ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Αυτόματη ξενάγηση πεζοπορίας με τεχνικές επαυξημένης πραγματικότητας (θέσης)»

Της φοιτήτριας
Δέσποινας Αθανασλέρη
Αρ. Μητρώου: 154427

Επιβλέπων
Ευκλείδης Κεραμόπουλος
Αναπληρωτής Καθηγητής

Ημερομηνία 29/01/2022

Τίτλος Δ.Ε. **Αυτόματη ξενάγηση πεζοπορίας με τεχνικές επαυξημένης πραγματικότητας (θέσης)**
Κωδικός Δ.Ε. **21100**
Ονοματεπώνυμο φοιτήτριας **Δέσποινα Αθανασλέρη**
Ονοματεπώνυμο εισηγητή **Ευκλείδης Κεραμόπουλος**
Ημερομηνία ανάληψης Δ.Ε. **05-01-2021**
Ημερομηνία περάτωσης Δ.Ε. **29-01-2022**

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Σε όσους πιστεύουν σε μένα»

1. Πρόλογος

Η αγάπη μου για τα ταξίδια, τους μεγάλους περιπάτους και την Θεσσαλονίκη αποτέλεσαν πηγή έμπνευσης για την σύλληψη αυτής της ιδέας και τελικά την δημιουργία μιας ταξιδιωτικής εφαρμογής.

Ο τρόπος που ταξιδεύει πλέον ο κόσμος είναι πιο αυτόνομος, δεν απευθύνεται σε ταξιδιωτικά γραφεία και ξεναγούς. Σε αυτό, έχουν συμβάλει σημαντικά τα έξυπνα τηλέφωνα και η εύκολη πρόσβαση στο διαδίκτυο, η πληροφορία είναι άμεσα διαθέσιμη με μια αναζήτηση.

Όμως, ένα πρόβλημα που αντιμετωπίζω όταν επισκέπτομαι μια πόλη για πρώτη φορά είναι ότι δεν γνωρίζω ποια είναι τα σημαντικότερα αξιοθέατα, πόσο κοντά είναι το ένα στο άλλο και τελικά τι είναι αυτό που βλέπω μπροστά μου. Αυτές οι πληροφορίες είναι όλες διαθέσιμες στο διαδίκτυο αλλά πολλές φορές είναι δύσκολο να τις βρω κάπου μαζεμένες και οργανωμένες σε ένα σημείο.

Κατά της διάρκεια της καραντίνας, περπατώντας μέσα στην πόλη είχα την ευκαιρία να την γνωρίσω καλύτερα. Αυτό το διάστημα συνειδητοποίησα πως δεν υπάρχει καλύτερος τρόπος για να γνωρίσεις και να ανακαλύψεις μια πόλη, τους ανθρώπους της, τον ρυθμό της και τις συνήθειες των κατοίκων της, από το να γυρνάς μέσα σε αυτή με τα πόδια.

Κάπως έτσι προέκυψε η ιδέα για μια ταξιδιωτική εφαρμογή που οι χρήστες θα έχουν την ευκαιρία να γνωρίσουν τα μνημεία και την ιστορία της πόλης περπατώντας μέσα σε αυτή και προτείνοντας και διαδρομές με ταξιδιωτικό ενδιαφέρον.

2. Περίληψη

Ο στόχος αυτής της εργασίας είναι η δημιουργία μιας εφαρμογής για κινητά Android, που θα δίνει την δυνατότητα στον χρήστη να περιηγηθεί στην Θεσσαλονίκη και να γνωρίσει τα μνημεία και την ιστορία της πόλης μέσα από διάφορες πεζοπορικές διαδρομές. Κάθε διαδρομή θα αποτελείται από κάποια σημεία ενδιαφέροντος και όταν ο χρήστης πλησιάζει σε ένα από τα σημεία αυτά ακολουθώντας την προτεινόμενη διαδρομή, θα εμφανίζονται πληροφορίες για το συγκεκριμένο σημείο, ενώ πατώντας πάνω στο παράθυρο πληροφοριών θα μεταφέρεται σε μια σελίδα με περισσότερες λεπτομέρειες για το σημείο αυτό.

Πέρα από αυτή την βασική λειτουργία της εφαρμογής ο χρήστης έχει πρόσβαση σε μία λίστα με τα μνημεία της πόλης και όταν επιλέγει κάποιο από αυτά μεταφέρεται στην παραπάνω σελίδα με τις λεπτομέρειες για το συγκεκριμένο μνημείο.

«Hiking tour with augmented reality (location) techniques»

«Despoina Athanasleri»

3. Abstract

The aim of this thesis is to create an application for Android smartphones, which will give the opportunity to the user to walk around Thessaloniki and get to know the city, the monuments, and the history through various hiking trails. Each route will contain some points of interest (monuments, squares, gardens, churches etc.).

While the user follows the suggested route, information about that spot will be displayed in an Info Window, which pops up when the user approach and if the user clicks on that Info Window a new Activity appears with more details about that spot.

Apart from this basic functionality of the application, the user has access to a list of the points of interest of the city and when the user selects one of them, he/she is transferred to the details page mentioned above.

4. Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Κεραμόπουλο που με εμπιστεύτηκε για την ανάληψη αυτής της εργασίας και ήταν διαθέσιμος όποτε χρειαζόταν. Καθώς και όσους με στήριξαν κατά την διάρκεια των σπουδών μου για να φτάσω εδώ που βρίσκομαι σήμερα.

5. Περιεχόμενα

6. Κατάλογος Εικόνων

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Αυτή η Διπλωματική Εργασία έχει ως στόχο την δημιουργία ενός τουριστικού οδηγού για ανθρώπους που θέλουν να γνωρίσουν την Θεσσαλονίκη, ιστορικά, κοινωνικά και πολιτισμικά. Ο χρήστης έχει την δυνατότητα να περιηγηθεί μέσα στην πόλη και να μάθει για αυτή. Ιδιαίτερη βαρύτητα δόθηκε στην λειτουργία επαύξησης περιεχομένου του χάρτη την στιγμή που η τοποθεσία του χρήστη απέχει λιγότερο από 20 μέτρα από το σημείο ενδιαφέροντος.

1.2 Δομή Διπλωματικής Εργασίας

Στο πρώτο κεφάλαιο της εργασίας παρουσιάζεται εν συντομία η δομή της Διπλωματικής Εργασίας αλλά και μια έρευνα που εκπονήθηκε από το TURKU UNIVERSITY OF APPLIED SCIENCES και αφορά την σχέση που έχουν οι νέοι με τα ταξίδια καθώς και τα συμπεράσματα που προκύπτουν από την έρευνα αυτή.

Στο δεύτερο κεφάλαιο της εργασίας αναλύεται η πλατφόρμα του Android και οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Αναλύονται επίσης τα στοιχεία από τα οποία αποτελείται μια Android εφαρμογή, πώς συνδέονται αυτά μεταξύ τους και τι πρέπει να γνωρίζει ένας προγραμματιστής όταν χτίζει μια εφαρμογή Android.

Παρακάτω αναλύεται η δομή και η λειτουργικότητα της εφαρμογής Android. Σε αυτό το σημείο δεν αναπτύσσονται τα τεχνικά χαρακτηριστικά της εφαρμογής αλλά οι δυνατότητες και τα οφέλη που έχει ο χρήστης από την εφαρμογή αυτή. Ο τρόπος που μπορεί να πλοηγηθεί μέσα στην εφαρμογή και οι διάφορες διεπαφές που συναντάει ο χρήστης κατά την χρήση της.

Στο προτελευταίο και μεγαλύτερο κεφάλαιο της εφαρμογής αναλύεται λεπτομερώς κάθε γραμμή κώδικα που παρουσιάζει προγραμματιστικό ενδιαφέρον καθώς φαίνεται και η δομή της εφαρμογής μέσα στο Android Studio.

Ενώ στο πέμπτο και τελευταίο κεφάλαιο αναφέρονται οι σκέψεις και τα συμπεράσματά μου για τις τεχνολογίες που χρησιμοποιήθηκαν ενώ παρουσιάζονται και ορισμένες προοπτικές για το μέλλον της εφαρμογής.

1.3 Έρευνα

Ο κόσμος και οι συνήθειές του αλλάζουν με μεγάλες ταχύτητες λόγω της τεχνολογικής ανάπτυξης. Η έρευνα που πραγματοποιήθηκε στο TURKU UNIVERSITY OF APPLIED SCIENCES αποδεικνύει ότι ο τρόπος που ταξιδεύει ο κόσμος είναι πιο αυτόνομος. Δηλαδή δεν βασίζεται στον ίδιο βαθμό και για τις ίδιες υπηρεσίες σε τουριστικά γραφεία, παρά μόνο για να εξασφαλίσουν την ασφάλεια των χρημάτων τους καθώς και για πιο περίπλοκα ταξίδια που δεν είναι εύκολο να οργανωθούν μέσω των διαθέσιμων ιστοσελίδων [1].

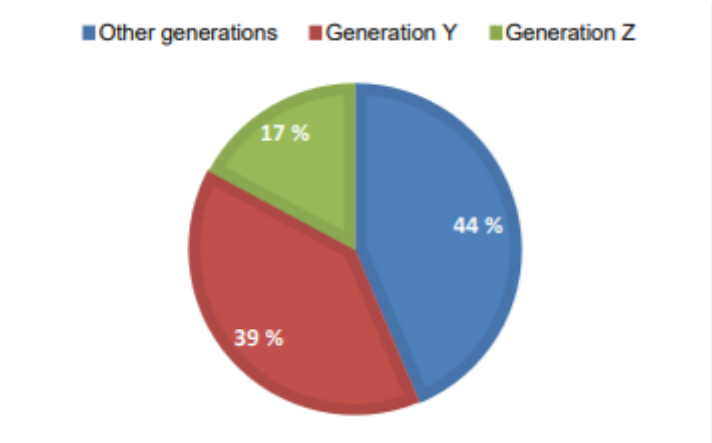
Η παρακάτω έρευνα εστίασε περισσότερο στις ηλικιακές ομάδες των γενιών Y και Z. Δηλαδή Σε ανθρώπους που γεννήθηκαν το διάστημα 1981-1996 και 1997-2012, αντίστοιχα. Αυτές οι δύο ηλικιακές ομάδες είναι περισσότερο εξοικειωμένες με την τεχνολογία από ανθρώπους που είναι γεννημένοι τις

προηγούμενες δεκαετίες ενώ έχουν και σε μεγαλύτερη εκτίμηση τα ταξίδια. Πιο συγκεκριμένα, οι Ευρωπαίοι πολίτες της γενιάς Y ταξιδεύουν κατά μέσο όρο τέσσερις με πέντε φορές τον χρόνο. Ενώ σύμφωνα με την έρευνα της Deloitte [2] όπου συμμετείχαν 16425 άτομα η νούμερο ένα προτεραιότητα των ερωτηθέντων είναι να «δουν και να ταξιδέψουν στον κόσμο».

Όσον αφορά την συμπεριφορά αυτών των δύο γενεών στα ταξίδια, τους αρέσει να ανακαλύπτουν νέα μέρη, να βιώνουν καινούργιες εμπειρίες καθώς αναζητούν την προσωπική ανάπτυξη. Ως ταξιδιώτες είναι ανεξάρτητοι, ανοιχτοί σε νέες, ξεχωριστές, ταξιδιωτικές εμπειρίες και οργανώνουν μόνοι τους τα ταξίδια τους, ενώ προτιμούν πιο «αυθεντικούς» και πρωτότυπους ταξιδιωτικούς προορισμούς [3].

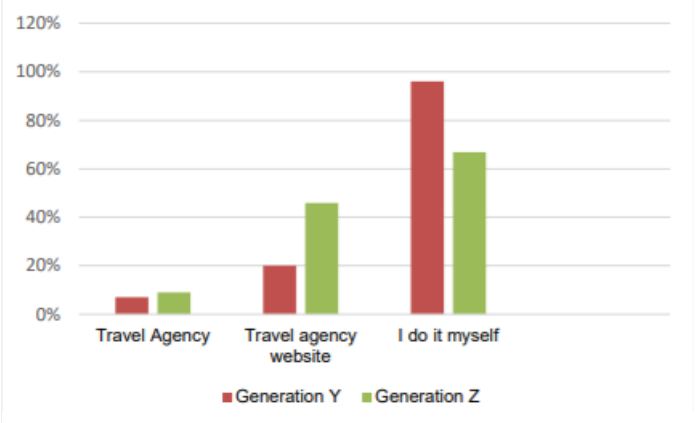
1.4 Αποτελέσματα της έρευνας

Η έρευνα του πανεπιστημίου έλαβε μέρος το διάστημα Οκτωβρίου και Νοεμβρίου 2020. Τα περισσότερα δεδομένα που συλλέχθηκαν είναι από τις ηλικιακές ομάδες που αναφέρθηκαν παραπάνω. Πιο συγκεκριμένα, τα ερωτηματολόγια που απαντήθηκαν ήταν 268, εκ των οποίων τα 105 (36%) απαντήθηκαν από άτομα που ανήκουν στην γενιά Y και 46 (17%) από άτομα της γενιάς Z (Εικόνα 1).



Εικόνα 1: Ηλικιακές ομάδες που συμμετείχαν στην έρευνα [1]

Όταν οι συμμετέχοντες ερωτήθηκαν πως οργανώνουν τα ταξίδια τους η συντριπτική πλειοψηφία της γενιάς Y (96%) απάντησε ότι κλείνουν τα ταξίδια τους μόνοι τους. (Εικόνα 2)



Εικόνα 2: Πως κλείνουν τα ταξίδια τους οι συμμετέχοντες [1]

Στο πλαίσιο της διπλωματικής εργασίας αυτά είναι τα δεδομένα που μας ενδιαφέρουν από την εν λόγο έρευνα [1].

1.5 Συμπεράσματα

Από την έρευνα συμπεραίνουμε ότι τουλάχιστον οι γενιές Y και Z προτιμούν να οργανώνουν τα ταξίδια τους μόνοι τους, χωρίς καμία καθοδήγηση από κάποιο τουριστικό γραφείο. Για αυτόν ακριβώς τον λόγο θεώρησα απαραίτητη την ύπαρξη μίας εφαρμογής που μπορεί να πάρει τον ρόλο του ξεναγού μέσα στην πόλη. Για να μπορέσει να την γνωρίσει καλύτερα και κάποιος που δεν έχει κλείσει το ταξίδι του μέσω κάποιου τουριστικού γραφείου ή που δεν έχει την οικονομική δυνατότητα να μισθώσει ξεναγό.

Κεφάλαιο 2ο: Τεχνολογίες που Χρησιμοποιήθηκαν

2.1 Εισαγωγή

Για την εκπόνηση αυτής της εργασίας χρησιμοποιήθηκαν κάποιες βασικές τεχνολογίες και εργαλεία. Συνοπτικά, η ανάπτυξη της εφαρμογής έγινε στο Android Studio. Για την αποθήκευση των δεδομένων χρησιμοποιήθηκε η διαδικτυακή βάση δεδομένων Firestore της Google. Αξιοποιήθηκαν οι αισθητήρες θέσης της συσκευής για να υλοποιηθεί η βασική λειτουργία της εφαρμογής, που είναι η επαύξηση του χάρτη με βάση την τρέχουσα τοποθεσία του χρήστη και η βιβλιοθήκη Google Maps σε συνδυασμό με το Directions API της Google.

2.2 Λογισμικό Android

Το Android ξεκίνησε σαν ένα open source λειτουργικό σύστημα βασισμένο σε πυρήνα Linux που αναπτύσσεται από την Google, για smartphones και tablets. Όμως τα τελευταία χρόνια όλο και περισσότερες συσκευές υποστηρίζουν το λογισμικό Android. Ορισμένες από αυτές είναι οι έξυπνες τηλεοράσεις (Android TV), τα έξυπνα ρολόγια (Android Wear), ακόμη και οι υπολογιστές αυτοκινήτων (Android Auto) [5].

2.3 Android Studio

Το Android Studio είναι το επίσημο IDE (Integrated Development Environment) για τον προγραμματισμό Android εφαρμογών, βασισμένο στο IntelliJ IDEA [14]. Σχεδιάστηκε αποκλειστικά για την δημιουργία Android εφαρμογών και είναι διαθέσιμο σε Windows, Linux και Mac OS X. Πριν την δημιουργία του Android Studio το κύριο IDE για την ανάπτυξη εφαρμογών Android ήταν το Eclipse Android Development Tools (ADT).

2.4 Αρχιτεκτονική Android

Η αρχιτεκτονική Android αποτελείται από τέσσερα επίπεδα όπως φαίνεται και στην Εικόνα 3.

2.4.1 Πυρήνας Kernel

Ξεκινώντας από κάτω προς τα πάνω, πρώτα συναντάται ο Πυρήνας (Linux Kernel). Ο οποίος μεσολαβεί για την επικοινωνία ανάμεσα στο Software και το Hardware. Χωρίς τον πυρήνα δεν θα ήταν δυνατό να έχουμε σταθερό και αξιόπιστο λειτουργικό σύστημα καθώς είναι υπεύθυνος για την εκτέλεση των βασικών λειτουργιών του συστήματος, όπως η διαχείριση των διεργασιών, η διαχείριση μνήμης, η χρήση υπηρεσιών δικτύωσης, η χρήση οδηγών για διάφορα τμήματα του Hardware κ.α. [5].

2.4.2 Βιβλιοθήκες & Android Runtime

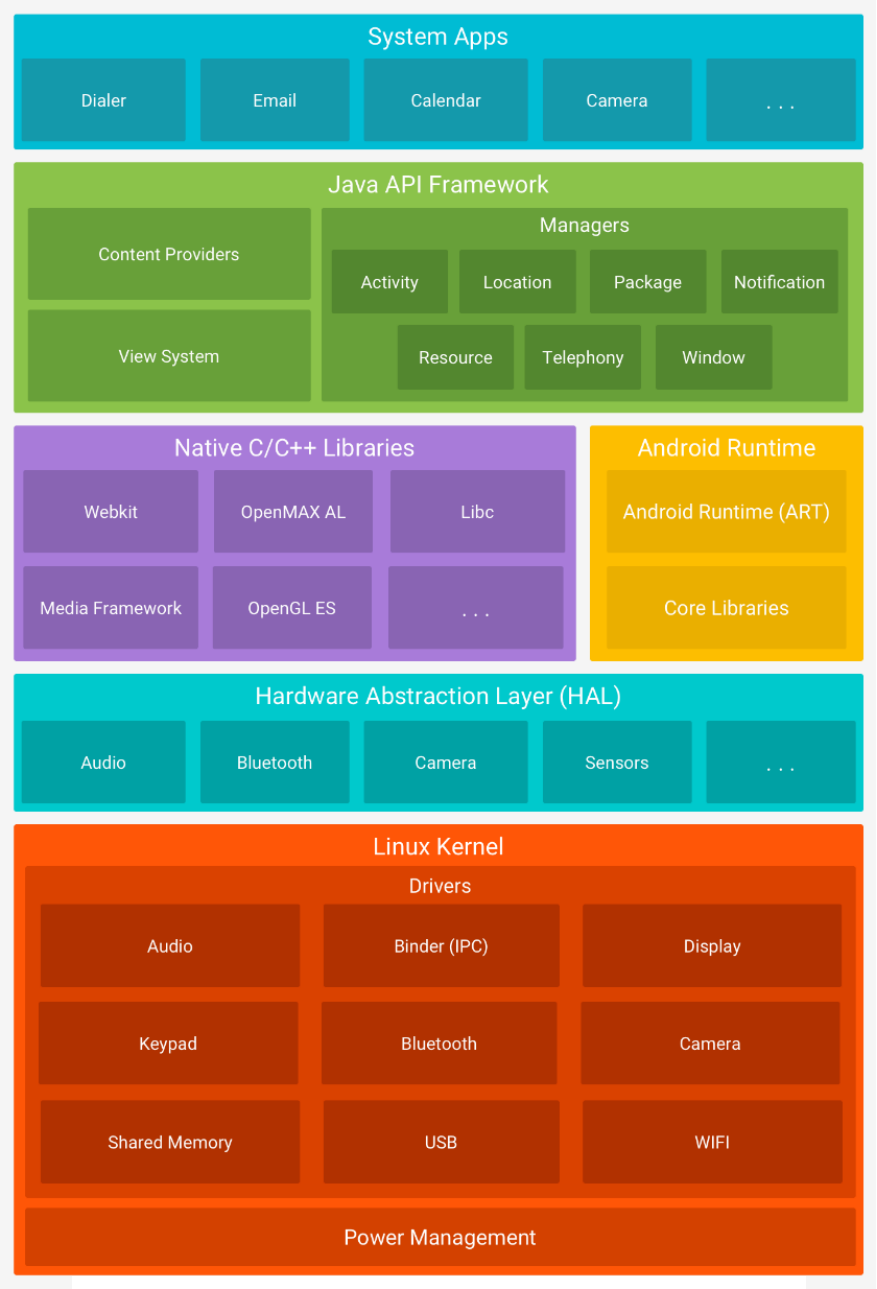
Πάνω από τον πυρήνα βρίσκονται οι βιβλιοθήκες που είναι γραμμένες σε C/C++ οι οποίες χρησιμοποιούνται από διάφορα στοιχεία του συστήματος υψηλότερου επιπέδου.

Στο ίδιο επίπεδο με τις βιβλιοθήκες βρίσκεται και το Android Runtime (χρόνος εκτέλεσης του Android). Εκεί μέχρι το Android 5.0 βρισκόταν η Dalvik Virtual Machine (DVM) η οποία αντικαταστάθηκε από την ART. Η βασική διαφορά ανάμεσα στις δύο εικονικές μηχανές εντοπίζεται στον τρόπο μετάφρασης του κώδικα σε γλώσσα μηχανής. Η πρώτη μεταφράζει την στιγμή που ο χρήστης τρέχει την εκάστοτε εφαρμογή ενώ από την έκδοση Lollipop και μετά η ART μεταφράζει και αποθηκεύει την μετάφραση ώστε να είναι πάντα διαθέσιμη στο λειτουργικό σύστημα (Ahead-Of-Time compiler). Η αλλαγή από

Dalvik σε ART έχει αυξήσει τον χώρο και τον χρόνο που απαιτείται για την εγκατάσταση μιας εφαρμογής όμως έχει και τρία βασικά πλεονεκτήματα:

- Ο απαιτούμενος χρόνος για να ανοίξουν οι εφαρμογές είναι λιγότερος
- Γίνεται καλύτερη διαχείριση των πόρων της συσκευής
- Καταναλώνεται λιγότερη ενέργεια

Εκτός από την εικονική μηχανή, στο Android Runtime, είναι τοποθετημένες και οι βασικές βιβλιοθήκες της Java [4].



