., Wang, C.-L., & Yeh, S.-C. (2010). An adaptive location estimator using tracking algorithms for indoor WLANs. *Wireless Networks*, *16*(7), 1987–2012.

Curran, K., Furey, E., Lunney, T., Santos, J., Woods, D., & McCaughey, A. (2011). An evaluation of indoor location determination technologies. *Journal of Location Based Services*, *5*(2), 61–78.

Furey, E., Curran, K., & McKevitt, P. (2012). HABITS: a Bayesian filter approach to indoor tracking and location. *International Journal of Bio-Inspired Computation*, *4*(2), 79.

Hile, H., & Borriello, G. (2008). Positioning and Orientation in Indoor Environments Using Camera Phones. *IEEE Computer Graphics and Applications*, *28*(4), 32–39.

Lim, H., Kung, L.-C., Hou, J. C., & Luo, H. (2008). Zero-configuration indoor localization over IEEE 802.11 wireless infrastructure. *Wireless Networks*, *16*(2), 405–420.

Liu, X., Xiaohan, L. I. U., Makino, H., & Mase, K. (2010). Improved Indoor Location Estimation Using Fluorescent Light Communication System with a Nine-Channel Receiver. *IEICE Transactions on Communications*, *E93-B*(11), 2936–2944.

Pontes, E., Maciel, T. F., & Linhares, A. (2015). Wireless Sensor Networks: Lifetime Extension and Node Localization and its Potential Application in Agriculture. *Revista de Tecnologia Da Informação E Comunicação*, *5*(1), 1–7.

Qiu, C., & Mutka, M. W. (2016). CRISP: cooperation among smartphones to improve indoor position information. *Wireless Networks*, *24*(3), 867–884.

Qiu, C., & Mutka, M. W. (2017). Silent whistle: Effective indoor positioning with assistance from acoustic sensing on smartphones. In *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. https://doi.org/10.1109/wowmom.2017.7974312

Reza, A. W., & Geok, T. K. (2008). Investigation of Indoor Location Sensing via RFID Reader Network Utilizing Grid Covering Algorithm. *Wireless Personal Communications*, *49*(1), 67–80.

Zhou, Y., Law, C. L., Guan, Y. L., & Chin, F. (2011). Indoor Elliptical Localization Based on Asynchronous UWB Range Measurement. *IEEE Transactions on Instrumentation and Measurement*, *60*(1), 248–257.