Εικόνα 3: Αρχιτεκτονική Android [30]

2.4.3 Application Framework

Το τρίτο επίπεδο είναι το επίπεδο πλαισίου των εφαρμογών, εκεί οι προγραμματιστές έχουν την δυνατότητα να αναπτύξουν τις εφαρμογές τους χρησιμοποιώντας τις δυνατότητες του Android. Παράλληλα έχουν τη δυνατότητα πέρα από το να χρησιμοποιήσουν τα ήδη υπάρχοντα στοιχεία να διευρύνουν το πλαίσιο ανάπτυξης προσθέτοντας δικές τους βιβλιοθήκες. Κάποια βασικά δομικά συστατικά του πλαισίου είναι τα παρακάτω:

- Activity Manager: Ελέγχει τον κύκλο ζωής των εφαρμογών και παρέχει την δυνατότητα διαχείρισης των Activities
- Content Provider: Παρέχει την δυνατότητα διανομής δεδομένων μεταξύ των εφαρμογών της συσκευής
- Resource Manager: Πόροι χαρακτηρίζονται τα στοιχεία μίας εφαρμογής εκτός του κώδικα. Όπως τα Strings, τα χρώματα, οι εικόνες και τα layouts.
- Location Manager: Βρίσκει την τρέχουσα τοποθεσία της συσκευής χρησιμοποιώντας τις υπηρεσίες εντοπισμού θέσης GPS ή και των κεραιών κινητής τηλεφωνίας.
- Notification Manager: Εμφανίζει στη γραμμή κατάστασης ειδοποιήσεις χωρίς να διακόπτει την ροή των δραστηριοτήτων του χρήστη
- Window Manager: Διαχειρίζεται τα ανοιχτά παράθυρα. Κάθε παράθυρο αντιστοιχίζεται με ένα Activity
- Telephony Manager: Μοιράζεται με την εφαρμογή πληροφορίες κλήσεων και οτιδήποτε σχετικό με τις υπηρεσίες τηλεφωνίας
- View System: Περιλαμβάνει τα γραφικά στοιχεία που χρησιμοποιούνται για τη δημιουργία των διεπαφών χρήστη. Τέτοια γραφικά στοιχεία είναι οι λίστες, τα πλέγματα, τα πλαίσια κειμένου, τα κουμπιά κ.α.

2.4.4 Επίπεδο Εφαρμογής

Στο υψηλότερο επίπεδο της αρχιτεκτονικής του Android βρίσκονται οι εφαρμογές που χρησιμοποιεί ο χρήστης. Τόσο οι προεγκαταστημένες όπως η κάμερα, η αριθμομηχανή, οι επαφές ή η συλλογή όσο και αυτές που εγκαθιστά ο χρήστης στη συσκευή από το Play Store, όπως η εφαρμογή του Facebook, του Instagram ή και διάφορα παιχνίδια [5].

2.5 Βασικά Συστατικά Εφαρμογών

Μια εφαρμογή Android αποτελείται από τέσσερα βασικά δομικά στοιχεία.

1. Activities
2. Services
3. Broadcast receivers
4. Content providers

Κάθε τύπος συστατικού εξυπηρετεί έναν ξεχωριστό σκοπό και έχει έναν ξεχωριστό κύκλο ζωής. Ο κύκλος ζωής ενός δομικού συστατικού καθορίζει τον τρόπο με τον οποίο ένα συστατικό «γεννιέται» και «πεθαίνει» [4].

2.6 Activities

Ένα Activity αποτελεί το πιο διαδεδομένο δομικό στοιχείο που συναντάται σε μία εφαρμογή Android καθώς είναι αυτό που αναπαριστά και μία ξεχωριστή οθόνη της εφαρμογής. Όλες οι εφαρμογές Android απαρτίζονται από μια συλλογή από Activities. Τα Activities αυτά οργανώνονται σε μία στοίβα και στην πρώτη θέση της στοίβας βρίσκεται η οθόνη που βλέπει την δεδομένη στιγμή ο χρήστης [4].

Ξεκινώντας μια Android εφαρμογή εμφανίζεται στην οθόνη του χρήστη το πρώτο Activity. Αυτό το Activity συνήθως αναφέρεται ως Main Activity. Η πλειοψηφία των εφαρμογών Android που συναντώνται αποτελούνται με περισσότερες από μία δραστηριότητες (Activities). Οι δραστηριότητες αυτές όπως αναφέρθηκε παραπάνω οργανώνονται σε στοίβες. Όταν ο χρήστης μεταφέρεται σε ένα νέο Activity αυτό ανεβαίνει στην κορυφή την στοίβας και το προηγούμενο Activity «χάνεται». Αυτό συμβαίνει κάθε φορά που ο χρήστης μεταφέρεται σε ένα νέο Activity [4].

Κάθε Activity περιλαμβάνει κάποια Views, αυτά τα Views βοηθάνε τον χρήστη να αλληλοεπιδράσει με τα Activities και κατ' επέκταση με την εφαρμογή. Ένα view μπορεί να είναι ένα κουμπί (Button), που όταν ο χρήστης πατάει πάνω σε αυτό ενεργοποιεί κάποια ενέργεια. Στο Android συναντώνται πολλά έτοιμα Views, τα πιο συνηθισμένα από αυτά είναι το TextView, το Button, το ImageView, το RecyclerView, το FragmentContainerView, το ScrollView και το Switch [4].

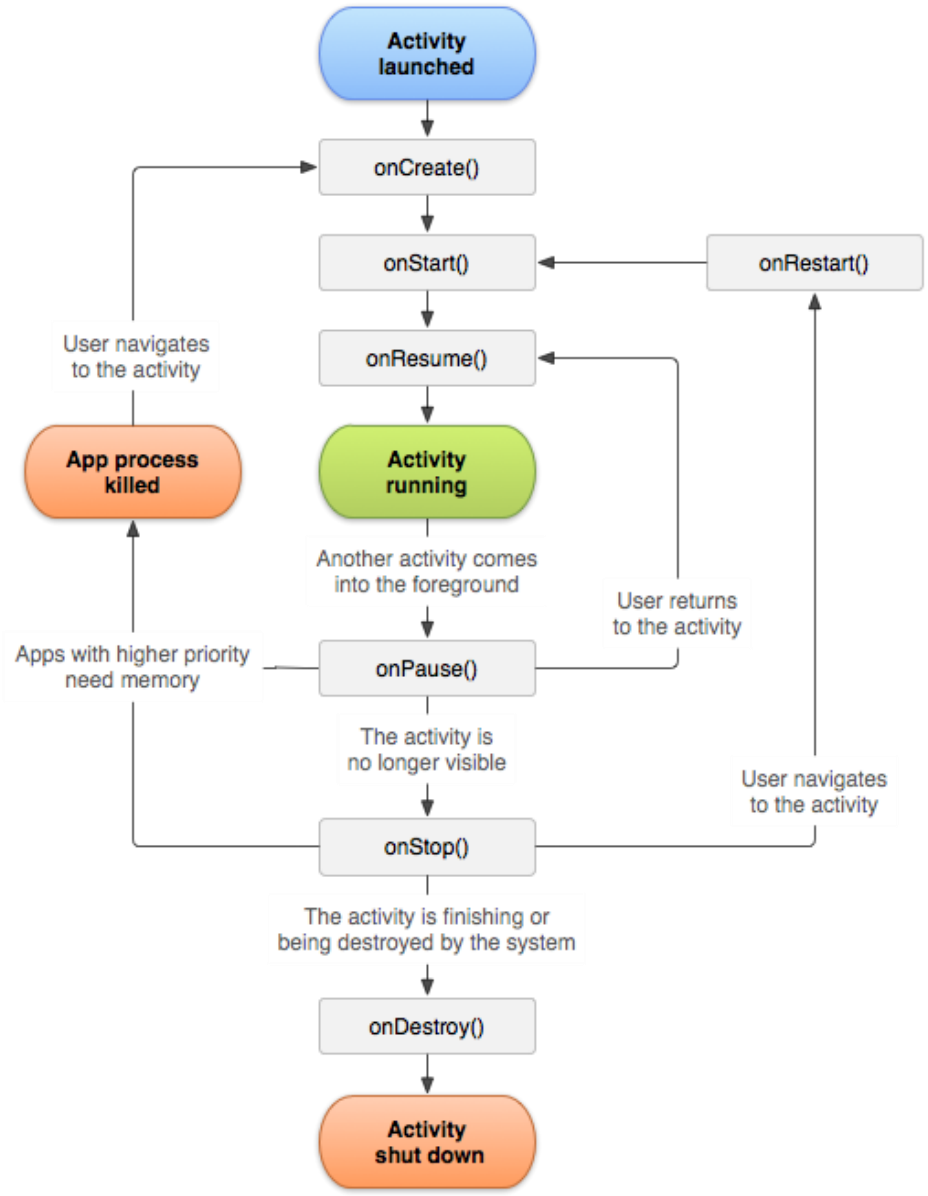
Κάθε Activity ενεργοποιείται με την βοήθεια κάποιων ασύγχρονων μηνυμάτων που ονομάζονται intents. Κάθε intent είναι μια αφηρημένη περιγραφή μίας πράξης που πρόκειται να εκτελεστεί και χρησιμοποιείται συνήθως για την εκκίνηση Activities και Services [4].

2.7 Ο κύκλος ζωής ενός Activity

Ο κύκλος ζωής ενός Activity αποτελείται από τρεις βασικές καταστάσεις. Αυτές φαίνονται και στην Εικόνα 4.

- Activity ενεργό: Σε αυτή την κατάσταση το Activity είναι ενεργό και ορατό στον χρήστη.
- Activity σε παύση: Όταν ένα Activity βρίσκεται σε παύση έχει σταματήσει προσωρινά την λειτουργία του γιατί ένα άλλο Activity βρίσκεται στο προσκήνιο, στην κορυφή της στοίβας.
- Activity σταματημένο: Όταν ένα Activity είναι σταματημένο συνεχίζει να διατηρεί τις πληροφορίες που είχε και να βρίσκεται στην κατάσταση που ήταν πριν σταματήσει αλλά σε περίπτωση που μία άλλη εφαρμογή χρειαστεί την μνήμη που καταλαμβάνει αυτό το Activity τότε αυτό παύει να υφίσταται.

Όπως φαίνεται στην Εικόνα 4, ο κύκλος ζωής ενός Activity ξεκινάει με την μέθοδο onCreate(), όπου ουσιαστικά «γεννιέται» η δραστηριότητα και τελειώνει με την μέθοδο onDestroy() («πεθαίνει»). Ο χρόνος που ένα Activity είναι ορατό ξεκινάει με την μέθοδο onStart() και σταματάει με την μέθοδο onStop(). Ο χρήστης μπορεί να αλληλεπιδράσει με το activity όταν βρίσκεται ανάμεσα στις μεθόδους onResume() και onPause(). Η μέθοδος onPause() καλείται όταν η συσκευή βρίσκεται σε αδράνεια ή όταν ξεκινάει μια άλλη δραστηριότητα. Κατά την επιστροφή στην δραστηριότητα καλείται η μέθοδος onResume() [6].



Εικόνα 4: Ο κύκλος ζωής του Activity [6]

2.8 Services

Μία υπηρεσία μπορεί να εκκινηθεί από κάποιο άλλο δομικό στοιχείο, να την αφήσει να τρέξει, να συνδεθεί σε αυτήν και να αλληλοεπιδράσει μαζί της. Μία υπηρεσία τρέχει στο background, μπορεί ο χρήστης να το γνωρίζει και να μπορεί ανά πάσα στιγμή να την σταματήσει ή να μην το γνωρίζει και να μπορεί να την διαχειριστεί μόνο το σύστημα. Δηλαδή μπορεί να την σταματήσει αν χρειαστεί μνήμη και να την ξεκινήσει ξανά αργότερα [7].

2.9 Broadcast Receiver

Η ενεργοποίηση των Broadcast Receiver γίνεται από το λειτουργικό σύστημα όταν συμβαίνει κάποιο γεγονός. Όπως η ένδειξη χαμηλής μπαταρίας, η ολοκλήρωση της λήψης μίας εφαρμογής ή ενός αρχείου,

μια αναπάντητη κλήση. Ένα Broadcast Receiver μπορεί να ξυπνήσει ένα Activity ή και να βγάλει στον χρήστη μια ειδοποίηση [8].

2.10 Content Provider

Οι Content Provider διαχειρίζονται τα δεδομένα και τα αρχεία που αποθηκεύονται στο σύστημα αρχείων της συσκευής. Αυτό το σύστημα αρχείων μπορεί να είναι σε μία SQLite βάση δεδομένων, στη Firebase και γενικά σε οποιοδήποτε μέσο μόνιμης αποθήκευσης [9].

2.10.1 Firebase Firestore

Για την αποθήκευση του περιεχομένου και των δεδομένων που εμφανίζονται στην εφαρμογή έχει χρησιμοποιηθεί η διαδικτυακή βάση δεδομένων, Google Cloud Firestore. Η επιλογή μιας διαδικτυακής βάσης δεδομένων επιτρέπει την εύκολη και άμεση ανανέωση και αλλαγή των δεδομένων, ενώ επιτρέπει στον χρήστη να έχει πρόσβαση στο ανανεωμένο περιεχόμενο χωρίς να χρειάζεται να ενημερώσει την εφαρμογή. Η Google Cloud Firestore παρέχει άριστη συνεργασία με τις εφαρμογές Android. Είναι μια NoSQL βάση δεδομένων, αυτό σημαίνει ότι τα δεδομένα αποθηκεύονται σε αρχεία που περιέχουν πεδία που αντιστοιχίζονται με τιμές, αυτά τα αρχεία περιέχονται μέσα σε συλλογές. Με αυτό τον τρόπο τα αρχεία οργανώνονται και μπορούν να ανακτηθούν με queries [10].

Κύρια χαρακτηριστικά του Firestore:

- Flexibility (ευελιξία): Το Firestore υποστηρίζει ευέλικτες, ιεραρχικές δομές δεδομένων. Τα δεδομένα μπορούν να αποθηκευτούν σε αρχεία, οργανωμένα σε συλλογές. Τα αρχεία μπορούν να περιέχουν περίπλοκα εμφωλευμένα αντικείμενα σε συνδυασμό με τμήματα των συλλογών.
- Expressive querying (εκφραστικά ερωτήματα): Το Firestore υποστηρίζει ερωτήματα για την ανάκτηση συγκεκριμένων εγγράφων ή για την ανάκτηση όλων των εγγράφων μίας συλλογής που ταιριάζουν στις παραμέτρους του ερωτήματος. Τα ερωτήματα αυτά μπορούν να περιέχουν πολλαπλά και σύνθετα φίλτρα για την ανάκτηση συγκεκριμένης πληροφορίας και για την ταξινόμηση του αποτελέσματος.
- Realtime updates (ενημερώσεις πραγματικού χρόνου): Το Firestore ενημερώνει τα δεδομένα σε όλες τις συνδεδεμένες συσκευές σε πραγματικό χρόνο.
- Offline support (υποστήριξη εκτός δικτύου): Το Firestore αποθηκεύει τα δεδομένα σε κάθε συσκευή, με τέτοιο τρόπο ώστε να λειτουργεί ακόμα και αν δεν υπάρχει σύνδεση στο δίκτυο. Όταν επανέρχεται η σύνδεση δικτύου το Firestore συγχρονίζει όλες τις αλλαγές που μπορεί να έχουν συμβεί.
- Design to scale: Το Google Firestore έχει σχεδιαστεί για να χειρίζεται τους πιο σκληρούς φόρτους εργασίας βάσης δεδομένων από τις μεγαλύτερες εφαρμογές.

2.11 Kotlin

Η Kotlin είναι μια open-source γλώσσα προγραμματισμού που τρέχει σε Java Virtual Machine(JVM), είναι μια αντικειμενοστραφής γλώσσα που υποστηρίζει και functional προγραμματισμό. Είναι μια γλώσσα προγραμματισμού σχετικά καινούργια, αλλά πολύ εξελιγμένη και ολοκληρωμένη. Δημιουργήθηκε από προγραμματιστές της IDE Jet Brains το 2011 ενώ κυκλοφόρησε επίσημα το 2016. Την δεδομένη στιγμή βρίσκεται στην έκδοση 1.6.0. Η Kotlin δεν χρησιμοποιείται μόνο για την δημιουργία Android εφαρμογών, αν και προτείνεται η χρήση Kotlin για την δημιουργία μιας εφαρμογής Android, συναντάται ωστόσο σε Backend Apps, σε Cross-platform Mobile App, σε Front-end web apps σε συνδυασμό με την βοήθεια της React και της Javascript. Αυτά τα χαρακτηριστικά καθιστούν την Kotlin μία γλώσσα που υποστηρίζει Multiplatform προγραμματισμό. Είναι συμβατή με την Java,

δηλαδή με την βοήθεια ενός plugin ένα κομμάτι κώδικα γραμμένο σε Java μετατρέπεται αυτόματα σε Kotlin. Τέλος η Kotlin έχει ανακοινωθεί ως η επίσημη γλώσσα για την ανάπτυξη Android εφαρμογών [11].

2.11.1 Βασικά χαρακτηριστικά της Kotlin

Τα βασικά χαρακτηριστικά της Kotlin είναι:

- Προσφέρει συντομότερη κωδικοποίηση
- Γίνεται compile γρηγορότερα
- Αξιοποιεί το JVM, που συνδυάζει χαρακτηριστικά του αντικειμενοστραφή προγραμματισμού και του functional προγραμματισμού.
- Υποστηρίζει μια ποικιλία από extension functions χωρίς να κάνει καμία αλλαγή στον κώδικα
- Μπορεί κάποιος να γράψει Kotlin κώδικα χρησιμοποιώντας κάποιο IDE ή CLI (command line interface)
- Το Casting βοηθάει τον προγραμματιστή να μειώσει το κόστος της εφαρμογής και να βελτιώσει την ταχύτητα και την απόδοσή της.

2.12 Java

Η Java είναι και αυτή μια multiplatform, αντικειμενοστραφής γλώσσα. Είναι από τις πιο διαδεδομένες γλώσσες προγραμματισμού και η ιστορία της ξεκίνησε το 1995. Δημιουργήθηκε από την Sun Microsystems όμως αργότερα το 2009 η Sun αγοράστηκε από την Oracle. Χρησιμοποιείται κατά κύριο λόγο για desktop applications και για back-end προγραμματισμό [11].

2.12.1 Βασικά χαρακτηριστικά της Java

Τα βασικά χαρακτηριστικά της Java είναι:

- Ο κώδικας τρέχει σχεδόν σε οποιαδήποτε πλατφόρμα
- Σχεδιάστηκε για αντικειμενοστραφή προγραμματισμό
- Είναι Multithread γλώσσα που επιτρέπει αυτόματη διαχείριση μνήμης
- Διευκολύνει τον κατανεμημένο προγραμματισμό καθώς είναι network-centric

2.13 Σύγκριση Kotlin – Java

Ανάμεσα στις δύο γλώσσες, Kotlin και Java, εντοπίζονται πολλές διαφορές οι οποίες παρουσιάζονται αναλυτικά παρακάτω.

Checked Exceptions: Είναι μια βασική διαφορά ανάμεσα στις δύο γλώσσες. Πιο συγκεκριμένα η Kotlin δεν έχει κάποιο μηχανισμό που να υποστηρίζει checked exceptions. Αυτό έχει ως αποτέλεσμα στην Java ο προγραμματιστής να χρησιμοποιεί και να πιάνει τα exceptions που προκύπτουν που οδηγεί σε πιο σταθερό κώδικα με καλή διαχείριση λαθών. Ενώ στην Kotlin δεν υπάρχει αυτή η δυνατότητα.

Code Conciseness: Συγκρίνοντας δύο ισοδύναμες κλάσεις Kotlin και Java που έχουν την ίδια λειτουργικότητα μπορεί κανείς εύκολα να καταλάβει ότι στην Kotlin απαιτείται λιγότερος κώδικας.

Coroutines: Όποτε ερχόμαστε αντιμέτωποι χρονοβόρες λειτουργίες και σύνθετες λειτουργίες που απαιτούν πόρους το νήμα που τις εκτελεί μπλοκάρει, με αποτέλεσμα να μπλοκάρει και η εφαρμογή έως ότου ολοκληρωθεί η διεργασία. Το Android λειτουργεί ως single-thread με αποτέλεσμα να παγώνει το περιβάλλον εργασίας όσο είναι απασχολημένο το νήμα. Η λύση που προσφέρει η Java σε αυτό το ζήτημα είναι ο Multithread προγραμματισμός και πιο συγκεκριμένα η δημιουργία ενός background

νήματος. Αυτή η λύση όμως αυξάνει την πολυπλοκότητα και κατά συνέπεια και τα σφάλματα που προκύπτουν στον κώδικα. Αν και ο multithread προγραμματισμός είναι κάτι που υποστηρίζει και η Kotlin σε αυτή την περίπτωση προσφέρει την δυνατότητα της χρήσης Coroutines. Τα Coroutines δεν λειτουργούν με την λογική της στοίβας, αντίθετα αναστέλλουν την λειτουργία που πρέπει χωρίς να μπλοκάρουν κάποιο thread και συνεχίζεται η εκτέλεση της λειτουργίας αργότερα. Σε αντίθεση με τον multithread προγραμματισμό ο κώδικας που χρησιμοποιεί coroutines είναι σαφής και συνοπτικός.

Data classes: Οι κλάσεις δεδομένων προορίζονται αποκλειστικά για τη διατήρηση δεδομένων. Σε αυτές τις κλάσεις συναντάται ελάχιστη ή καθόλου λειτουργικότητα. Στην Java στις κλάσεις δεδομένων περιέχονται δομητές, μεταβλητές, setter και getter μέθοδοι. Αντίθετα στην Kotlin οι κλάσεις δεδομένων αρχικά δηλώνονται με την λέξη κλειδί data και έτσι ο compiler γνωρίζει ότι έχει να αντιμετωπίσει μια κλάση δεδομένων χωρίς να χρειαστεί να δηλωθούν δομητές και επιπλέον μέθοδοι.

Higher-Order Functions and Lambdas: Οι Higher-Order Functions είναι συναρτήσεις που δέχονται συναρτήσεις ως παραμέτρους ή επιστρέφουν συναρτήσεις. Επίσης οι συναρτήσεις στην Kotlin είναι πρώτης κλάσης, αυτό σημαίνει ότι μπορούν να αποθηκευτούν σε δομές δεδομένων και σε μεταβλητές που μπορούν να περάσουν σαν παράμετροι αλλά και να επιστραφούν από άλλες higher-order functions. Η Kotlin, ως στατική γλώσσα προγραμματισμού χρησιμοποιεί ένα εύρος τύπων συναρτήσεων για την αναπαράσταση άλλων συναρτήσεων. Επιπρόσθετα στην Kotlin συναντώνται και οι Lambda expressions. Οι Lambda και οι Anonymous συναρτήσεις που είναι γνωστά και ως function literals.

Implicit Widening Conversions: Από την Kotlin δεν υποστηρίζεται η άμεση μετατροπή αριθμών από μικρότερους σε μεγαλύτερους, σε αντίθεση με την Java. Στην Kotlin πρέπει να εκτελέσει μια άμεση και σαφή μετατροπή.

Inline Functions: Οι μεταβλητές που έχουν πρόσβαση στην συνάρτηση είναι γνωστές ως closures. Η χρήση High-end function μπορεί να επιβάλει αρκετές κυρώσεις στον χρόνο εκτέλεσης. Κάθε συνάρτηση στην Kotlin είναι ένα αντικείμενο και καταγράφει ένα closure. Τόσο οι κλάσεις όσο και οι συναρτήσεις αντικειμένων απαιτούν εκχώρηση μνήμης. Αυτές μαζί με εικονικές κλήσεις εισάγουν γενικά το χρόνο εκτέλεσης. Μια τέτοια επιπλέον επιβάρυνση μπορεί να αποφευχθεί με την κλίση των εκφράσεων Lambda στην Kotlin. Ένα τέτοιο παράδειγμα είναι η συνάρτηση lock(). Σε αντίθεση με την Kotlin, η Java δεν παρέχει υποστήριξη για ενσωματωμένες συναρτήσεις, αλλά μπορεί να χρησιμοποιήσει final μεθόδους.

Native Support for Delegation: Το Delegation είναι η διαδικασία που ένα αντικείμενο μεταβιβάζει την λειτουργία του σε ένα άλλο. Το class Delegation είναι μια εναλλακτική λύση για την κληρονομικότητα στην Kotlin. Αυτό καθιστά δυνατή τη χρήση πολλαπλών κληρονομικοτήτων.

Null Safety: Ένα από τα πιο ενοχλητικά ζητήματα που έχει να αντιμετωπίσει ένας προγραμματιστής είναι τα NullPointerExceptions. Στην Java οι προγραμματιστές μπορούν να εκχωρήσουν την τιμή null σε οποιαδήποτε μεταβλητή. Αν λοιπόν μετά προσπαθήσουν να κάνουν αναφορά σε ένα αντικείμενο που έχει μηδενική τιμή έρχονται αντιμέτωποι με το NullPointerException. Στην Kotlin όλοι οι τύποι είναι μηδενικοί από προεπιλογή και αν οι προγραμματιστές προσπαθήσουν να επιστρέψουν ή να εκχωρήσουν μηδενικά τότε θα αποτύχει στην στιγμή της μεταγλώττισης. Για να εκχωρηθεί η τιμή null σε μια μεταβλητή στην Kotlin πρέπει να δηλωθεί ως nullable, αυτό γίνεται αν προσθέσει ο προγραμματιστής ένα ερωτηματικό μετά την δήλωση του τύπου (πχ val number: Int? = null).

Primitive Types: Υπάρχουν 8 τύποι δεδομένων, συμπεριλαμβανομένων των char, double, float και int. Σε αντίθεση με την Kotlin, μεταβλητές ενός τύπου δεν είναι αντικείμενα σε Java. Αυτό σημαίνει ότι δεν αποτελούν αντικείμενο που δημιουργείται από μια κλάση ή μια δομή.

Smart Casts: Πριν ένα αντικείμενο μπορέσει να αλλάξει τύπο στην Java πρέπει να ελεγχθεί ο τύπος του. Στην Kotlin η αλλαγή τύπου μιας μεταβλητής γίνεται αυτόματα.

Static Members: Τα static μέλη η Java τα χρησιμοποιεί για να δηλώσει ότι αυτό το instance αυτή της μεταβλητής θα είναι διαθέσιμο όπου εμφανίζεται η κλάση που την περιέχει. Η Kotlin δεν χρησιμοποιεί static μέλη.

Support for Constructors: Η Kotlin σε αντίθεση με την Java μπορεί να έχει έναν ή παραπάνω δευτερεύοντες δομητές πέρα από τον ένα κύριο δομητή.

Ternary Operator: Η Java μπορεί να χρησιμοποιήσει τον τριαδικό τελεστή ο οποίος λειτουργεί όπως η if. Πιο συγκεκριμένα περιέχει μία συνθήκη και ελέγχει αν αυτή η συνθήκη είναι αληθής ή ψευδής και ανάλογα το αποτέλεσμα επιστρέφεται μία από τις δύο τιμές.

(condition) ? (value1) : (value 2)

Wildcard types: Οι τύποι wildcards είναι διαθέσιμοι μόνο στην Java. Η Kotlin για να καλύψει αυτό το κενό χρησιμοποιεί Declaration-site variance και type projections [12].

2.14 Αισθητήρες

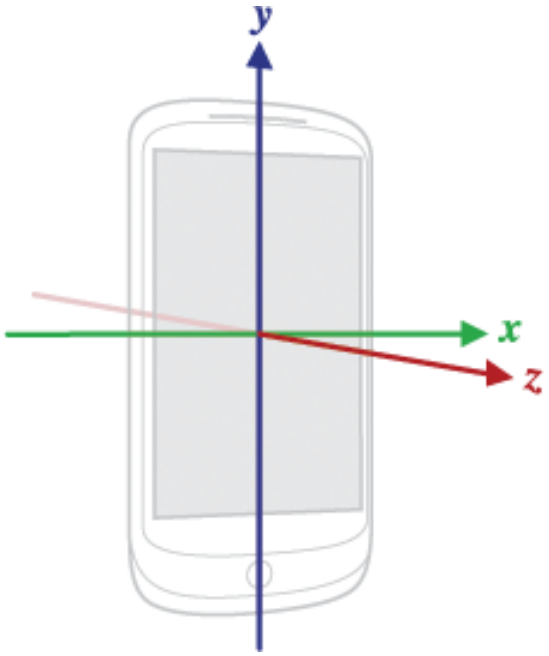
Ένα έξυπνο τηλέφωνο περιέχει αισθητήρες που προσδιορίζουν τις κινήσεις που κάνει η συσκευή, την θέση της στον χώρο, αλλά και τις αλλαγές που εντοπίζονται στο περιβάλλον της. Κάθε συσκευή Android έχει διαφορετικούς αισθητήρες, όμως οι περισσότερες συσκευές διαθέτουν επιταχυνσιόμετρο και μαγνητόμετρο.

Όταν μιλάμε για αισθητήρες το μυαλό μας πάει συνήθως σε Hardware-based αισθητήρες, δηλαδή σε αυτούς που εξαρτώνται από κάποιο συγκεκριμένο εξάρτημα που διαθέτει η συσκευή. Όμως οι αισθητήρες μπορεί να είναι και Software-based δηλαδή εικονικοί. Πιο αναλυτικά, τα αποτελέσματά τους να προκύπτουν από ένα συνδυασμό μετρήσεων από διάφορους Hardware-based αισθητήρες.

Οι αισθητήρες που μπορεί να έχει μια συσκευή Android χωρίζονται σε τρεις κατηγορίες. Τους αισθητήρες κίνησης, τους αισθητήρες θέσης και τους αισθητήρες περιβάλλοντος. Θα ασχοληθούμε περισσότερο με τους αισθητήρες κίνησης και θέσης καθώς αυτοί είναι που αφορούν την εφαρμογή [4].

2.14.1 Αισθητήρες κίνησης

Οι αισθητήρες αυτής της κατηγορίας μετράνε τις δυνάμεις της επιτάχυνσης και της περιστροφής που ασκούνται στην κινητή συσκευή στους άξονες x, y και z που φαίνονται και στην Εικόνα 5.



Εικόνα 5: Άξονες αισθητήρων κίνησης [13]

2.14.1.1 Επιταχυνσιόμετρο

Το επιταχυνσιόμετρο ανήκει στους Hardware-based αισθητήρες. Ο αισθητήρας αυτός είναι υπεύθυνος για να μετράει την δύναμη της επιτάχυνσης που ασκείται στην συσκευή στους τρεις άξονες x, y, z , της Εικόνας 3. Η χρήση του επιταχυνσιόμετρου γίνεται συνήθως σε παιχνίδια, για την αναγνώριση απότομων κινήσεων, όμως σε ορισμένες περιπτώσεις χρησιμοποιείται και για τον υπολογισμό της ταχύτητας του χρήστη [13].

2.14.1.2 Γυροσκόπιο

Το γυροσκόπιο, όπως και το επιταχυνσιόμετρο ανήκει στους Hardware-based αισθητήρες. Η δουλειά αυτού του αισθητήρα είναι η μέτρηση του ρυθμού περιστροφής της συσκευής γύρω από τον κάθε ένα από τους τρείς άξονες x, y, z (Εικόνα 3). Οι μετρήσεις αυτού του αισθητήρα αξιοποιούνται για τον εντοπισμό κινήσεων όπως μία στροφή ή μια περιστροφή της συσκευής [13].

2.14.1.3 Αισθητήρας βαρύτητας

Αυτός ο αισθητήρας στις περισσότερες συσκευές είναι Software-based καθώς η δύναμη της βαρύτητας που ασκείται στην συσκευή μπορεί να υπολογιστεί με την βοήθεια του επιταχυνσιόμετρου [13].

2.14.1.4 Αισθητήρας οριζόντιας επιτάχυνσης

Ο αισθητήρας οριζόντιας επιτάχυνσης υπολογίζει τη δύναμη της οριζόντιας επιτάχυνσης που ασκείται στους τρείς άξονες. Ανήκει και αυτός στους Software-based αισθητήρες και η τιμή του μπορεί να

υπολογιστεί χρησιμοποιώντας τις μετρήσεις από το επιταχυνσιόμετρο και από τον αισθητήρα βαρύτητας [13].

2.14.1.5 Αισθητήρας διανύσματος γωνίας περιστροφής

Είναι ο τρίτος αισθητήρας που συνήθως είναι Software-based, οι μετρήσεις του οποίου υπολογίζονται συνήθως από το επιταχυνσιόμετρο, το γυροσκόπιο και το μαγνητόμετρο, το οποίο ανήκει στους αισθητήρες θέσης και θα αναπτυχθεί παρακάτω. Ο συγκεκριμένος αισθητήρας στην ουσία επιστρέφει την γωνία περιστροφής της συσκευής γύρω από έναν από τους άξονες x, y, z [13].

2.14.1.6 Μετρητής βημάτων

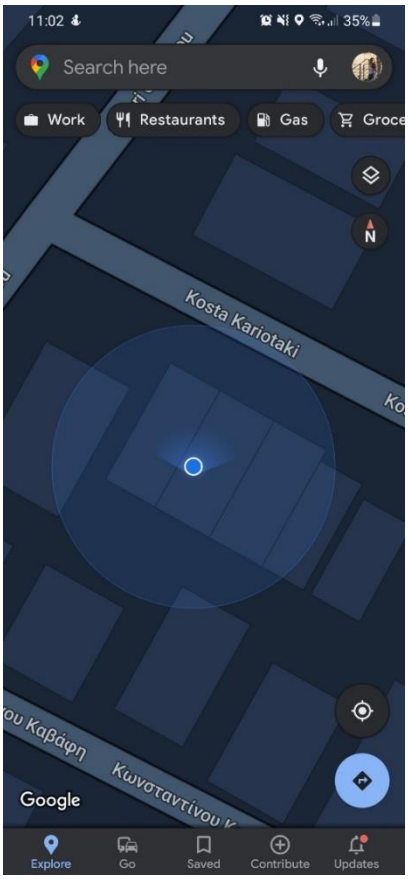
Ο μετρητής βημάτων είναι ένας hardware-based αισθητήρας που μετράει τα βήματα που κάνει ο χρήστης όταν κρατάει την συσκευή. Η τιμή αυτού του αισθητήρα δεν μηδενίζεται, δηλαδή μεταφέρεται από την μια εφαρμογή στην άλλη σε περίπτωση που χρησιμοποιείται από παραπάνω από μία εφαρμογές [13].

2.14.2 Αισθητήρες θέσης

Σε αυτή την κατηγορία αισθητήρων θα μπορούσε να βρίσκεται και ο αισθητήρας διανύσματος γωνίας περιστροφής καθώς τα αποτελέσματα που παράγει συμβάλουν στο να κατανοήσουμε την κατάσταση περιστροφής της συσκευής. Γενικά οι αισθητήρες θέσης έχουν ως στόχο να δώσουν την δυνατότητα σε μία εφαρμογή να αντιληφθεί τη θέση της συσκευής πάνω στους άξονες x, y, z [13].

2.14.2.1 Μαγνητόμετρο

Ο αισθητήρας αυτός μετράει το μαγνητικό πεδίο στους τρεις γνωστούς άξονες της Εικόνας 5. Ο αισθητήρας αυτός χρησιμοποιείται συνήθως για την εμφάνιση κατεύθυνσης του χρήστη όπως φαίνεται στην Εικόνα 6 [13].



Εικόνα 6: Εμφάνιση της κατεύθυνσης της συσκευής

2.15 Geolocation στο Android

Η ανάκτηση της τοποθεσίας της συσκευής στο Android γίνεται με δύο τρόπους. Ο πρώτος και λιγότερο ακριβής είναι με την χρήση της IP διεύθυνσης του WiFi δικτύου που ενδέχεται να είναι συνδεδεμένη η συσκευή, καθώς και μέσω των αναγνωριστικών των σταθμών των κινητών δικτύων (GSM/3G/4G cell-id). Για την μέγιστη δυνατή ακρίβεια στην ανάκτηση της θέσης του χρήστη χρησιμοποιείται ο δείκτης GPS που προσφέρει ακρίβεια έως και 10 μέτρα [4].

Για την ανάκτηση της τοποθεσίας της συσκευής απαιτείται η κατασκευή ενός αντικειμένου της κλάσης LocationManager καθώς και ένα αντικείμενο της κλάσης LocationListener με την χρήσης της εντολής new. Με το LocationListener υλοποιείται η διεπαφή LocationListener καθώς και οι τέσσερις παρακάτω μέθοδοι που βοηθούν στην διαχείριση της θέσης του χρήστη:

- onLocationChanged(): περιλαμβάνει ένα object της κλάσης Location που περιέχει την θέση του χρήστη.
- onStatusChanged(): περιλαμβάνει πληροφορίες σχετικά με την κατάσταση σύνδεσης του παρόχου ανάκτησης θέσης.
- onProviderEnabled(): καλείται όταν ο πάροχος ανάκτησης θέσης ενεργοποιείται από τον χρήστη
- onProviderDisabled(): καλείται όταν ο πάροχος ανάκτησης θέσης απενεργοποιείται από τον χρήστη

2.16 Χάρτες

Οι χάρτες Google είναι η πιο πετυχημένη εφαρμογή Android. Πρωτοεμφανίστηκε το 2005 και άλλαξε τα δεδομένα στον τρόπο που λειτουργούν οι διαδικτυακές εφαρμογές χαρτογράφησης στο διαδίκτυο. Οι χάρτες Google είναι διαθέσιμοι για τους προγραμματιστές μέσω της υπηρεσίας Google Maps API 2 (Application Programming Interface) [4].

Το Google Maps API είναι μια δωρεάν υπηρεσία της Google αλλά έχει κάποιους περιορισμούς. Αυτοί οι περιορισμοί αναφέρονται στον αριθμό των request που μπορεί να κάνει ο χρήστης μέσα σε ένα ορισμένο χρονικό διάστημα. Αυτό το όριο διαφέρει για κάθε υπηρεσία που προσφέρουν οι χάρτες και μπορεί να επεκταθεί με συνδρομές επί πληρωμή [15].

Η Google διαχώρισε τα Google Maps APIs σύμφωνα με τις παραπάνω υπηρεσίες, ενδεικτικά κάποιες υπηρεσίες αναφέρονται παρακάτω:

- Maps SDK for Android: Παρέχει γενικές πληροφορίες χαρτών
- Directions API: Παρέχει οδηγίες ανάμεσα σε τοποθεσίες, ανάλογα με το μέσο μεταφοράς
- Places API: Παρέχει λεπτομερείς πληροφορίες για ένα συγκεκριμένο μέρος
- Distance Matrix API: Υπολογίζει τις αποστάσεις και τους χρόνους ταξιδιού ανάμεσα σε δύο σημεία ανάλογα με το μέσο μεταφοράς.

2.17 UX/UI

Κατά την σχεδίαση της εφαρμογής κύριο μέλημά μου ήταν η δημιουργία ενός καλαίσθητου προϊόντος όπου δεν θα στόχευε σε κάποια συγκεκριμένη ομάδα ανθρώπων. Αυτό σημαίνει ότι έπρεπε να δημιουργήσω μια εφαρμογή για όλες τις ηλικίες και για ανθρώπους που δεν έχουν την ίδια εξοικείωση με την τεχνολογία.

2.17.1 UX

Ένα από τα σημαντικότερα κομμάτια στην δημιουργία mobile εφαρμογών είναι το user experience (UX). Το UX δεν επικεντρώνεται μόνο στην εμφάνιση του προϊόντος αλλά και στην εμπειρία χρήσης του.

Ο σχεδιασμός στο UX περιλαμβάνει διαδικασίες όπως τον σχεδιασμό, την προτυποποίηση, την έρευνα, την ευχρηστία, την προσβασιμότητα, την ευρωστία και την λειτουργικότητα.

Όταν ένα προϊόν δημιουργείται χωρίς να λαμβάνεται υπόψιν το user experience τότε συνήθως το αποτέλεσμα δεν είναι το επιθυμητό. Το προϊόν συνήθως είναι δύσχρηστο, μη παραγωγικό και δυσνόητο. Έτσι ο χρήστης θα αναζητήσει ένα άλλο προϊόν που τον βολεύει περισσότερο και τον διευκολύνει [16].

Λαμβάνοντας υπόψιν το UX ο προγραμματιστής εξασφαλίζει τα εξής:

1. Αυξάνει την παραγωγικότητα
2. Αυξάνει τις πωλήσεις και τα έσοδα
3. Μειώνει το κόστος υποστήριξης και εκπαίδευσης
4. Μειώνει το κόστος ανάπτυξης και χρόνου ανάπτυξης
5. Μειώνει το κόστος συντήρησης
6. Αυξάνει την ικανοποίηση των πελατών

2.18 UI

Πολλοί άνθρωποι δεν μπορούν να εντοπίσουν τις διαφορές ανάμεσα στο UX και στο UI, όμως τα όρια ανάμεσα σε αυτές τις δύο έννοιες είναι πιο διακριτά από όσο πιστεύεται. Το UI επικεντρώνεται περισσότερο στην εμφάνιση και στην αίσθηση που δημιουργεί η διεπαφή. Δίνεται δηλαδή έμφαση στα χρώματα, την γραμματοσειρά, το σχήμα των κουμπιών και γενικά το layout [16].

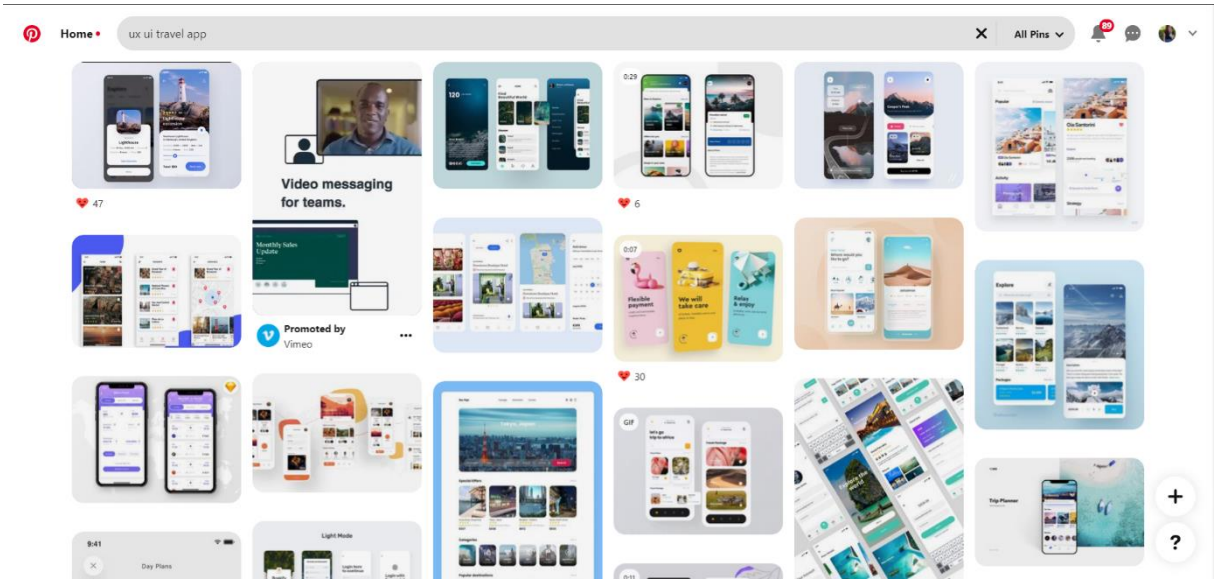
2.19 Διαδικασία σχεδίασης UX

Για τον σχεδιασμό UX πρέπει να ακολουθηθούν κάποια βήματα [16]:

1. Αναζήτηση και Πλάνο
2. Στρατηγική
3. Έρευνα
4. Ανάλυση
5. Σχέδιο
6. Παραγωγή

2.19.1 Αναζήτηση ιδεών

Στο πλαίσιο της διπλωματικής δεν έγινε επίσημη έρευνα για την απόσπαση πληροφοριών. Συγκεντρώθηκαν ανεπίσημα απόψεις και αντιδράσεις από το οικογενειακό και φιλικό περιβάλλον. Ενώ συνέβαλλε σημαντικά και η περιήγηση στο διαδίκτυο σε διάφορες σελίδες και εφαρμογές δημιουργικού περιεχομένου (πχ Pinterest Εικόνα 7) [17].



Εικόνα 7: Σελίδα δημιουργικού περιεχομένου Pinterest

2.19.2 Χρώματα

Το χρώμα είναι ένας εύκολος τρόπος για να περιγραφεί κάτι καθώς και για να ξεχωρίζουν τα αντικείμενα μεταξύ τους. Όμως πέρα από την διευκόλυνση που προσφέρουν στην καθημερινή μας επικοινωνία παίζουν σημαντικό ρόλο και στην ψυχολογία μας.

Η εφαρμογή που σχεδίασα αποτελείται από 3 χρώματα. Το λευκό φόντο, το οποίο επιλέχθηκε κυρίως επειδή στις εφαρμογές με μαύρο φόντο που συνηθίζονται τα τελευταία χρόνια (Dark Mode) είναι πολύ

δύσκολο να δει ο χρήστης αυτό που προβάλλεται στην οθόνη του όταν βρίσκεται κάτω από τον ήλιο. Η συγκεκριμένη εφαρμογή θα χρησιμοποιείται μόνο σε εξωτερικούς χώρους και το πιο σύνηθες θα είναι να χρησιμοποιηθεί κατά την διάρκεια της ημέρας, σε ευνοϊκές συνθήκες για περίπατο.

Τα άλλα δύο χρώματα που θα συναντήσει κάποιος στην εφαρμογή είναι δύο διαφορετικές αποχρώσεις του μπλε. Δηλαδή χρησιμοποιήθηκε το μονόχρωμο σύστημα χρωμάτων (Monochromatic color scheme), αυτό σημαίνει ότι χρησιμοποιήθηκαν δύο διαφορετικοί τόνοι από την ίδια πλευρά του χρωματικού κύκλου [18].

Η επιλογή του μπλε χρώματος έγινε γιατί αυτό το χρώμα σύμφωνα με την γενική ψυχολογία χρωμάτων συμβολίζει την ηρεμία και την υπευθυνότητα και πιο συγκεκριμένα το ανοιχτό μπλε αντιπροσωπεύει την οικειότητα και την ανανέωση. Επίσης είναι συνδεδεμένο με την ειρήνη και σε ορισμένους πολιτισμούς έχει πνευματικές και θρησκευτικές προεκτάσεις [19].

2.19.3 Σχήματα

Στις περισσότερες εφαρμογές τα πιο συνηθισμένα σχήματα που χρησιμοποιούνται είναι τα τετράγωνα και τα ορθογώνια παραλληλόγραμμα. Αυτά τα σχήματα δίνουν στον χρήστη την αίσθηση της αξιοπιστίας και της ασφάλειας. Όμως η χρήση αποκλειστικά τέτοιων σχημάτων μερικές φορές παραπέμπει σε έλλειψη φαντασίας και δημιουργικότητας για πολλούς χρήστες. Για αυτό τον λόγο χρησιμοποιήθηκαν κατά κύριο λόγο ορθογώνια παραλληλόγραμμα τόσο στα κουμπιά της εφαρμογής όσο και στα cells των list view, με την διαφορά ότι επέλεξα να έχουν στρογγυλοποιημένες τις γωνίες. Η στρογγυλοποίηση των γωνιών επιλέχθηκε γιατί δίνει μία πιο «παιχνιδιάρικη» διάθεση στο layout της εφαρμογής [20].

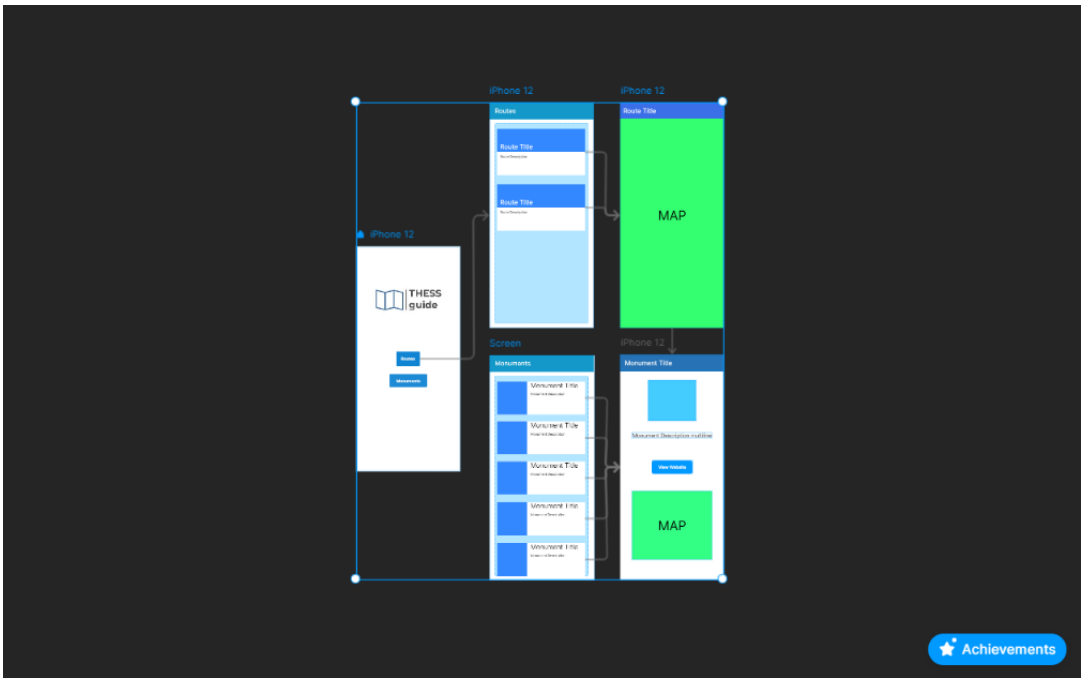
2.19.4 Σχεδιασμός διεπαφών

Πριν ξεκινήσει ο σχεδιασμός των διεπαφών πρέπει να δημιουργηθούν κάποια πρωτότυπα, που είναι γνωστά και ως wireframes. Αυτά τα πρωτότυπα χωρίζονται σε δύο κατηγορίες, την low-fidelity (lo-fi) και την high-fidelity (hi-fi). Οι διαφορές που εντοπίζονται ανάμεσα σε αυτές τις δύο κατηγορίες αφορούν τον τρόπο που σχεδιάζεται το πρωτότυπο [22].

Πιο συγκεκριμένα, το lo-fi wireframe είναι μια πιο «βασική» έκδοση του σχεδιασμού πρωτοτύπου. Το lo-fi wireframe μπορεί να σχεδιαστεί στο χέρι, σε ένα φύλλο χαρτί ή και πιο πρόχειρα σε ένα πρόγραμμα σχεδιασμού UI (Εικόνα 8), είναι μια γρήγορη λύση και είναι αρκετή εφόσον περιέχει τα στοιχεία που απαιτούνται για τον σχεδιασμό της διεπαφής και μπορεί να έχει η ομάδα ένα σημείο αναφοράς.

Αντίθετα το hi-fi wireframe θέλει περισσότερο χρόνο για να δημιουργηθεί, είναι πιο λεπτομερές και αναπαριστά την τελική εμφάνιση που αναμένεται να έχει η διεπαφή, με μεγάλη λεπτομέρεια [22].

Για την συγκεκριμένη εργασία χρησιμοποιήθηκε η πρώτη κατηγορία σχεδιασμού πρωτοτύπου (Εικόνα 8), δηλαδή το lo-fi και χρησιμοποιήθηκε το εργαλείο prototyping Framer[21]. Η επιλογή αυτής της προτυποποίησης έγινε επειδή δεν υπήρχε η ανάγκη για τον σχεδιασμό μίας τόσο λεπτομερούς απεικόνισης των διεπαφών εφόσον δεν δημιουργήθηκε κάποια ομάδα ανθρώπων που θα έπρεπε να εκτελέσουν ο καθένας διαφορετικά κομμάτια αυτής της εφαρμογής.



Εικόνα 8: Low-fidelity wireframe

2.20 Επίλογος

Ολοκληρώνοντας αυτό το κεφάλαιο φαίνεται πως για την δημιουργία των εφαρμογών Android η Google έχει μεριμνήσει για τις περισσότερες απαιτήσεις που μπορεί να έχει μία εφαρμογή. Η αρχιτεκτονική αυτών των εφαρμογών έχει χτιστεί έτσι ώστε να είναι όσο πιο αποτελεσματικές, οικονομικές γίνεται όσον αφορά την κατανάλωση πόρων. Αξίζει να σημειωθεί πως έχει δημιουργηθεί μια γλώσσα προγραμματισμού, η Kotlin, η οποία ιδανική για τις ανάγκες ανάπτυξης μια εφαρμογής Android, ενώ παράλληλα η βάση δεδομένων Firebase είναι πολύ εύχρηστη, ενώ από τα μεγαλύτερα πλεονεκτήματα της είναι ότι είναι online database, παράλληλα για κάποιον που έχει κάποια σχέση με τις βάσεις δεδομένων είναι πολύ εύκολο να την χρησιμοποιήσει και να καλύψει τις ανάγκες της εφαρμογής του. Τέλος πολύ σημαντικό ρόλο για την δημιουργία μίας οποιασδήποτε εφαρμογής παίζει και η σχεδίαση του περιβάλλοντος της δηλαδή το κομμάτι του UX/UI, αυτό είναι που θα δημιουργήσει τις πρώτες εντυπώσεις στον χρήστη ενώ μπορεί να καθορίσει ακόμη και τον χρόνο που θα χρειαστεί ο χρήστης ώστε να μάθει να χρησιμοποιεί μια εφαρμογή, αλλά και τον χρόνο που θα του πάρει για να φέρει σε πέρας μια εργασία.

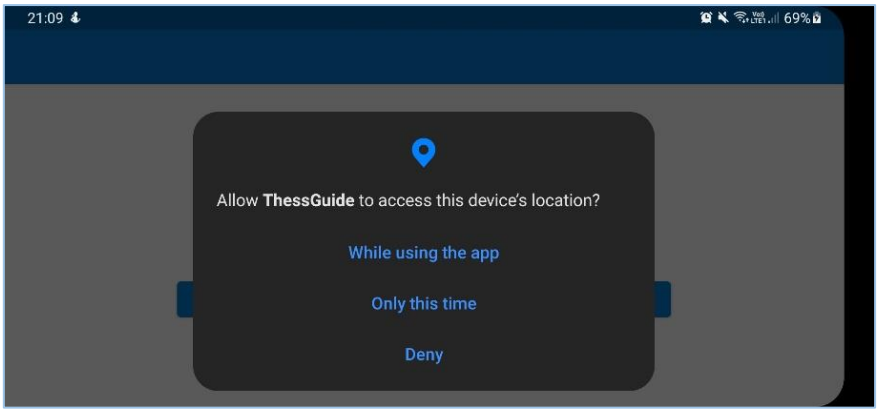
Κεφάλαιο 3ο: Λειτουργίες της Εφαρμογής

Εισαγωγή

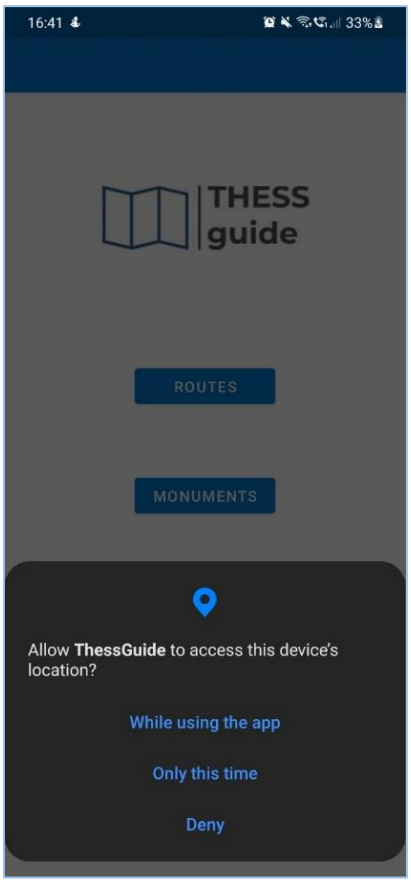
Σε αυτό το κεφάλαιο θα παρουσιαστούν σύντομα και περιληπτικά η διεπαφή και οι λειτουργίες της εφαρμογής σε portrait και landscape.

3.1 Η πρώτη εικόνα

Ανοίγοντας για πρώτη φορά την εφαρμογή θα ζητηθεί από τον χρήστη να επιτρέψει στην εφαρμογή την πρόσβαση στην τοποθεσία του (Εικόνα 9 και Εικόνα 10). Αυτό συμβαίνει γιατί η εφαρμογή δεν μπορεί να είναι λειτουργική χωρίς την ακριβή τοποθεσία του χρήστη. Για να μπορέσει να συνεχιστεί η χρήση της εφαρμογής η προτεινόμενη επιλογή είναι “While using the app”.



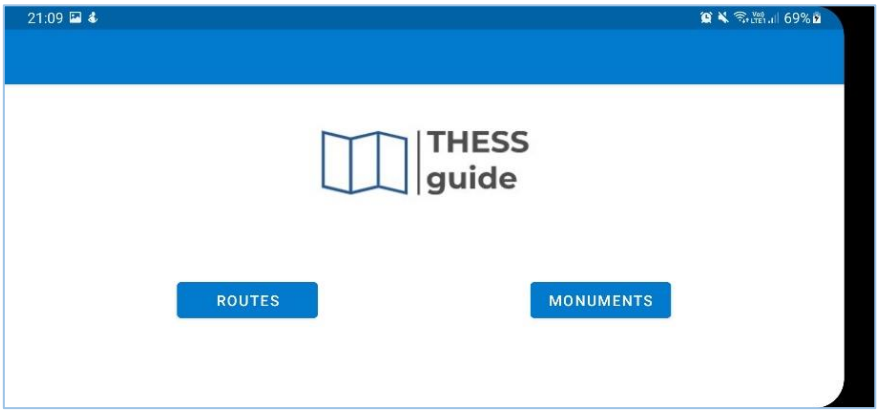
Εικόνα 9: Αίτηση άδειας για πρόσβαση στην τοποθεσία (landscape)



Εικόνα 10: Αίτηση άδειας για
πρόσβαση στην τοποθεσία (portrait)

3.2 Αρχική Οθόνη

Η Αρχική οθόνη αποτελείται από δύο κουμπιά που παρέχουν στον χρήστη την δυνατότητα να αποκτήσει πρόσβαση στις βασικές λειτουργίες της εφαρμογής. (Εικόνα 11 και Εικόνα 12)



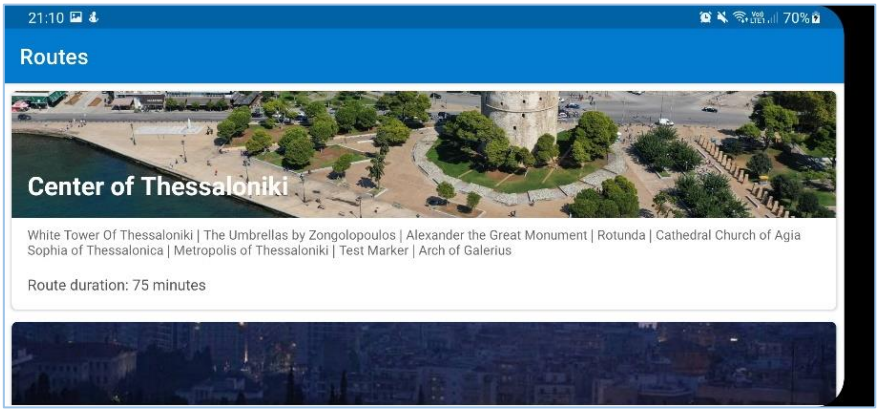
Εικόνα 11: Αρχική Οθόνη (landscape)

Εικόνα 12: Αρχική Οθόνη (portrait)

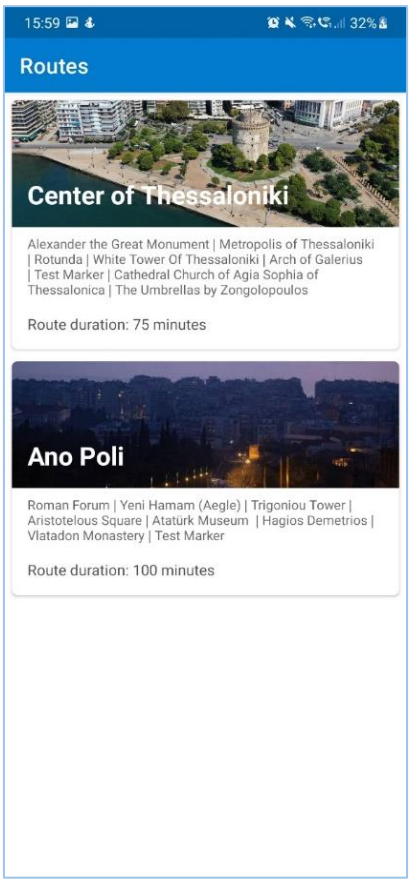
3.3 Διαδρομές

Ο χρήστης επιλέγοντας το κουμπί “Routes” της Εικόνας 11 ή 12 μεταφέρεται στην οθόνη της Εικόνας 13 και 14. Στην οθόνη αυτή μπορεί να επιλέξει ανάμεσα σε ένα πλήθος τουριστικών διαδρομών ποια τον ενδιαφέρει να ακολουθήσει.

Για κάθε διαδρομή αναφέρονται τα μνημεία που θα μπορεί να δει, ο χρόνος που θα χρειαστεί για να ολοκληρώσει την διαδρομή, ένας ενδεικτικός τίτλος και μία φωτογραφία για την κάθε διαδρομή.



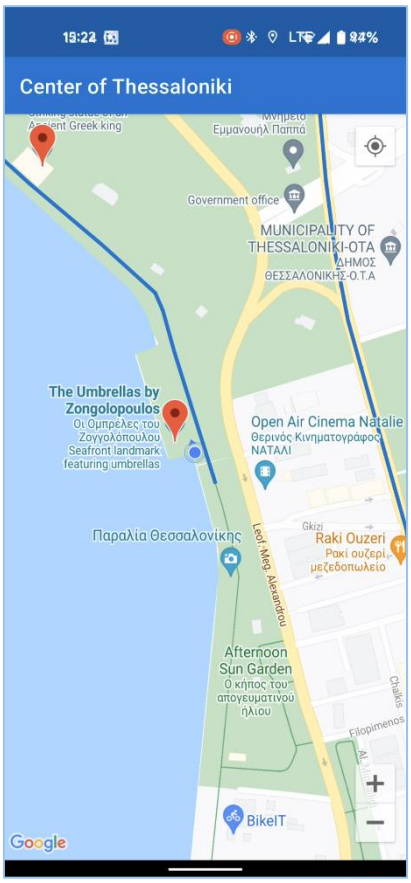
Εικόνα 13: List View Διαδρομών (landscape)



Εικόνα 14: List View Διαδρομών (portrait)

3.4 Χάρτης

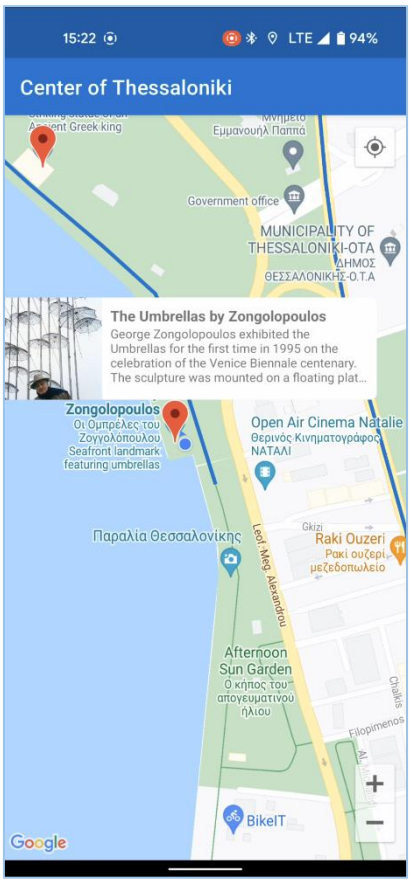
Με την επιλογή μιας διαδρομής από την προηγούμενη οθόνη (Εικόνα 13 ή 14) ο χρήστης μεταφέρεται στην παρακάτω διεπαφή. Εκεί βλέπει ένα χάρτη (Εικόνα 15), σημειωμένο με κάποια markers (πινέζες). Αυτές οι "πινέζες" είναι τοποθετημένες σε κάποια σημεία που έχουν τουριστικό ενδιαφέρον (μνημεία, έργα τέχνης, μουσεία, πλατείες, εκκλησίες κ.α.). Οι πινέζες είναι ενωμένες μεταξύ τους, ορίζοντας έτσι την διαδρομή που θα ακολουθήσει ο χρήστης ξεκινώντας από την τρέχουσα τοποθεσία του.



Εικόνα 15: Χάρτης με την διαδρομή

3.5 Επαύξηση Περιεχομένου

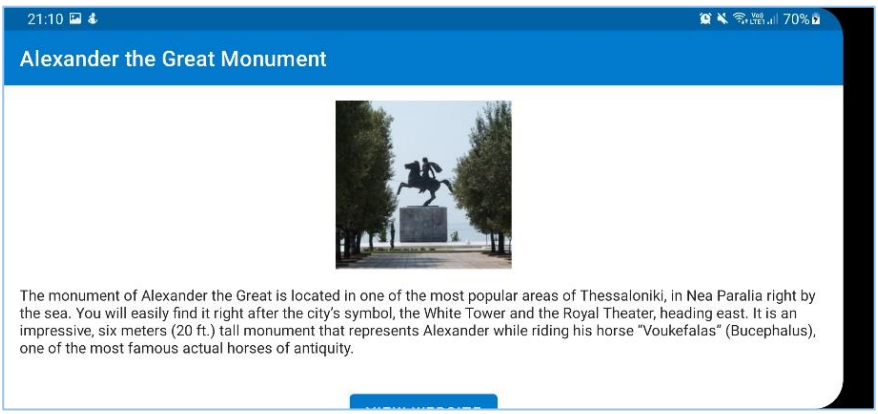
Όταν ο χρήστης πλησιάζει σε απόσταση 20 μέτρων μια από τις πινέζες, γίνεται επαύξηση του χάρτη καθώς αναδύεται το παράθυρο πληροφοριών της πινέζας (Εικόνα 16). Σε αυτό το παράθυρο ο χρήστης μπορεί να δει το όνομα του “σημείου ενδιαφέροντος”, μια φωτογραφία του και ένα απόσπασμα με κάποιες πληροφορίες.

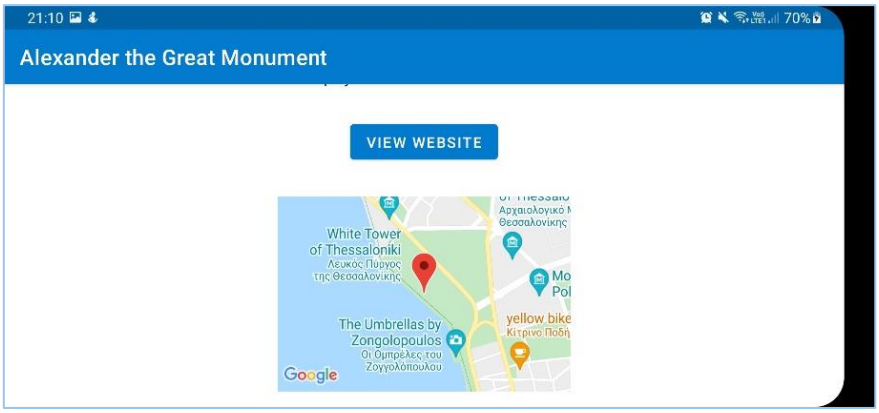


Εικόνα 16: Επαύξηση περιεχομένου του χάρτη

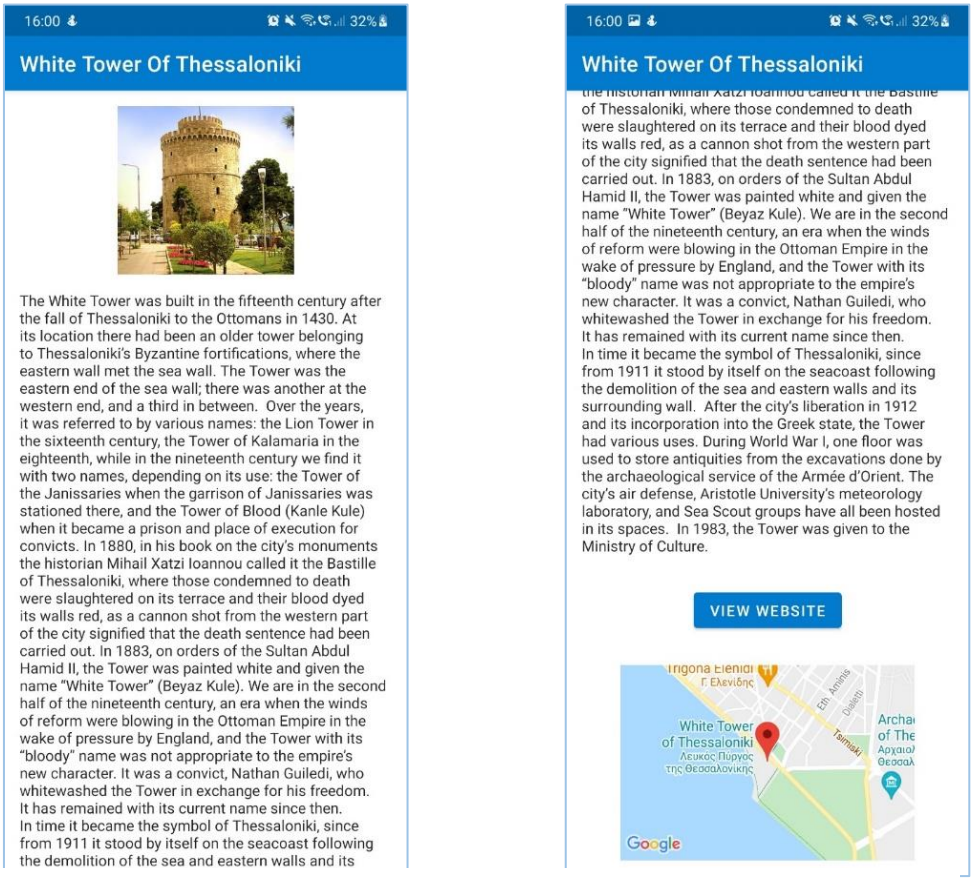
3.6 Σελίδα σημείου ενδιαφέροντος

Πατώντας πάνω στο αναδυόμενο παράθυρο πληροφοριών ο χρήστης, μεταφέρεται στην σελίδα λεπτομερειών (Εικόνα 17 και Εικόνα 18). Εκεί μπορεί να δει τις υπόλοιπες πληροφορίες για το συγκεκριμένο μνημείο, την θέση του μνημείου στον χάρτη και ένα κουμπί που τον μεταφέρει στην επίσημη σελίδα του Δήμου Θεσσαλονίκης για το μνημείο, εφόσον αυτή υπάρχει.





Εικόνα 17: Λεπτομέρειες για το σημείο ενδιαφέροντος (landscape)

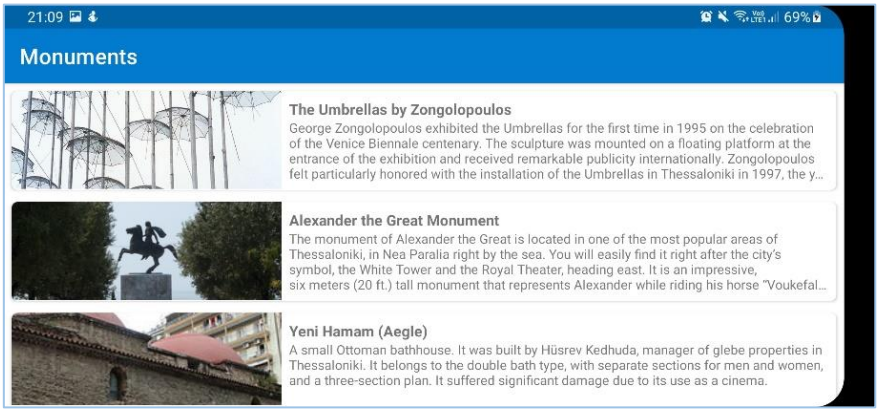


Εικόνα 18: Λεπτομέρειες για το σημείο ενδιαφέροντος (portrait)

3.7 Σημεία Ενδιαφέροντος (Μνημεία)

Ο χρήστης επιλέγοντας το κουμπί "Monuments" από την Αρχική οθόνη (Εικόνα 12) μεταφέρεται στην οθόνη της Εικόνας 19 ή Εικόνας 20 ανάλογα με την περιστροφή της συσκευής. Σε αυτή την διεπαφή

βλέπει μια λίστα από τα διάφορα σημεία ενδιαφέροντος που συναντώνται και στις “Διαδρομές”. Για κάθε σημείο ενδιαφέροντος ο χρήστης βλέπει το όνομα, μια φωτογραφία και ένα απόσπασμα από την περιγραφή. Μόλις ο χρήστης επιλέξει ένα από τα σημεία ενδιαφέροντος μεταφέρεται στην “Σελίδα σημείου ενδιαφέροντος” (Εικόνα 17).



Εικόνα 20: List View σημείων ενδιαφέροντος (landscape)

Εικόνα 19: List View σημείων ενδιαφέροντος (portrait)

3.8 Επίλογος

Παρουσιάστηκαν όλες οι επιμέρους διεπαφές χρήστη και οι λειτουργίες τους, με σκοπό την κατανόηση της λειτουργίας της εφαρμογής και την ευκολότερη προσαρμογή στο επόμενο κεφάλαιο που αναλύεται ο κώδικας που υπάρχει πίσω από αυτές τις διεπαφές που μόλις παρουσιάστηκαν.

Κεφάλαιο 4ο: Ανάλυση της Εφαρμογής

4.1 Εισαγωγή

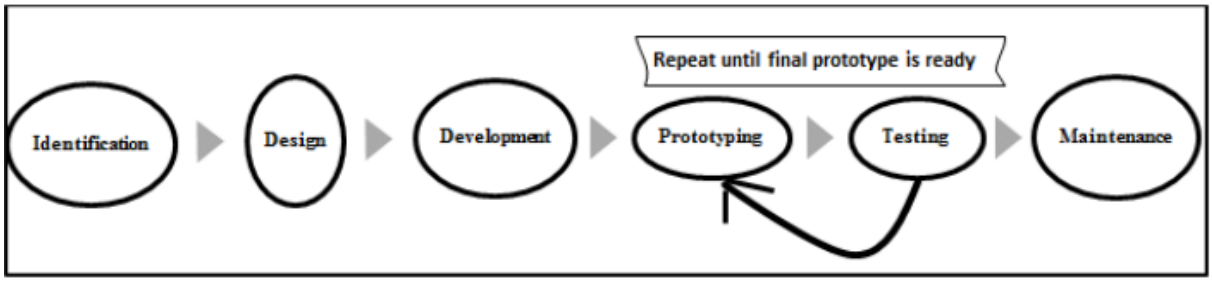
Στο τέταρτο κεφάλαιο της εργασίας θα αναλυθούν όλα στα στοιχεία της εφαρμογής και κυρίως ο κώδικας που βρίσκεται από πίσω.

4.2 Κύκλος ζωής ανάπτυξης εφαρμογών για κινητά (MADLC)

Το 2014 προτάθηκε ένας νέος κύκλος ζωής ανάπτυξης εφαρμογών για κινητά από τους Tejas Vithani και Anand Kumar το MADLC. Σε αυτόν τον κύκλο ζωής περιλαμβάνονται τα εξής στάδια: Ταυτοποίηση, Σχεδιασμός, Ανάπτυξη, Προτυποποίηση, Δοκιμή και Συντήρηση [23].

1. Φάση Ταυτοποίησης: Στο πρώτο στάδιο γίνεται συλλογή ιδεών και η κατηγοριοποίησή τους. Αυτές οι ιδέες μπορεί να προέρχονται από τον πελάτη, από τους προγραμματιστές ή από την συνεργασία αυτών των δύο.
2. Φάση Σχεδίασης: Στην φάση αυτή η ιδέα της ομάδας του mobile development παίρνει κάποια αρχική μορφή, σχεδιάζεται το UI της εφαρμογής (wireframing).
3. Φάση Ανάπτυξης: Στο στάδιο αυτό ξεκινάει η κωδικοποίηση της εφαρμογής. Η κωδικοποίηση μπορεί να προχωράει παράλληλα για διάφορα Modules της εφαρμογής.
4. Φάση Προτυποποίησης: Κατά την προτυποποίηση αναλύονται οι λειτουργικές απαιτήσεις κάθε προτύπου. Τα πρότυπα ελέγχονται και στέλνονται στον πελάτη για ανατροφοδότηση. Μετά την ανατροφοδότηση οι απαιτούμενες αλλαγές περνάνε πάλι από την φάση της Ανάπτυξης. Όταν το δεύτερο πρότυπο είναι έτοιμο ενσωματώνεται στο πρώτο, ελέγχεται και στέλνεται στον πελάτη για ανατροφοδότηση. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να καταλήξουμε στο τελικό πρότυπο.
5. Φάση Ελέγχου: Ο έλεγχος των προτύπων γίνεται σε simulator και στην συνέχεια σε φυσική συσκευή.
6. Φάση Συντήρησης: Στο τελικό στάδιο συλλέγονται σχόλια και παρατηρήσεις των χρηστών της εφαρμογής και γίνονται οι κατάλληλες διορθώσεις και βελτιώσεις.

Παρακάτω παρατίθεται η εικόνα 21 που βοηθάει στην ευκολότερη κατανόηση του κύκλου ζωής MADLC.



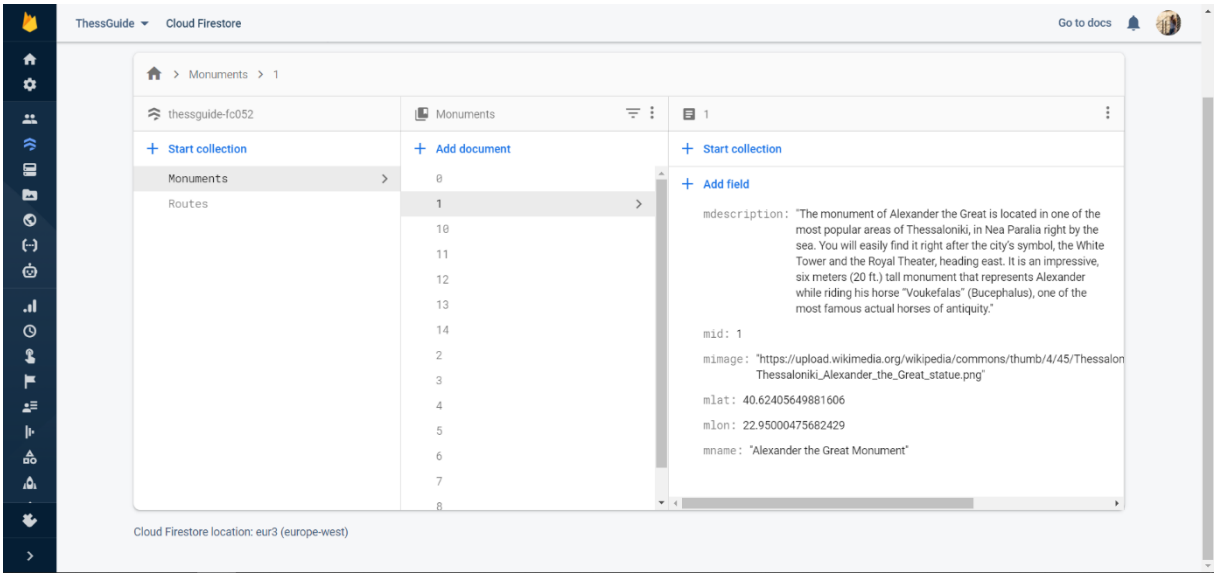
Εικόνα 21: MADLC [24]

4.3 Συλλογές Βάσης Δεδομένων

Το αμέσως επόμενο βήμα από την σχεδίαση των wireframes είναι η σχεδίαση και δημιουργία της βάσης δεδομένων, την οργάνωση των collection, των document και των field που περιέχονται σε αυτά. Τα collection που δημιουργήθηκαν για τις ανάγκες της εφαρμογής είναι το Monuments και το Routes.

4.3.1 Monuments

Στην συλλογή Monuments περιέχονται δεκαπέντε document που αναπαριστούν τα δεκαπέντε σημεία ενδιαφέροντος της πόλης. Κάθε document περιγράφεται από κάποια πεδία. Αυτά τα πεδία που περιγράφουν ένα document της συλλογής Monuments φαίνονται στην εικόνα 22 και αναλύονται παρακάτω είναι τα εξής:



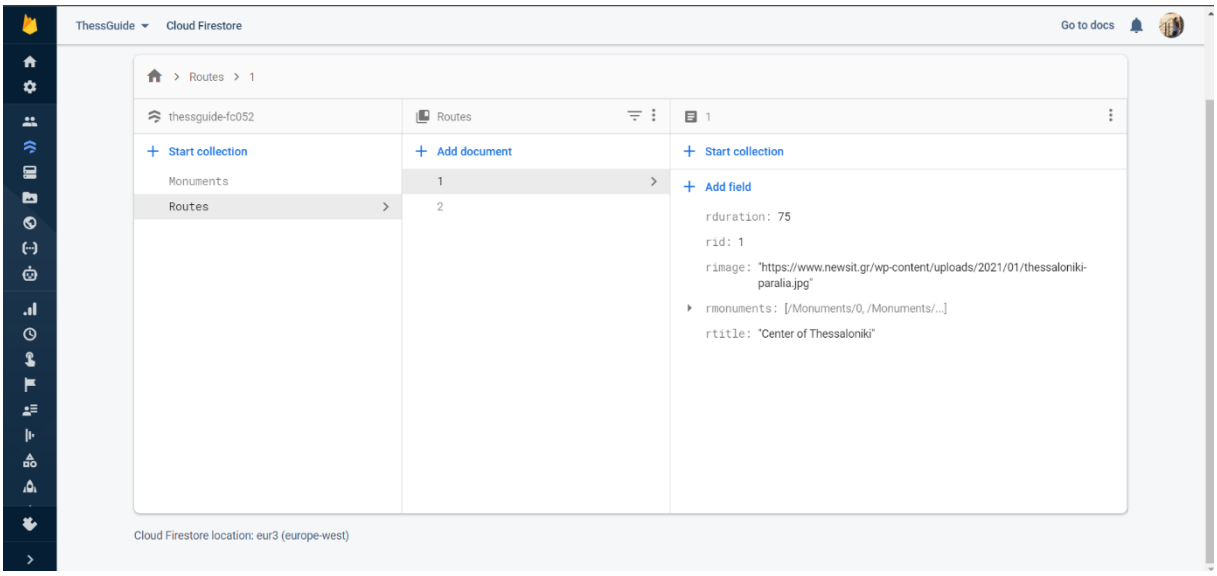
Εικόνα 21: Collection Monuments στο Firebase

- mdescription: Είναι τύπου String και περιέχει μια περιγραφή για το συγκεκριμένο σημείο ενδιαφέροντος.
- mid: Είναι τύπου Number και αναπαριστά το μοναδικό id του σημείου ενδιαφέροντος.
- mimage: Είναι τύπου String και περιέχει ένα url που οδηγεί σε μια εικόνα του σημείου ενδιαφέροντος.
- mlat: Είναι τύπου Number και αναπαριστά το γεωγραφικό μήκος που βρίσκεται το σημείο ενδιαφέροντος.
- mlon: Είναι τύπου Number και αναπαριστά το γεωγραφικό πλάτος που βρίσκεται το σημείο ενδιαφέροντος.
- mname: Είναι τύπου String και περιέχει τον τίτλο/όνομα που έχει αποδοθεί στο σημείο ενδιαφέροντος.
- mwebsite: Είναι τύπου String και περιέχει ένα url που οδηγεί στην ιστοσελίδα του σημείου ενδιαφέροντος εφόσον αυτή υπάρχει.

4.3.2 Routes

Η δεύτερη συλλογή που συναντάται στην βάση δεδομένων της εφαρμογής είναι η Routes. Στην συλλογή αυτή περιέχονται δύο documents, δηλαδή δύο διαδρομές οι οποίες αποτελούνται από τα fields που φαίνονται στην εικόνα 23 και αναλύονται παρακάτω:

- rduration: Είναι τύπου Number και αντιπροσωπεύει τον χρόνο που θα χρειαστεί ο χρήστης για να ολοκληρώσει την συγκεκριμένη διαδρομή.
- rid: Είναι τύπου Number και αναπαριστά το μοναδικό id της διαδρομής
- rimage: Είναι τύπου String και περιέχει ένα url που οδηγεί σε μια εικόνα της διαδρομής.
- rtitle: Είναι τύπου String και περιέχει τον τίτλο/όνομα που έχει αποδοθεί στην διαδρομή
- rmonuments: Είναι τύπου Array η οποία αποτελείται από εφτά κελία τύπου reference και πιο συγκεκριμένα τύπου Monument.



Εικόνα 22: Collection Routes στο Firebase

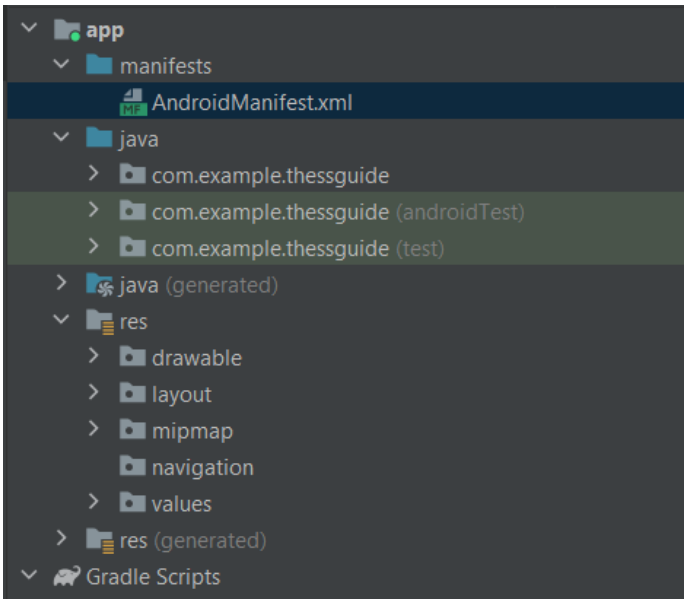
4.4 Δομή του project στο Android Studio

Στην εικόνα 24 φαίνεται η δομή μίας εφαρμογής στο Android Studio. Ξεκινώντας από πάνω προς τα κάτω συναντάμε πρώτα το αρχείο AndroidManifest.xml το αρχείο αυτό περιγράφει τις απαραίτητες πληροφορίες για την εφαρμογή στα Android build tools, στο λειτουργικό σύστημα Android και στο Google Play. Σε ένα Android project μπορεί να συναντήσουμε πάνω από ένα αρχείο manifest αν υπάρχουν πάνω από μία παραλλαγές για την ίδια εφαρμογή. [25]

Κάτω από τον φάκελο manifest συναντάται ο φάκελος java. Ο πρώτος υποφάκελος περιέχει τον πηγαίο κώδικα σε Kotlin ο οποίος θα αναλυθεί λεπτομερώς παρακάτω. Οι άλλοι δύο φάκελοι που βρίσκονται μέσα στον φάκελο java, οι androidTest και Test, περιέχουν java αρχεία που χρησιμοποιούνται για τον έλεγχο της σωστής λειτουργίας της εκάστοτε εφαρμογής. [26]

Στον φάκελο res (resources) βρίσκονται αρχεία που χρησιμοποιούνται στην σχεδίαση των διεπαφών. Τα αρχεία αυτά κατηγοριοποιούνται ανά είδος στους υποφακέλους που βρίσκονται μέσα στον φάκελο res. Για την συγκεκριμένη εφαρμογή οι κατηγορίες αρχείων που συναντάμε είναι drawable, layout, mipmap, navigation και values. Στον φάκελο drawable υπάρχουν όλες οι εικόνες και τα σχήματα που συναντώνται μέσα στην εφαρμογή. Στον φάκελο layout περιλαμβάνονται όλες οι διεπαφές που έχουν σχεδιαστεί για την εφαρμογή, δηλαδή τα activities, τα fragments ή κάποια άλλα views. Στον φάκελο values περιλαμβάνονται τα αρχεία colors.xml, dimens.xml, strings.xml κ.α. οπού στο αρχείο colors περιλαμβάνουν τα χρώματα που συναντώνται στην εφαρμογή, στο αρχείο dimens τις διαστάσεις που μπορεί να έχουν κάποια widgets που χρησιμοποιούνται στην εφαρμογή, στον φάκελο midmap

περιέχεται ένα launcher.xml που περιέχει ένα εύρος μεγεθών του ίδιου εικονιδίου που η χρήση τους εξαρτάται από το μέγεθος της συσκευής που τα χρησιμοποιεί και τέλος στο αρχείο strings που είναι ίσως και το πιο «σημαντικό» από τα υπόλοιπα και περιέχει όλα τα resources της εφαρμογής [27].



Εικόνα 23: Δομή Android project

4.5 AndroidManifest.xml

Το AndroidManifest.xml είναι απαραίτητο root αρχείο για κάθε Android project. Σε αυτό το αρχείο περιλαμβάνονται οι απολύτως απαραίτητες πληροφορίες που χρειάζονται για την δημιουργία του πρότζεκτ. Ουσιαστικά ενημερώνει το σύστημα για τα στοιχεία της εφαρμογής. Εδώ δηλώνονται το όνομα του πακέτου της εφαρμογής, τα components από τα οποία αποτελείται, δηλαδή τα activities, services, receivers και providers, τα δικαιώματα που απαιτούνται για την λειτουργία της καθώς και τα απαραίτητα χαρακτηριστικά υλικού και λογισμικού που χρειάζεται η εφαρμογή για την εύρυθμη λειτουργία της (Εικόνα 25)[25].

Όταν γίνεται build μια εφαρμογή το package name αντικαθίσταται από το application ID το οποίο ορίζεται στο grandle και χρησιμοποιείται ως μοναδικός προσδιοριστής της εφαρμογής κατά την μεταφορά της στο Google Play. Όταν μια εφαρμογή μεταφέρεται στο Google Play παίρνει και άλλες πληροφορίες από το Android Manifest και συγκεκριμένα πληροφορίες που αναφέρουν ποιες συσκευές είναι ικανές να υποστηρίξουν τις απαιτήσεις αυτής της εφαρμογής τόσο από άποψη λειτουργικού συστήματος όσο και από τις απαιτήσεις του υλικού[25].

Στο AndroidManifest δηλώνονται και τα δικαιώματα που χρειάζεται η εφαρμογή για να λειτουργήσει. Όπως φαίνεται και στις γραμμές πέντε με οκτώ του κώδικα (εικόνα 25), καθώς και τα διάφορα components που συνιστούν την εφαρμογή. Δηλαδή τα activities, services, meta-data και libraries που φαίνονται από τις γραμμές δέκα και μετά του AndroidManifest (εικόνα 25 και εικόνα 26)[25].

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.thessguide">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />

    <application
        android:allowBackup="true"
        android:fullBackupContent="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ThessGuide">
        <activity
            android:name=".adapters.CustomInfoWindowAdapter"
            android:label="@string/title_activity_custom_info_window_adapter"
            android:theme="@style/Theme.ThessGuide.NoActionBar" />
        <activity android:name=".ui.MonumentDetailsActivity" />
        <activity android:name=".ui.RoutesActivity" />
        <activity android:name=".ui.MonumentsActivity" />
```

Εικόνα 24: Android Manifest(1)

AndroidManifest.xml

```xml
            android:label="@string/title_activity_custom_info_window_adapter"
            android:theme="@style/Theme.ThessGuide.NoActionBar" />
        <activity android:name=".ui.MonumentDetailsActivity" />
        <activity android:name=".ui.RoutesActivity" />
        <activity android:name=".ui.MonumentsActivity" />
        <!--
             The API key for Google Maps-based APIs is defined as a string resource.
             (See the file "res/values/google_maps_api.xml").
             Note that the API key is linked to the encryption key used to sign the APK.
             You need a different API key for each encryption key, including the release key that is used to
             sign the APK for publishing.
             You can define the keys for the debug and release targets in src/debug/ and src/release/.
        -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".ui.MapsActivity"
            android:label="@string/title_activity_maps" />
        <activity android:name=".ui.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Εικόνα 25: Android Manifest(2)

4.6 Δικαιώματα (Permissions)

Τα δικαιώματα (permissions) μιας εφαρμογής είναι απαραίτητα για να έχει πρόσβαση σε διάφορα προστατευμένα μέρη του συστήματος ή άλλων εφαρμογών. Επίσης δηλώνονται και permissions που είναι απαραίτητα να έχουν οι άλλες εφαρμογές για να έχουν πρόσβαση σε αυτή την εφαρμογή.

Η δήλωση ενός δικαιώματος γίνεται μέσα στο Manifest και έχει την μορφή «uses-permission android:name=”PERMISSION_NAME”/>». Για την συγκεκριμένη εφαρμογή τα permissions που δηλώνονται είναι το INTERNET, το ACCESS_FINE_LOCATION, το ACCESS_COARSE_LOCATION και το ACCESS_BACKGROUND_LOCATION (Εικόνα 27)[26].

```
5       <uses-permission android:name="android.permission.INTERNET" />
6       <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
7       <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
8       <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
```

Εικόνα 26: Δικαιώματα

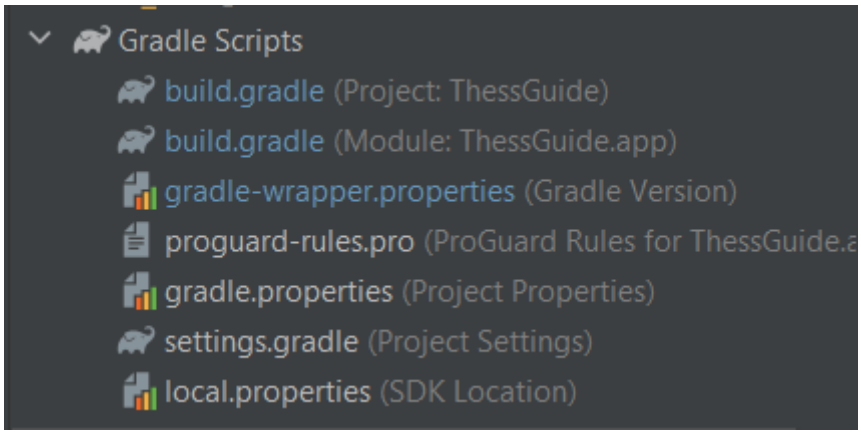
Τα δικαιώματα ACCESS_FINE_LOCATION και ACCESS_COARSE_LOCATION πρέπει να παραχωρηθούν για να μπορεί να ανακτηθεί η τρέχουσα τοποθεσία της συσκευής. Πιο συγκεκριμένα το ACCESS_FINE_LOCATION επιτρέπει στο API να προσδιορίσει την τοποθεσία μέσω των διαθέσιμων παρόχων τοποθεσίας όπως το Παγκόσμιο Σύστημα Εντοπισμού Θέσης (GPS) ενώ το ACCESS_COARSE_LOCATION χρησιμοποιεί τα δεδομένα κινητής τηλεφωνίας ή το WiFi.

Το δικαίωμα INTERNET είναι αυτό που επιτρέπει στην εφαρμογή την πρόσβαση στο διαδίκτυο [26]. Χωρίς αυτό το δικαίωμα η συγκεκριμένη εφαρμογή δεν μπορεί να λειτουργήσει. Καθώς δεν θα έχει πρόσβαση στα βασικά δεδομένα της εφαρμογής.

4.7 Αρχεία Gradle

Τα αρχεία Gradle είναι αρκετά όπως φαίνονται και στην εικόνα 28 και συναντώνται στον κατάλογο του project από την στιγμή της δημιουργίας του. Το Gradle είναι ένα προηγμένο σύνολο εργαλείων build που χρησιμοποιούνται για την αυτοματοποίηση και την διαχείριση της διαδικασίας build. Σε ένα πρότζεκτ μπορεί να υπάρχουν παραπάνω από ένα build και το καθένα να παρέχει διαφορετικές λειτουργίες. Οι παραλλαγές αυτές ονομάζονται flavors και εξυπηρετούν τις διάφορες εκδόσεις της ίδιας εφαρμογής. Για παράδειγμα η δωρεάν, περιορισμένη έκδοση και η επί πληρωμή, πλήρης έκδοση [28].

Όπως φαίνεται και στην εικόνα 28 για όλα τα project έχουν δύο build.gradle αρχεία το Top Level και το Module Level(:app).



Εικόνα 27: Grandle Αρχεία

4.7.1 Top Level Build Αρχείο build.gradle (ThessGuide)

Στο Top Level (Project) καθορίζονται οι ρυθμίσεις του build που ισχύουν σε όλα τα modules, τα repositories και τα dependencies που είναι κοινά στο project (Εικόνα 29) [28].

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    ext.kotlin_version = '1.5.20'
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:7.0.3'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.5.31"
        classpath 'com.google.gms:google-services:4.3.10'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        mavenCentral()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Εικόνα 28: build.gradle (ThessGuide)

4.7.2 Module Level Build Αρχείο build.gradle (:app)

Στο Module Level (:app) αφορά το συγκεκριμένο Module που είναι τοποθετημένο και δηλώνονται βασικές πληροφορίες, όπως η ελάχιστη απαιτούμενη έκδοση Android που μπορεί να τρέξει η εφαρμογή, το applicationId, η έκδοση της εφαρμογής, το targetSDKVersion και τα dependencies. Τόσο οι βασικές πληροφορίες που αφορούν τις παραμέτρους της εφαρμογής όσο και τα dependencies φαίνονται στις εικόνες 30 και 31 [28].

```
plugins {
    id 'com.android.application'
    id 'com.google.gms.google-services'
    id 'kotlin-android'
}

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.example.thessguide"
        minSdkVersion 21
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
```

Εικόνα 29: build.gradle(:app) (1)

```
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures {
        viewBinding true
    }
}
dependencies {
    //noinspection GradleDependency
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    implementation 'com.google.android.gms:play-services-maps:17.0.1'
    implementation 'com.google.maps.android:android-maps-utils:2.2.0'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.navigation:navigation-fragment-ktx:2.3.5'
    implementation 'androidx.navigation:navigation-ui-ktx:2.3.5'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    implementation platform('com.google.firebase:firebase-bom:26.5.0')
    implementation 'com.google.firebase:firebase-analytics-ktx'
    implementation 'com.google.firebase:firebase-firestore:23.0.4'
    implementation 'androidx.recyclerview:recyclerview:1.2.1'
    implementation 'com.squareup.picasso:picasso:2.71828'
    implementation "androidx.cardview:cardview:1.0.0"
    implementation 'com.google.android.gms:play-services-location:18.0.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    implementation 'com.squareup.okhttp3:okhttp:5.0.0-alpha.2'
}
```

Εικόνα 30: build.gradle(:app) (2)

4.8 Βιβλιοθήκες

Τα dependencies είναι οι βιβλιοθήκες που χρησιμοποιούνται στο project και μόλις δηλωθεί μια βιβλιοθήκη στα dependencies, το Android Studio πρέπει να αναζητήσει την βιβλιοθήκη αυτή στις πηγές που έχουν δηλωθεί στο build.gradle (Project), στα repositories και από την στιγμή αυτή και μετά όλες οι κλάσεις και οι μέθοδοι των βιβλιοθηκών είναι έτοιμες για χρήση. Αξίζει να σημειωθεί ότι με την χρήση βιβλιοθηκών αντικρίζεται ένα αμελητέο αντίκτυπο στην συνολική μνήμη της εφαρμογής [29].

4.8.1 Google Play Services

Τα Google Play Services συνδέουν τις εφαρμογές με άλλες υπηρεσίες της Google, όπως το Google Sign In ή τους Χάρτες της Google. Με την χρήση των βιβλιοθηκών είναι δυνατόν οι χρήστες της εφαρμογής να αποκτήσουν πρόσβαση στις υπηρεσίες της Google χρησιμοποιώντας τα διαπιστευτήριά τους.

Για την συγκεκριμένη εφαρμογή τα google services που αξιοποιούνται είναι οι χάρτες (maps) και η τοποθεσία (location):

```
implementation 'com.google.android.gms:play-services-maps:17.0.1'
implementation 'com.google.android.gms:play-services-location:18.0.0'
```

4.8.2 AndroidX test

Μια ακόμη βιβλιοθήκη που συναντάται στην εφαρμογή είναι η AndroidX. Με αυτή τη βιβλιοθήκη η εφαρμογή έχει την δυνατότητα να τρέχει τις νέες λειτουργίες που έχουν προστεθεί σε νέες εκδόσεις να τρέχουν σωστά και σε παλιότερες εκδόσεις της εφαρμογής.

```
androidTestImplementation 'androidx.test.ext:junit:1.1.3'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
```

4.9 Data classes

Για να μπορέσουν να χρησιμοποιηθούν τα παραπάνω δεδομένα που είναι αποθηκευμένα στο firestore μέσα στην εφαρμογή δημιουργείται μια data class (εικόνα 32 και 33). Οι data classes περιέχουν μόνο μεταβλητές. Οι κλάσεις αυτές είναι απλά δοχεία για δεδομένα που χρησιμοποιούνται από άλλες κλάσεις. Αυτές οι κλάσεις δεν περιέχουν καμία πρόσθετη λειτουργικότητα και δεν μπορούν να λειτουργήσουν ανεξάρτητα με τα δεδομένα που περιέχουν.

```
1  package com.example.thessguide.models
2
3  import com.google.firebase.firestore.DocumentReference
4
5  data class Route(
6          var rid: Double? = null,
7          var rtitle: String? = null,
8          var rduration: Double? = null,
9          var rmonuments: List<DocumentReference> = emptyList(),
10         var rimage: String? = null)
```

Εικόνα 31: Route Data Class

```
1  package com.example.thessguide.models
2
3  import java.io.Serializable
4
5  data class Monument(
6      var mid: Int? = null,
7      var mname: String? = null,
8      var mdescription: String? = null,
9      var mlat: Double? = null,
10     var mlon: Double? = null,
11     var mimage: String? = null,
12     var mwebsite: String? = null) : Serializable
13
```

Εικόνα 32: Monument Data Class

4.10 Activities

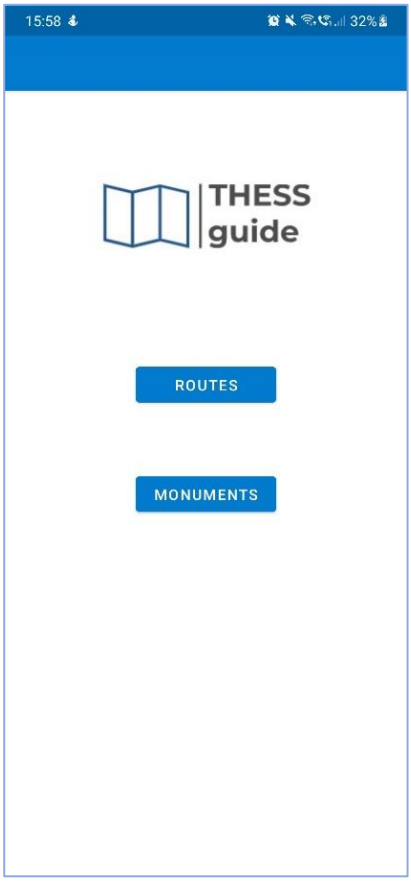
Σε αυτή την ενότητα θα αναλυθούν όλα τα activities που συναντώνται στην εφαρμογή.

4.10.1 Main Activity και Main Menu Fragment

Το MainActivity είναι το Activity που έχει δηλωθεί στο Manifest ως το Activity με το οποίο ξεκινάει η εφαρμογή (Εικόνα 34). Στην MainActivity έχει προστεθεί ένα Fragment που περιλαμβάνει τα περιεχόμενα που βλέπουμε στην αρχική οθόνη της εφαρμογής όπως φαίνεται στην εικόνα 35.

```
40      <activity android:name=".ui.MainActivity">
41          <intent-filter>
42              <action android:name="android.intent.action.MAIN" />
43              <category android:name="android.intent.category.LAUNCHER" />
44          </intent-filter>
45      </activity>
```

Εικόνα 33: Main Launcher



Εικόνα 34: MainActivity με το MainMenuFragment

Στην κλάση της MainActivity (Εικόνα 36), στην γραμμή 24 αρχικοποιείται το Cloud Firestore και στις γραμμές 25 με 32 προστίθεται το MainMenuFragment στο fragment container που έχει το MainActivity. Ενώ στην κλάση του MainMenuFragment (Εικόνα 37) αρχικοποιούνται τα κουμπιά της αρχικής οθόνης και φαίνονται οι ενέργειες που συμβαίνουν όταν πατήσει ο χρήστης το κάθε κουμπί. Πιο συγκεκριμένα αν πατηθεί το κουμπί με id bn_monuments ξεκινάει το Activity με τίτλο MonumentsActivity ενώ αν πατηθεί το κουμπί με id bn_routes ξεκινάει το Activity με τίτλο RoutesActivity.

```
11  class MainActivity : AppCompatActivity() {
12      companion object {
13          @SuppressLint( ...value: "StaticFieldLeak")
14          lateinit var db : FirebaseFirestore
15          lateinit var fm: FragmentManager
16      }
17
18      override fun onCreate(savedInstanceState: Bundle?) {
19          super.onCreate(savedInstanceState)
20          setContentView(R.layout.activity_main)
21
22          supportActionBar?.title = ""
23
24          db = FirebaseFirestore.getInstance()
25          fm = supportFragmentManager
26
27          if(findViewById<View>(R.id.fragment_container)!=null){
28              if (savedInstanceState!=null){
29                  return
30              }
31              fm.beginTransaction().add(R.id.fragment_container, MainMenuFragment()).commit()
32          }
33      }
34  }
```

Εικόνα 35: Κλάση MainActivity

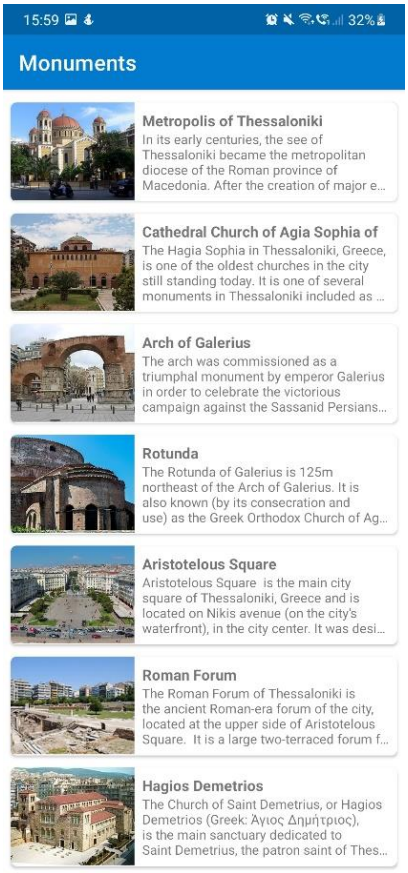
```
12  class MainMenuFragment : Fragment(), View.OnClickListener {
13      private lateinit var routesBn: Button
14      private lateinit var monumentsBn: Button
15
16      override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
17          // Inflate the layout for this fragment
18          val view = inflater.inflate(R.layout.fragment_main_menu, container,  attachToRoot: false)
19          routesBn = view.findViewById(R.id.bn_routes)
20          routesBn.setOnClickListener(this)
21          monumentsBn = view.findViewById(R.id.bn_monuments)
22          monumentsBn.setOnClickListener(this)
23
24          return view
25      }
26
27      override fun onClick(v: View?) {
28          if (v != null) {
29              when(v.id){
30                  R.id.bn_monuments -> startActivity(Intent(context, MonumentsActivity::class.java))
31                  R.id.bn_routes -> startActivity(Intent(context, RoutesActivity::class.java))
32              }
33          }
34      }
35  }
```

Εικόνα 36: Κλάση MainMenuFragment

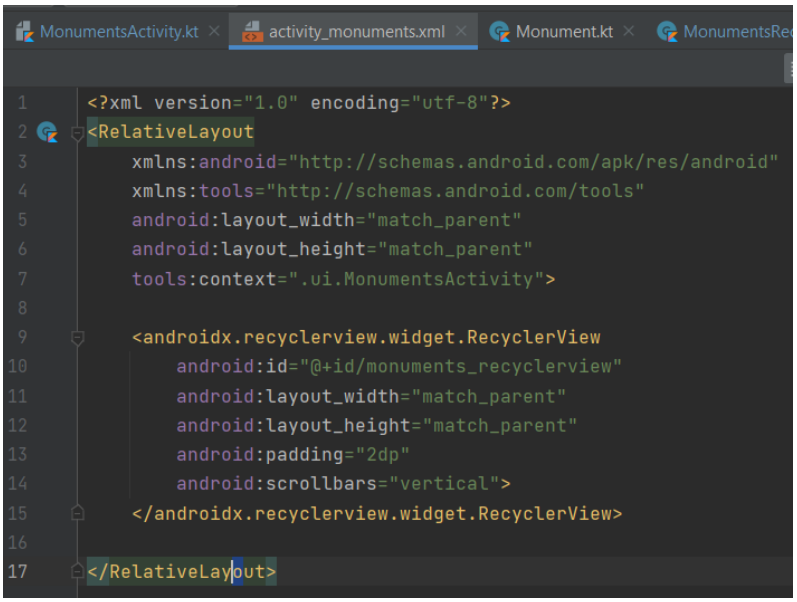
4.10.2 Monuments Activity

Όταν ο χρήστης επιλέξει το κουμπί Monuments που βρίσκεται στην αρχική διεπαφή εμφανίζεται το MonumentsActivity που περιέχει μία λίστα από διάφορα μνημεία, η λίστα αυτή εμφανίζεται μέσω ενός RecyclerView όπως φαίνεται στις εικόνες 38 και 39.

Πιο συγκεκριμένα στην εικόνα 39 βλέπουμε το xml αρχείο που δείχνει ότι το MonumentsActivity αποτελείται μόνο από ένα κάθετο RecyclerView. Αυτό το RecyclerView αποτελείται από έναν αριθμό «κελιών» (monument_cell_layout.xml) ίσο με το πλήθος των μνημείων που βρίσκονται στην βάση Firebase, το κάθε κελί είναι ένα CardView που περιέχει ένα ImageView και δύο TextView, ένα για τον τίτλο και ένα για την περιγραφή του μνημείου.



Εικόνα 37: Monuments Activity UI



Εικόνα 38: activity_monuments.xml

Για να γεμίσει το RecyclerView με δεδομένα έπρεπε πρώτα να ανακτηθούν από το Firebase, αυτό γίνεται στις γραμμές 32 με 44 του κώδικα της εικόνας 40. Σε αυτές τις γραμμές κώδικα καλείται η κλάση MonumetsRecyclerAdapter(List<Monument>, OnMonumentClickListener). Στην κλάση αυτή, της οποίας ο κώδικας φαίνεται στην εικόνα 41, δημιουργούνται τα view holder που στην προκυμμένη περίπτωση είναι CardView που περιέχουν ένα ImageView και δύο TextView. Τα view γεμίζουν με τα δεδομένα που περνάνε στην κλάση μέσω του List<Monument> και εμφανίζονται στην οθόνη του χρήστη. Αν ο χρήστης πατήσει σε ένα από αυτά τα CardView καλείται η μέθοδος onItemClicked(Monument) που βρίσκεται στην κλάση MonumentsActivity (εικόνα 42) και ο χρήστης μεταφέρεται στο MonumentDetailsActivity.

```
const val MONUMENT_EXTRA = "monument"

class MonumentsActivity : AppCompatActivity(), OnMonumentClickListener {
    private lateinit var recyclerView: RecyclerView
    private lateinit var layoutManager: LinearLayoutManager
    private lateinit var monumentList: List<Monument>
    private lateinit var monumentsRecyclerAdapter: MonumentsRecyclerAdapter
    private lateinit var collectionReference: CollectionReference

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_monuments)
        supportActionBar?.title = "Monuments"
        recyclerView = findViewById(R.id.monuments_recyclerview)
        layoutManager = LinearLayoutManager( context: this)
        recyclerView.layoutManager = layoutManager
        collectionReference = MainActivity.db.collection( collectionPath: "Monuments")
        collectionReference.get().addOnSuccessListener {  it: QuerySnapshot!
            if (!it.isEmpty){
                monumentList =  it.toObjects(Monument::class.java)
                monumentsRecyclerAdapter = MonumentsRecyclerAdapter(monumentList,  onMonumentClickListener: this)
                recyclerView.hasFixedSize()
                recyclerView.adapter = monumentsRecyclerAdapter
            } else {
                Toast.makeText(applicationContext,  text: "collection doesn't exist", Toast.LENGTH_LONG).show()
            }
        }.addOnFailureListener { exception ->
            Toast.makeText(applicationContext,  text: "query operation failed: $exception", Toast.LENGTH_LONG).show()
        }
    }
```

Εικόνα 39: Κλάση MonumentsActivity

```
class MonumentsRecyclerAdapter(private var list: List<Monument>, private val onMonumentClickListener: OnMonumentClickListener) :
    RecyclerView.Adapter<MonumentsRecyclerAdapter.MonumentsViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MonumentsViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.monument_cell_layout, parent, attachToRoot: false)
        return MonumentsViewHolder(view)
    }

    override fun onBindViewHolder(holder: MonumentsViewHolder, position: Int) {
        holder.bind(list[position], onMonumentClickListener)
    }


    override fun getItemCount(): Int {
        return list.size
    }

    class MonumentsViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        private val mname: TextView = itemView.findViewById(R.id.titleTextView)
        private val mdescription: TextView = itemView.findViewById(R.id.descriptionTextView)
        private val mimage: ImageView = itemView.findViewById(R.id.monumentImageView)

        fun bind(monument: Monument, clickListener: OnMonumentClickListener){
            mname.text = monument.mname
            mdescription.text = monument.mdescription
            Picasso.get().load(monument.mimage).into(mimage)

            itemView.setOnClickListener{  it: View!
                clickListener.onItemClicked(monument)
            }
        }
```

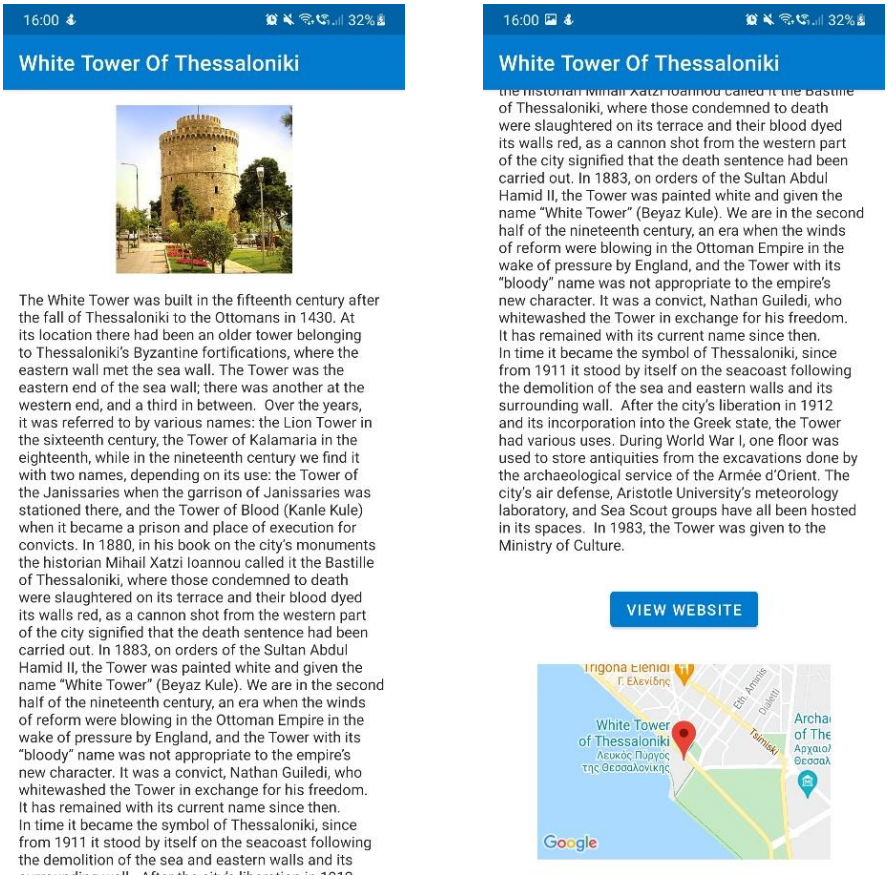
Εικόνα 40: Κλάση MonumentsRecyclerAdapter

```
45    override fun onItemClicked(monument: Monument) {
46        val intent = Intent( packageContext: this, MonumentDetailsActivity::class.java)
47        intent.putExtra(MONUMENT_EXTRA, monument)
48        startActivity(intent)
49    }
50 }
```

Εικόνα 41: Μέθοδος onItemClicked(Monument)

4.10.3 Monument Details Activity

Μόλις επιλεγεί ένα Monument από το MonumentsRecyclerView ο χρήστης μεταφέρεται σε μια διεπαφή που περιέχει όλες τις σημαντικές πληροφορίες του μνημείου. Όπως φαίνεται και στην εικόνα 43 η διεπαφή του MonumentDetailsActivity αποτελείται αρχικά από ένα ScrollView περιέχει όλα τα view που θα αναφερθούν σε λίγο, αυτό συμβαίνει γιατί λόγω των πολλών πληροφοριών που περιέχονται μέσα σε αυτή την διεπαφή χρειαζόμαστε περισσότερη πληροφορία από μια μόνο οθόνη. Μέσα στο ScrollView περιέχονται ένα ImageView που απεικονίζει το εν λόγω μνημείο, ένα TextView που περιέχει την περιγραφή του μνημείου του οποίου το μέγεθος είναι μεταβαλλόμενο ανάλογα το μέγεθος του κειμένου, ένα Button που όπως θα δούμε και στον κώδικα του MonumentDetailsActivity μας μεταφέρει στο website του μνημείου και στο κάτω μέρος της διεπαφής βλέπουμε ένα μικρό MapView που μας δείχνει την θέση του μνημείου στον χάρτη και έχουμε την δυνατότητα να πλοηγηθούμε προς αυτό το σημείο πατώντας πάνω στον χάρτη και ανοίγοντας τους Χάρτες της Google.



Εικόνα 42: Monument Details Activity

Για την προσθήκη του Marker στον μικρό χάρτη χρειάστηκαν οι παρακάτω γραμμές κώδικα που φαίνονται στην εικόνα 44. Στην γραμμή 88 κρατάμε το στίγμα του σημείου σε μια μεταβλητή. Στην γραμμή 89 προσθέτουμε τον Marker στο στίγμα που κρατήσαμε και στην γραμμή 90 προσδιορίζουμε την απόσταση της «κάμερας» από το σημείο. Υπάρχουν πέντε επίπεδα zoom και είναι τα εξής:

1: Κόσμος
5: Περιοχή στεριάς/ Ήπειρος
10: Πόλη
15: Δρόμοι
20: Κτήρια

Στην συγκεκριμένη περίπτωση επιλέχθηκε το επίπεδο 15 για να μπορεί ο χρήστης να προσανατολιστεί αναγνωρίζοντας τα γύρω σημεία.

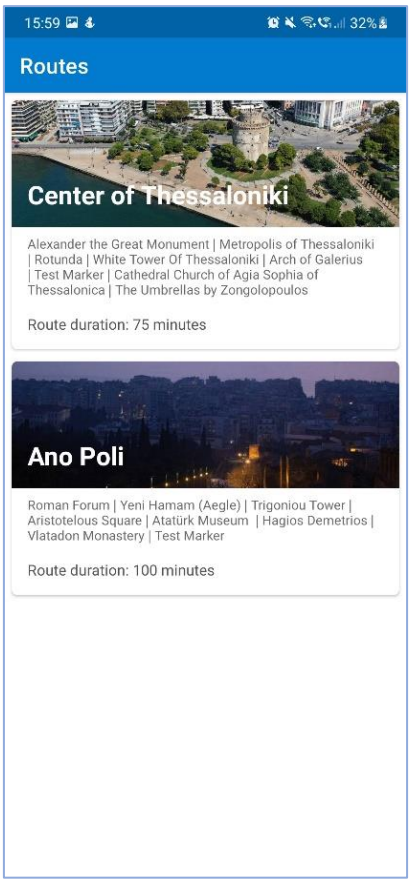
```
87    override fun onMapReady(map: GoogleMap) {
88        val monumentLocation = LatLng(monument.mlat!!, monument.mlon!!)
89        map.addMarker(MarkerOptions().position(monumentLocation))
90        map.animateCamera(CameraUpdateFactory.newLatLngZoom(monumentLocation,  zoom: 15F))
91    }
```

Εικόνα 43: Προσθήκη Marker στον χάρτη

4.10.4 Routes Activity

Επιστρέφοντας στην πρώτη οθόνη στην MainActivity αν ο χρήστης πατήσει στο κουμπί Routes μεταφέρεται στο RoutesActivity. Σε αυτό το activity παρατηρούμε πολλές ομοιότητες με το MonumentsActivity. Πιο αναλυτικά, βλέπουμε ένα RecyclerView, που περιέχει τόσα CardViews όσα και οι διαδρομές και σε κάθε CardView βλέπουμε ένα ImageView και 3 TextView, ένα για τον τίτλο της διαδρομής, ένα που να «περιγράφει» την διαδρομή, όπου στην ουσία απλά αναφέρονται όλα τα σημεία ενδιαφέροντος από τα οποία θα περάσει ο χρήστης και το τελευταίο TextView όπου φαίνεται ο χρόνος που θα χρειαστεί ο χρήστης για να την ολοκληρώσει.

Όπως φαίνεται και από την περιγραφή της διεπαφής αλλά και από την εικόνα 45 δεν συναντώνται μεγάλες διαφορές με το MonumentsActivity. Όμως έχει ενδιαφέρον να δούμε πως δημιουργείται το κείμενο που βλέπουμε στο TextView της περιγραφής. Παρατηρώντας τις γραμμές κώδικα που φαίνονται στην εικόνα 46 βλέπουμε ότι δημιουργείται μια λίστα names της οποία τα περιεχόμενα είναι μεταβαλλόμενα, δηλαδή mutable και από την λίστα rmonuments: List<DocumentReference> κάνουμε map, δηλαδή περνάμε από κάθε document snapshot στοιχείο της λίστας και κρατάμε μόνο το όνομα του μνημείου (mname), μέσα στην λίστα names. Στο τέλος μετατρέπουμε την λίστα names σε συμβολοσειρά String.



Εικόνα 44: Διεπαφή RoutesActivity

```
41          val names : MutableList<String?> = mutableListOf()
42
43          route.rmonuments.map { documentReference ->
44              documentReference.get().addOnSuccessListener {  it: DocumentSnapshot!
45                  names.add(it.toObject(Monument::class.java)?.mname)
46                  rdescription.text = names.joinToString( separator: " | ")
47              }
48          }
```

Εικόνα 45: Γέμισμα του TextView της περιγραφής

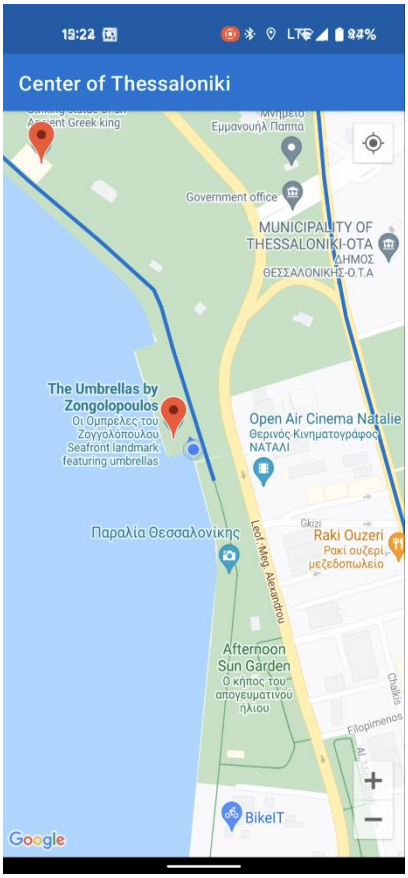
```
50  override fun onItemClicked(route: Route) {
51      val intent = Intent( packageContext: this, MapsActivity::class.java)
52      val monumentPaths : ArrayList<String> = route.rmonuments.map {  it: DocumentReference
53          it.path
54      } as ArrayList<String>
55      intent.putStringArrayListExtra( name: "monuments", monumentPaths)
56      intent.putExtra( name: "routeTitle", route.rtitle)
57      intent.putExtra( name: "routeID", route.rid)
58      startActivity(intent)
59  }
```

Εικόνα 46: Κλήση του MapsActivity

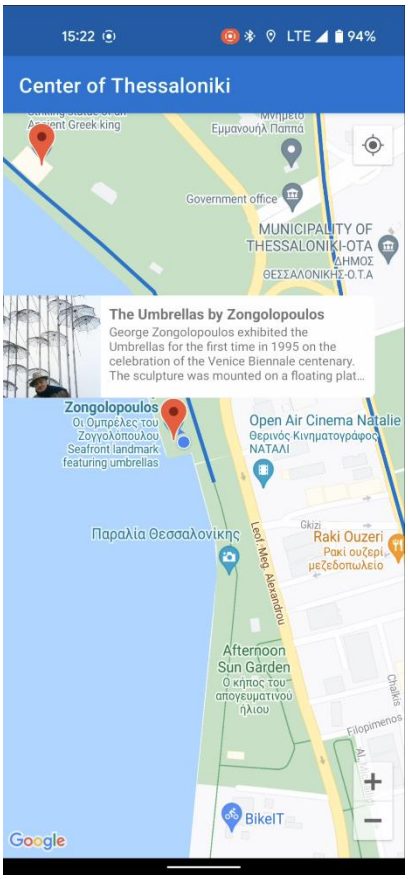
4.10.5 Map Activity

Όταν ο χρήστης επιλέξει μια από τις διαδρομές μεταφέρεται στο MapsActivity και όπως φαίνεται στην εικόνα 47 με την κλήση του activity μεταφέρονται και πληροφορίες όπως ένα ArrayList με τα μνημεία της διαδρομής, ο τίτλος της διαδρομής και το id της.

Το MapsActivity περιέχει τις βασικές λειτουργίες της εφαρμογής και αυτές θα αναλυθούν σε αυτή την ενότητα. Όπως φαίνεται και στην εικόνα 48 αυτό που βλέπει ο χρήστης όταν επιλέγει μια διαδρομή, είναι μια διεπαφή χάρτη με κάποιους marker και μια μπλε γραμμή που υποδεικνύει στον χρήστη μια διαδρομή από την τρέχουσα τοποθεσία του μέχρι το τελευταίο σημείο ενδιαφέροντος, περνώντας όμως πρώτα απ' όλα τα υπόλοιπα. Όταν ο χρήστης πλησιάσει σε ένα από τα σημεία ενδιαφέροντος τότε όπως φαίνεται στην εικόνα 49 εμφανίζεται ένα custom InfoWindow που περιέχει ένα ImageView και δύο TextView.



Εικόνα 47: Διαδρομή (Center of Thessaloniki)



Εικόνα 48: Επαύξηση Χάρτη με Info Window

Ας αναλύσουμε όμως πως δουλεύουν όλα αυτά που βλέπει ο χρήστης στην οθόνη της κινητής του συσκευής. Τα δύο στοιχεία που είναι απαραίτητα για την έναρξη της διεπαφής είναι η τρέχουσα τοποθεσία του χρήστη ανά πάσα στιγμή και τα μνημεία που συναντώνται στην συγκεκριμένη διαδρομή. Όπως φαίνεται στην onCreate() μέθοδο της κλάσης MapsActivity (Εικόνα 50). Στην σειρά 75 καλείται η μέθοδος createLocationRequest(). Στην οποία, όπως φαίνεται στην εικόνα 51 δηλώνονται ο χρόνος που θα μεσολαβεί ανάμεσα στις ενημερώσεις της τρέχουσας τοποθεσίας της συσκευής, η ποιότητα του σήματος της τοποθεσίας (GPS ή WiFi/ Cell tower positioning) και από την στιγμή που έχουν οριστεί αυτές οι τιμές και οι ρυθμίσεις της τοποθεσίας ανταποκρίνονται καλείται η μέθοδος startLocationUpdates() (Εικόνα 52) στην οποία ελέγχεται αν έχουν δωθεί τα απαραίτητα permissions και αφού επιβεβαιωθούν ξεκινάει η εφαρμογή να ζητάει ενημερώσεις της τοποθεσίας.

```
55
56  override fun onCreate(savedInstanceState: Bundle?) {
57      super.onCreate(savedInstanceState)
58      setContentView(R.layout.activity_maps)
59      // Obtain the SupportMapFragment and get notified when the map is ready to be used.
60      val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as SupportMapFragment
61      mapFragment.getMapAsync( callback: this)
62
63      fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
64
65      supportActionBar?.title = intent.getStringExtra( name: "routeTitle")
66      monumentsDR = intent.getStringArrayListExtra( name: "monuments")!!.map {  FirebaseFirestore.getInstance().document(it) }
67
68      locationCallback = object : LocationCallback() {
69          override fun onLocationResult(p0: LocationResult) {
70              super.onLocationResult(p0)
71              lastLocation = p0.lastLocation
72          }
73      }
74
75      createLocationRequest()
76
77  }
78
```

Εικόνα 49: Μέθοδος onCreate() στο MapsActivity

```
178     private fun createLocationRequest() {
179         locationRequest = LocationRequest()
180         //specifies the rate that the app will receive  updates
181         locationRequest.interval = 3000
182         //specifies the fastest rate at which the app can handle updates
183         locationRequest.fastestInterval = 1000
184         locationRequest.priority = LocationRequest.PRIORITY_HIGH_ACCURACY
185
186         val builder = LocationSettingsRequest.Builder()
187                 .addLocationRequest(locationRequest)
188
189         val client = LocationServices.getSettingsClient(this)
190         val task = client.checkLocationSettings(builder.build())
191
192         //if task -> success, location -> updates
193         task.addOnSuccessListener {  it: LocationSettingsResponse!
194             locationUpdateState = true
195             startLocationUpdates()
196         }
197
198         //if task -> fail, open dialog to fix problem
199         task.addOnFailureListener { e ->
200             if (e is ResolvableApiException) {
201                 // Location settings are not satisfied, but this can be fixed
202                 // by showing the user a dialog.
203                 try {
204                     // Show the dialog by calling startResolutionForResult(),
205                     // and check the result in onActivityResult().
206                     e.startResolutionForResult(this@MapsActivity,
207                             REQUEST_CHECK_SETTINGS
208                     )
209                 } catch (sendEx: IntentSender.SendIntentException) {
210                     // Ignore the error.
```

Εικόνα 50: Μέθοδος createLocationRequest()

```
146     private fun startLocationUpdates() {
147         if (ActivityCompat.checkSelfPermission( context: this,
148                         android.Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
149             ActivityCompat.requestPermissions( activity: this,
150                     arrayOf(android.Manifest.permission.ACCESS_FINE_LOCATION),
151                     LOCATION_PERMISSION_REQUEST_CODE
152             )
153             return
154         }
155
156         fusedLocationClient.requestLocationUpdates(locationRequest, locationCallback, Looper.getMainLooper())
157     }
```

Εικόνα 51: Μέθοδος startLocationUpdates()

Όταν ξεκινάει το Map Activity μια από τις μεθόδους που καλούνται είναι η onMapReady() (Εικόνα 53) σε αυτή τη μέθοδο αρχικά λαμβάνονται τα user interface settings για τον χάρτη και ενεργοποιούνται οι λειτουργίες zoom του χάρτη. Μετά η λίστα monuments γεμίζει με αντικείμενα τύπου Monument και όταν ολοκληρωθεί η διαδικασία, αν η μεταβλητή lastlocation που περιέχει την τελευταία τοποθεσία του χρήστη δεν είναι κενή καλείται η μέθοδος createRoute(List<Monument>) (Εικόνα 54).

Σε αυτή την μέθοδο αφού ταξινομηθούν τα μνημεία με βάση το id τους, για κάθε μνημείο προστίθεται στον χάρτη ένας Marker με βάση τις συντεταγμένες του μνημείου. Για κάθε marker αποθηκεύονται και κάποιες ακόμη πληροφορίες, όπως είναι ο τίτλος του σημείου ενδιαφέροντος, η εικόνα του καθώς και μια σύντομη περιγραφή του σημείου ενδιαφέροντος. Αφού ανακτηθούν αυτές οι πληροφορίες τις περνάμε στο Info Window του κάθε marker. Το οποίο είναι custom, δηλαδή δεν είναι όπως το default info window που περιέχει άπλα ένα TextView, αλλά όπως βλέπουμε και στην εικόνα 49 ο χρήστης βλέπει ένα ImageView και δύο TextView.

Στην εικόνα 55 φαίνεται μέρος του CustomInfoWindowAdapter() και πιο συγκεκριμένα η μέθοδος windowText() που αναθέτει τις τιμές σε κάθε view.

Πίσω στην μέθοδο createRoute(List<Monument>) (Εικόνα 54), στις γραμμές από 93 μέχρι 100 βλέπουμε ότι αν ο χρήστης πατήσει πάνω σε ένα από αυτά τα Info Windows τότε θα μεταφερθεί στο MonumentDetailsActivity() του συγκεκριμένου σημείου ενδιαφέροντος.

```
158
159     override fun onMapReady(googleMap: GoogleMap) {
160         map = googleMap
161         map.uiSettings.isZoomControlsEnabled = true
162
163         monumentsDR.map { documentReference ->
164             documentReference.get().addOnCompleteListener {  it: Task<DocumentSnapshot!>
165                 monuments.add(it.result.toObject(Monument::class.java)!!)
166                 if(monuments.size == monumentsDR.size){
167                     if(lastLocation != null) {
168                         createRoute(monuments)
169                     } else {
170                         createLocationRequest()
171                     }
172                 }
173             }
174         }
175         setUpMap()
176     }
177
```

Εικόνα 52: Μέθοδος onMapReady()

```
78
79      //ADD MARKERS AND CREATE ROUTE
80      @SuppressLint( ...value: "PotentialBehaviorOverride")
81      private fun createRoute(monuments: List<Monument>){
82          val sortedMonuments = monuments.sortedBy {
83              monument: Monument -> monument.mid
84          }
85          for(m in sortedMonuments){
86              val monumentLocation = LatLng(m.mlat!!, m.mlon!!)
87              path.add(monumentLocation)
88              markers.add(map.addMarker(MarkerOptions()
89                  .position(monumentLocation).title(m.mname).snippet( snippet: m.mimage+"@"+m.mdescription)))
90              map.setInfoWindowAdapter(CustomInfoWindowAdapter( context: this@MapsActivity))
91          }
92
93          map.setOnInfoWindowClickListener{  it: Marker
94              for(monument in sortedMonuments)
95                  if(it.title == monument.mname){
96                      val intent = Intent( packageContext: this@MapsActivity, MonumentDetailsActivity::class.java)
97                      intent.putExtra( name: "monument", monument)
98                      startActivity(intent)
99                  }
100         }
101
102         val currentLocation = LatLng(lastLocation.latitude, lastLocation.longitude)
103         val url = getURL(currentLocation, path.last(), path , mode: "walking")
104         GetDirection(url).execute()
105     }
106
```

Εικόνα 53: Μέθοδος createRoute(List<Monument>)

```
16  class CustomInfoWindowAdapter() : AppCompatActivity(), GoogleMap.InfoWindowAdapter {
17      private lateinit var mWindow: View
18      private lateinit var mContext: Context
19
20      @SuppressLint( ...value: "InflateParams")
21      constructor(context: Context) : this() {
22          mContext = context
23          mWindow = LayoutInflater.from(context).inflate(R.layout.activity_custom_info_window,  root: null)
24      }
25
26      override fun onCreate(savedInstanceState: Bundle?) {
27          super.onCreate(savedInstanceState)
28          setContentView(R.layout.activity_custom_info_window)
29      }
30
31      private fun windowText(marker: Marker, view: View){
32          val image = marker.snippet.substringBefore( delimiter: '@')
33          val imageViewImage: ImageView = view.findViewById(R.id.customInfoWindowImage)
34
35          Picasso.get().load(image).into(imageViewImage)
36
37          val title = marker.title
38          val textViewTitle: TextView = view.findViewById(R.id.customInfoWindowTitle)
39
40          if(title!=null){
41              textViewTitle.text = title
42          }
43
44          val snippet = marker.snippet.substringAfter( delimiter: '@')
45          val textViewSnippet: TextView = view.findViewById(R.id.snippet)
46
47          textViewSnippet.text = snippet
48      }
```

Εικόνα 54: Κλάση CustomInfoWindowAdapter()

Στην μέθοδο createRoute(List<Monument>) (Εικόνα 54), στην γραμμή 102 μετατρέπουμε το Location σε LanLng και στην γραμμή 103 καλούμε την μέθοδο getURL(LanLng, MutableList<LanLng>, String) (Εικόνα 56), η οποία επιστρέφει ένα String που είναι το URL του Directions API που μας δίνει ένα JSON αρχείο (Εικόνα 57).

Για την δημιουργία αυτού του URL είναι απαραίτητα κάποια στοιχεία στην κατάλληλη μορφή για να μπορέσουν να μπουν μέσα στο String που θα είναι το τελικό URL. Αρχικά είναι απαραίτητες οι συντεταγμένες της τρέχουσας τοποθεσίας του χρήστη με την μορφή που φαίνεται στην γραμμή 247, δηλαδή αποτελείται από το String «origin=» και ακολουθούν το γεωγραφικό πλάτος και το γεωγραφικό μήκος χωρισμένα με κόμμα. Ακολουθεί ο προορισμός του χρήστη όπως φαίνεται στην γραμμή 248 που εκφράζεται με παρόμοιο τρόπο με την αφετηρία.

Στην συγκεκριμένη εφαρμογή η διαδρομή του χρήστη δεν εξαρτάται από δύο σημεία, την αφετηρία και τον τερματισμό αλλά σημασία έχουν και οι ενδιάμεσες στάσεις. Για αυτό τον λόγο δημιουργούμε το String waypointsToStr το οποίο στην αρχή περιέχει την λέξη «waypoints=» και στην συνέχεια αναγράφεται το γεωγραφικό πλάτος και το γεωγραφικό μήκος της πρώτης στάσης χωρισμένα με κόμμα και μετά ακολουθεί μία κατακόρυφη μπάρα « | » για να την διαχωρίσει από τις συντεταγμένες της επόμενης στάσης.

Στο τέλος των παραμέτρων του URL διευκρινίζεται ο τρόπος που θα μετακινηθεί ο χρήστης και στην συγκεκριμένη εφαρμογή θα είναι πάντα με τα πόδια.

Όλοι αυτοί οι παράμετροι μπαίνουν στην σειρά χωρισμένοι με το σύμβολο του ampersand « & », πριν από αυτές τις παραμέτρους διευκρινίζεται τι μορφή θέλουμε να επιστρέφει το API call και μετά από τις παραμέτρους δηλώνεται το API Key και τελικά προκύπτει το παρακάτω URL.

```
245     //DRAW ROUTE
246     private fun getURL(origin: LatLng, destination: LatLng, waypoints: MutableList<LatLng>, mode: String) : String {
247         val strOrigin = "origin=" + origin.latitude + "," + origin.longitude
248         val strDestination = "destination=" + destination.latitude + "," + destination.longitude
249         waypoints.removeLast()
250         var wayPointsToStr = "waypoints="
251         for(point in waypoints) {
252             wayPointsToStr += (point.latitude.toString() + "," + point.longitude + "|")
253         }
254         val strWayPoints = wayPointsToStr.removeSuffix( suffix: "|")
255         print(strWayPoints)
256         val strMode = "mode=$mode"
257         val parameters = "$strOrigin&$strDestination&$strWayPoints&$strMode"
258         val output = "json"
259
260         print("\n https://maps.googleapis.com/maps/api/directions/$output?$parameters&sensor=true&key=${getString(
261             R.string.google_maps_key
262         )} \n ")
263
264         return "https://maps.googleapis.com/maps/api/directions/$output?$parameters&sensor=true&key=${getString(
265             R.string.google_maps_key
266         )}"
267     }
```

Εικόνα 55: Μέθοδος getURL(LanLng, MutableList<LanLng>, String)

```
{
   "geocoded_waypoints" : [
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJqzUkDB4VqBQRa4klaWktlBw",
         "types" : [ "premise" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJ-ZtKbR05qBQRh3cL6pAGktc",
         "types" : [ "street_address" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJbwAeGQM5qBQRF3qkBMXLiO8",
         "types" : [ "premise" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJX8svogM5qBQRVZiNiBz94gw",
         "types" : [ "street_address" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJPRlWYgY5qBQRNN9qCJVp8Gs",
         "types" : [ "establishment", "point_of_interest" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJRezMSwc5qBQRjWjq_bPueck",
         "types" : [ "street_address" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJMSSAKqk5qBQRwssZ-H1wH6M",
         "types" : [ "establishment", "point_of_interest", "tourist_attraction" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJAVucuao5qBQRXs6-aoiqS9A",
         "types" : [ "street_address" ]
      }
   ],
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
                  "lat" : 40.6333583,
                  "lng" : 22.9952276
               },
               "southwest" : {
                  "lat" : 40.4756465,
                  "lng" : 22.9438633
               }
            },
            "copyrights" : "Map data ©2021",
            "legs" : [
               {
                  "distance" : {
                     "text" : "18.2 km",
                     "value" : 18151
                  },
                  "duration" : {
                     "text" : "3 hours 39 mins",
                     "value" : 13141
                  },
```


Εικόνα 56: JSON Αρχείο

https://maps.googleapis.com/maps/api/directions/json?origin=40.475859,22.9713778&destination=40.633254334384574,22.952871519544534&waypoints=40.621865656414414,22.951385434883793|40.62405649881606,22.95000475682429|40.6264252047587,22.94844132126022|40.630611343455776,22.944482351498483|40.63278672958328,22.946998702686766|40.63218760618888,22.951754721560913&mode=walking&key=GOOGLE_MAPS_KEY

Αυτό το URL περνάει σαν παράμετρος στην inner class GetDirection(String) (Εικόνα 58) και σε αυτή την κλάση υπάρχουν μέθοδοι που καλούν το URL που δημιουργήσαμε, παίρνουν το JSON (Εικόνα 57) που επιστρέφει και δημιουργούν την διαδρομή με την βοήθεια της μεθόδου decodePolyline() (Εικόνα 60) ενώ καθορίζονται τα χαρακτηριστικά της γραμμής, όπως το χρώμα και το πάχος, ενώ ζωγραφίζει την γραμμή, αυτά γίνονται με την βοήθεια της μεθόδου onPostExecute() (Εικόνα 59).

```
271     override fun doInBackground(vararg params: Void?): List<List<LatLng>> {
272         val client = OkHttpClient()
273         val request = Request.Builder().url(url).build()
274         val response = client.newCall(request).execute()
275         val data = response.body!!.string()
276         Log.d( tag: "GoogleMap" ,  msg: " data : $data")
277         val result =  ArrayList<List<LatLng>>()
278         try{
279             val respObj = Gson().fromJson(data, GoogleMapsDTO::class.java)
280
281             val path =  ArrayList<LatLng>()
282
283             for (i in 0 until respObj.routes[0].legs.size){
284                 for(j in 0 until respObj.routes[0].legs[i].steps.size) {
285                     path.addAll(decodePolyline(respObj.routes[0].legs[i].steps[j].polyline.points))
286                     instructions.add(respObj.routes[0].legs[i].steps[j])
287                 }
288             }
289             result.add(path)
290         }catch (e:Exception){
291             e.printStackTrace()
292         }
293         return result
294     }
```

Εικόνα 57: inner class GetDirection(String) και η μέθοδος doInBackground(Void?): List<List<LatLng>>

```
296     override fun onPostExecute(result: List<List<LatLng>>) {
297         val lineOption = PolylineOptions()
298         for (i in result.indices){
299             lineOption.addAll(result[i])
300             lineOption.width( width: 10f)
301             lineOption.color(resources.getColor(R.color.colorPrimary))
302             lineOption.geodesic( geodesic: true)
303             lineOption.points
304         }
305         map.addPolyline(lineOption)
306     }
307 }
```

Εικόνα 58: μέθοδος onPostExecute()

```
309     fun decodePolyline(encoded: String): List<LatLng> {
310         val poly = ArrayList<LatLng>()
311         var index = 0
312         val len = encoded.length
313         var lat = 0
314         var lng = 0
315
316         while (index < len) {
317             var b: Int
318             var shift = 0
319             var result = 0
320
321             do {
322                 b = encoded[index++].toInt() - 63
323                 result = result or (b and 0x1f shl shift)
324                 shift += 5
325             } while (b >= 0x20)
326
327             val dlat = if (result and 1 != 0) (result shr 1).inv() else result shr 1
328             lat += dlat
329
330             shift = 0
331             result = 0
332             do {
333                 b = encoded[index++].toInt() - 63
334                 result = result or (b and 0x1f shl shift)
335                 shift += 5
336             } while (b >= 0x20)
337             val dlng = if (result and 1 != 0) (result shr 1).inv() else result shr 1
338             lng += dlng
339
340             val latLng = LatLng((lat.toDouble() / 1E5),(lng.toDouble() / 1E5))
341             poly.add(latLng)
342         }
343
344         return poly
345     }
346
347 }
```

Εικόνα 59: μέθοδος decodePolyline()

Αφού σημειώθηκαν τα σημεία ενδιαφέροντος στον χάρτη, δημιουργήθηκαν τα custom παράθυρα πληροφοριών και δημιουργήθηκε η διαδρομή που ενώνει όλα αυτά τα σημεία, πρέπει να υλοποιηθεί και το κομμάτι της επαύξησης του περιεχομένου του χάρτη με βάση την τοποθεσία του χρήστη. Για αυτόν τον λόγο δημιουργήθηκε η μέθοδος handleInfoWindows() (Εικόνα 62). Στην μέθοδο αυτή η οποία καλείται κάθε 10 δευτερόλεπτα γίνεται ένα loop που εσωτερικά συγκρίνεται η τρέχουσα τοποθεσία του χρήστη με την τοποθεσία κάθε ενός σημείου ενδιαφέροντος, αν η απόσταση αυτών των δύο σημείων είναι μικρότερη από είκοσι μέτρα τότε το παράθυρο πληροφοριών του συγκεκριμένου μνημείου ανοίγει, όσο ο χρήστης παραμένει σε ακτίνα μικρότερη ή ίση των 20 μέτρων το παράθυρο παραμένει ανοιχτό και θα κλείσει μόνο όταν ο χρήστης ξεφύγει από αυτά τα όρια.

Όπως αναφέρθηκε και νωρίτερα η μέθοδος αυτή καλείται κάθε δέκα δευτερόλεπτα μέσα από την μέθοδο onResume() (Εικόνα 61).

```
107     //this method is called every 10 seconds
108     private fun handleInfoWindows() {
109         markers.forEach {  it: Marker
110             val markerLocation = Location(LocationManager.GPS_PROVIDER)
111             markerLocation.latitude = it.position.latitude
112             markerLocation.longitude = it.position.longitude
113             if(lastLocation.distanceTo(markerLocation)<=20F){
114                 it.showInfoWindow()
115             } else if(lastLocation.distanceTo(markerLocation)>20F){
116                 it.hideInfoWindow()
117             }
118         }
119     }
```

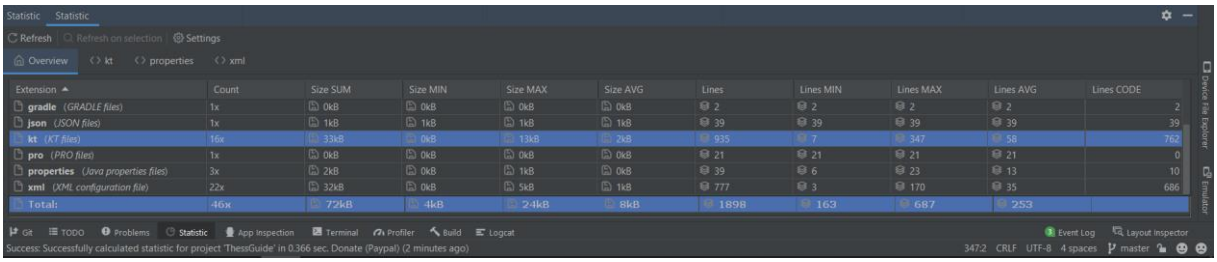
Εικόνα 610: Μέθοδος handleInfoWindows()

```
232     public override fun onResume() {
233         super.onResume()
234         handler.postDelayed(Runnable {
235             handler.postDelayed(runnable!!, delay.toLong())
236             handleInfoWindows()
237         }.also { runnable = it }, delay.toLong())
238
239         if (!locationUpdateState) {
240             startLocationUpdates()
241         }
242     }
```

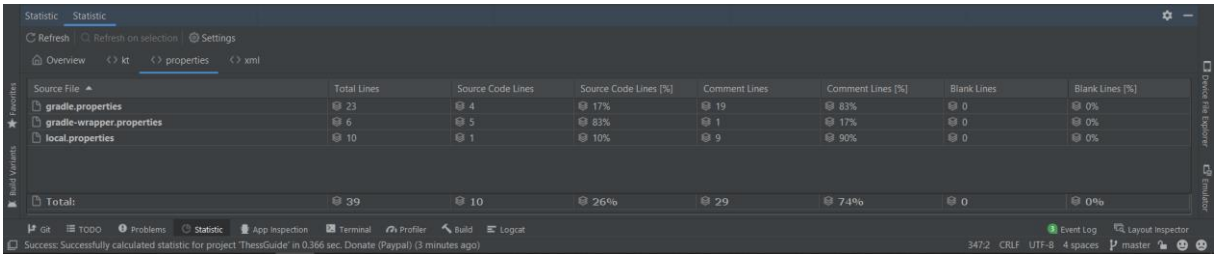
Εικόνα 601: Μέθοδος onResume()

4.11 Επίλογος

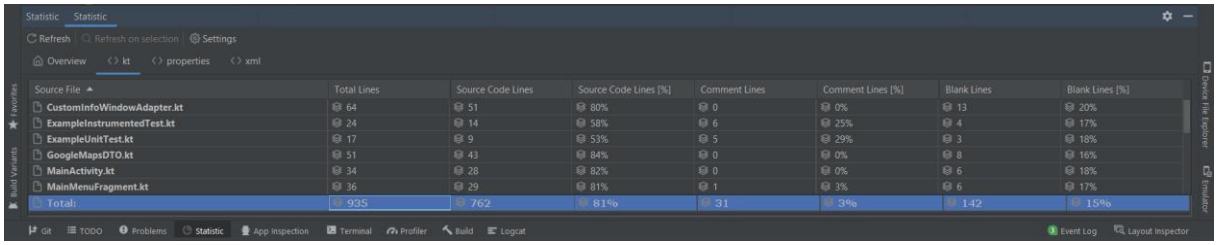
Συνολικά η εφαρμογή αποτελείται από 14 κλάσεις και από 935 γραμμές κώδικα Kotlin. Περισσότερα στατιστικά στοιχεία για το συγκεκριμένο πρότζεκτ φαίνονται αναλυτικά στις εικόνες 62, 63, 64 και 65, τα εικονιζόμενα στοιχεία είναι διαθέσιμα μέσω του plugin Statistics που είναι διαθέσιμο στο Android Studio. Για την δημιουργία των παραπάνω κλάσεων δεν χρησιμοποιήθηκε κάποιο design pattern και αυτό είχε ως αποτέλεσμα να δυσκολευτώ στην εξέλιξη της εφαρμογής. Όπως φάνηκε και παραπάνω ιδιαίτερα σημαντικό ρόλο έπαιξαν τα API της Google για την δημιουργία των βασικών λειτουργιών της εφαρμογής.



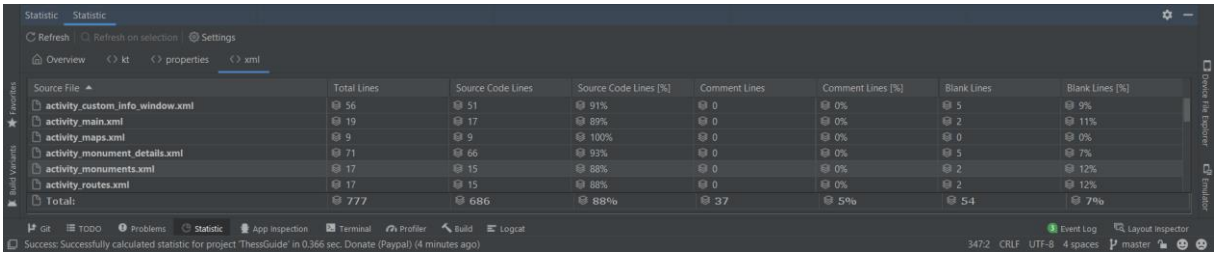
Εικόνα 63: Overview Statistics



Εικόνα 62: Properties Statistics



Εικόνα 64: Kotlin Statistics



Εικόνα 65: XML Statistics

Κεφάλαιο 5ο: Συμπεράσματα και βελτιώσεις

Μετά από σχεδόν ένα χρόνο και δύο ταξίδια που μεσολάβησαν στην διάρκεια της δημιουργίας αυτής της διπλωματικής κατέληξα στο συμπέρασμα ότι στα ταξίδια μας το κινητό μας τηλέφωνο είναι αναπόσπαστο μέρος την καθημερινότητάς μας. Μας είναι απαραίτητα τόσο για τον προσανατολισμό μας σε μία άγνωστη πόλη, όσο και για την οργάνωση του ταξιδιού μας και για την εύρεση πληροφοριών για την πόλη.

Η επιλογή ανάπτυξης της εφαρμογής σε Android έγινε κυρίως λόγω των περιορισμών που τίθενται από την Apple για την ανάπτυξη iOS εφαρμογών. Πιο συγκεκριμένα, για την ανάπτυξη μιας iOS εφαρμογής είναι απαραίτητη η κατοχή ηλεκτρονικού υπολογιστή της Apple (Mac) και η κατοχή iPhone. Όμως τελικά η ανάπτυξη της εφαρμογής σε Android είναι αρκετά απλή για κάποιον που έχει μια εξοικείωση με τον προγραμματισμό και συγκεκριμένα με την ανάπτυξη εφαρμογών για κινητά. Η επιλογή της Kotlin έναντι της Java έκανε την διαδικασία του προγραμματισμού ακόμη πιο ευχάριστη και πιο γρήγορη. Η χρήση της Firebase ήταν η καλύτερη επιλογή βάσης δεδομένων που μπορούσε να γίνει για την υλοποίηση αυτής της εφαρμογής, η σχεδίαση της βάσης ήταν ευκολότερη από τον σχεδιασμό μίας SQL βάσης δεδομένων, ενώ ακόμη μεγαλύτερη ευκολία ήταν ότι η Firebase είναι μια Cloud βάση δεδομένων που σημαίνει ότι υπάρχει πρόσβαση σε αυτήν απομακρυσμένα.

Μελλοντικές βελτιώσεις και αλλαγές θα ήταν η δυνατότητα επιλογής γλώσσας, η διεύρυνση της εφαρμογής και για άλλες πόλεις, το φιλτράρισμα των διαδρομών με βάση την διάρκεια τους ή να περιλαμβάνονται κάποια συγκεκριμένα μνημεία, ενώ θα είχε ενδιαφέρον να δίνεται στον χρήστη η επιλογή να φτιάξει την δικιά του διαδρομή επιλέγοντας ποια μνημεία θέλει να δει και τέλος μια αλλαγή που θα βελτίωνε κατά πολύ την εμπειρία χρήσης της θα ήταν η προσθήκη και ηχητικής περιγραφής του σημείου ενδιαφέροντος.

Η σημαντικότερη και πιο ουσιαστική βελτίωση της εφαρμογής είναι η ανακατασκευή της σύμφωνα με κάποιο design pattern. Τα design patterns είναι ένας τρόπος για να οργανωθεί ο κώδικας σύμφωνα με κάποιους κανόνες, παρέχουν βασικές λύσεις σε προβλήματα που έχει να αντιμετωπίσει ο προγραμματιστής κατά την διάρκεια του σχεδιασμού της εφαρμογής.

Με την χρήση των design patterns ο κώδικας είναι πιο ευανάγνωστος και καθαρός, τόσο για τον ίδιο τον προγραμματιστή που έχει γράψει αυτές τις γραμμές κώδικα, όσο και για κάποιον που θα χρειαστεί να τις διαβάσει και να τις επεξεργαστεί. Ενώ παράλληλα καθορίζεται μια κοινή γλώσσα ανάμεσα στα μέλη μίας ομάδας για το πως θα προχωρήσει ο σχεδιασμός και η δημιουργία της εφαρμογής.

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Rautio Pia-Noora, *TRAVEL AGENCY IN FINLAND AND GENERATIONS Y AND Z*. TURKU UNIVERSITY OF APPLIED SCIENCES, 2020.

[2] «Deloitte Global Millennial Survey 2019» visited 2021. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/global/Documents/About-Deloitte/deloitte-2019-millennial-survey.pdf.

[3] Ketter E. 2019, *Millenial travel: toursim micro-trends of European Generation Y* visited 2021. Available:https://www.emerald.com/insight/content/doi/10.1108/JTF-10-2019-0106/full/pdf?title=millennial-travel-tourism-micro-trends-of-european-generation-y.

[4] Δαμιανός Γαβαλάς, Βλάσης Κασαπάκης και Θωμάς Χατζηδημήτρης, *Κινητές Τεχνολογίες. Κινητός Ιστός – Κινητές Εφαρμογές στην Πλατφόρμα Android – Επαυξημένη Πραγματικότητα*. Αθήνα, 2015.

[5] William Stalling, *Operating Systems Internals and Design Principles. 8th Edition*. Pearson Education, Inc, 2015.

[6] Android Developers Documentation, "Understand the Activity Lifecycle" visited 2021. [Online]. Available: https://developer.android.com/guide/components/activities/activity-lifecycle.

[7] Android Developers Documentation, "Services Overview" visited 2021. [Online]. Available: https://developer.android.com/guide/components/services.

[8] Android Developers Documentation, "Broadcasts Overview" visited 2021. [Online]. Available: https://developer.android.com/guide/components/broadcasts.

[9] Android Developers Documentation, "Content Provider Basics" visited 2021. [Online]. Available: https://developer.android.com/guide/topics/providers/content-provider-basics.

[10] Firebase Documentation, "Cloud Firestore" visited 2021. [Online]. Available: https://firebase.google.com/docs/firestore.

[11] Kotlin vs Java: What's the Difference?, visited 2022 [online] Available: https://www.guru99.com/kotlin-vs-java-difference.html

[12] Kotlin vs Java: Important Differences That You Must Know, visited 2022 [online] Available: https://hackr.io/blog/kotlin-vs-java

[13] Android Developers Documentation, "Sensors Overview" visited 2021. [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_overview.

[14] Android Developers Documentation, "Meet Android Studio" visited 2021. [Online]. Available: https://developer.android.com/studio/intro/?gclid=Cj0KCQjwssyJBhDXARIsAK98ITSVTaylnK7qyf5pZptXBsofMghL7BNVge6_sy3Vg09fCGKZOcxpRrgaAlLuEALw_wcB&gclsrc=aw.ds.

[15] Google Maps Platform, "Google Maps Platform Billing" visited 2021. [Online]. Available: https://developers.google.com/maps/billing/gmp-billing?hl=en_US#maps-product.

[16] Elvis Canziba, *Hands-On UX Design for Developers: Design, prototype, and implement compelling user experiences from scratch.* Birmingham – Mumbai, 2018.

[17] Pinterest, visited 2021.Available: https://gr.pinterest.com/.

[18] W3schools, “Monochromatic Color Schemes” visited 2021. [Online]. Available: https://www.w3schools.com/colors/colors_monochromatic.asp.

[19] Smashing Magazine, “Color Theory for Designers, Part 1: The Meaning of Color” visited 2021. [Online]. Available: https://www.smashingmagazine.com/2010/01/color-theory-for-designers-part-1-the-meaning-of-color/.

[20] UX Collective, “Psychology of shapes in Design: how different shapes can affect people behavior” visited 2021. [Online]. Available: https://uxdesign.cc/psychology-of-shapes-in-design-how-different-shapes-can-affect-people-behavior-13cace04ce1e.

[21] Framer, “Prototyping Tool” visited 2021. [Online]. Available: https://framer.com/projects/.

[22] UI Planet, “Lo-fi vs hi-fi wireframes, and the importance of designing the Flow” visited 2021. [Online]. Available: https://uxplanet.org/lo-fi-vs-hi-fi-wireframes-and-the-importance-of-designing-the-flow-9b283ae62982.

[23] Tejas Vithani and Anand Kumar(2014) ,”Modeling the Mobile Application Development Lifecycle”,Proceedings of the International MultiConference of Engineers and Computer Scientists 2014 Vol I,IMECS 2014, March 12 - 14, Hong Kong

[24] Kaur, Anureet & Kaur, Kulwant. (2015). Suitability of Existing Software Development Life Cycle (SDLC) in Context of Mobile Application Development Life Cycle (MADLC). International Journal of Computer Applications. 116. 1-6. 10.5120/20441-2785.

[25] Android Developers Documentation, App Manifest Overview visited 2021. [online] Available: https://developer.android.com/guide/topics/manifest/manifest-intro

[26] Android Developers Documentation, Manifest.permission visited 2021. [online] Available: https://developer.android.com/reference/android/Manifest.permission#ACCESS_COARSE_LOCATION

[27] Android Project folder Structure, geeksforgeeks visited 2021. [online] Available: https://www.geeksforgeeks.org/android-project-folder-structure/

[28] Android Developers Documentation, Configure your build visited 2021. [online] Available: https://developer.android.com/studio/build

[29] Android Developers Documentation, Add build dependencies, visited 2021. [online] Available: https://developer.android.com/studio/build/dependencies

[30] Platform Architecture, visited 2021 [online] Available: https://developer.android.com/guide/platform