ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή Ψηφιακός Ξεναγός για Συσκευή με Λειτουργικό Android

Βέρα Χριστιάν

ΑΜ: 45350

Εισηγητής: Δρ. Ευάγγελος Κοσμάτος, Καθηγητής

ΑΘΗΝΑ 2019

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή Ψηφιακός Ξεναγός για Συσκευή με Λειτουργικό Android

Προπτυχιακός Φοιτητής, Βέρα Χριστιάν

ΑΜ: 45350

Εισηγητής: Δρ. Ευάγγελος Κοσμάτος, Καθηγητής

Εξεταστική Επιτροπή:

Πρεζεράκος Γεώργιος

Ευάγγελος Κοσμάτος

Γιαννακόπουλος Παναγιώτης

Ημερομηνία εξέτασης: 4/6/2019

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος **Βέρα Χριστιάν**, του **Εντμόντ**, με αριθμό μητρώου **71345350**, φοιτητής του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών του Πανεπιστημίου Δυτικής Αττικής πριν αναλάβω την εκπόνηση της Πτυχιακής Εργασίας μου, δηλώνω ότι ενημερώθηκα για τα παρακάτω:

«Η Πτυχιακή Εργασία (Π.Ε.) αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο του συγγραφέα, όσο και του Ιδρύματος και θα πρέπει να έχει μοναδικό χαρακτήρα και πρωτότυπο περιεχόμενο.

Απαγορεύεται αυστηρά οποιοδήποτε κομμάτι κειμένου της να εμφανίζεται αυτούσιο ή μεταφρασμένο από κάποια άλλη δημοσιευμένη πηγή. Κάθε τέτοια πράξη αποτελεί προϊόν λογοκλοπής και εγείρει θέμα Ηθικής Τάξης για τα πνευματικά δικαιώματα του άλλου συγγραφέα. Αποκλειστικός υπεύθυνος είναι ο συγγραφέας της Π.Ε., ο οποίος φέρει και την ευθύνη των συνεπειών, ποινικών και άλλων, αυτής της πράξης.

Πέραν των όποιων ποινικών ευθυνών του συγγραφέα σε περίπτωση που το Ίδρυμα του έχει απονείμει Πτυχίο, αυτό ανακαλείται με απόφαση της Συνέλευσης του Τμήματος. Η Συνέλευση του Τμήματος με νέα απόφασης της, μετά από αίτηση του ενδιαφερόμενου, του αναθέτει εκ νέου την εκπόνηση της Π.Ε. με άλλο θέμα και διαφορετικό επιβλέποντα καθηγητή. Η εκπόνηση της εν λόγω Π.Ε. πρέπει να ολοκληρωθεί εντός τουλάχιστον ενός ημερολογιακού 6μήνου από την ημερομηνία ανάθεσης της. Κατά τα λοιπά εφαρμόζονται τα προβλεπόμενα στο άρθρο 18, παρ. 5 του ισχύοντος Εσωτερικού Κανονισμού.»

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή ολοκληρώθηκε μέσα από πολλή προσπάθεια και αρκετές αλλαγές ώστε να φτάσει στην τελική της μορφή. Η εφαρμογή αποτελείται από αρκετά διαφορετικά κομμάτια και ενδιαφέρουσες τεχνολογίες όπως το Android, Spring, REST API που μου δόθηκε η ευκαιρία να εντρυφήσω κατά την εκπόνηση. Αρωγός στην προσπάθεια μου αποτέλεσε ο επιβλέπων Καθηγητής μου Κ. Ευάγγελος Κοσμάτος τον οποίο θα ήθελα να ευχαριστήσω θερμά για την καθοδήγηση και τις καίριες συμβουλές που μου προσέφερε.

Επιπλέον, θα ήθελα να ευχαριστήσω την οικογένεια μου που με στηρίζει καθόλη την διάρκεια των σπουδών μου ώστε να αποκτήσω τα απαραίτητα εφόδια για ένα καλύτερο μέλλον.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία ασχολείται με την ανάπτυξη ενός mobile Ψηφιακού Ξεναγού για το λειτουργικό Android. Η κυριαρχία των smartphones τα τελευταία χρόνια σε συνδυασμό με τα Ελληνικά Τοπία που αποτελούν πόλο έλξης κάθε χρονιά, καθιστά ευκαιρία για δημιουργία εφαρμογής με έντονο τουριστικό προσανατολισμό. Ο χρήστης έχει την ευκαιρία με την ευκολία του κινητού του να περιηγηθεί σε διάσημες Αθηναϊκές Περιοχές που παρουσιάζουν Ιστορικό και Τουριστικό ενδιαφέρον, να ενημερωθεί για αυτές αλλά και να ειδοποιείται ότι έφτασε στο επιλεγόμενο σημείο.

ABSTRACT

This dissertation is intended to create a Tour Guide App for Android. This version of app caters to users who visit for the first time Athens, Greece. Project consists of a mobile frontend and a backend which serves with POIs the mobile. The user has the chance to learn and walk through preloaded historical Athenian neighborhoods. Moreover, when he is 15 meters away from a Point of Interest he is notified through a pop-up message.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Android, Software Development

ΛΕΞΕΙΣ ΚΛΕΔΙΑ: εφαρμογή ψηφιακός ξεναγός,λειτουργικό android,ανάπτυξη λογισμικού

KEYWORDS: android mobile development, backend development, tour guide app, java, spring boot, mysql, retrofit, Jackson, REST API

Πίνακας περιεχομένων

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο αναλύεται το αντικείμενο της πτυχιακής, ο σκοπός της, μια μικρή ιστορική αναδρομή στα smartphones (έξυπνα κινητά) με λογισμικό Android και την οργάνωση που θα ακολουθήσει η πτυχιακή.

1.1 Αντικείμενο και Σκοπός Πτυχιακής

Η παρούσα πτυχιακή αποσκοπεί στον σχεδιασμό και υλοποίηση μιας εφαρμογής όπου μπορεί να αποτελέσει τον προσωπικό ψηφιακό ξεναγό του κατόχου. Κυριότερα, ο χρήστης της εφαρμογής μπορεί να επιλέξει κάποιες από τις προεπιλεγμένες διαδρομές που παρουσιάζουν ιστορικό και τουριστικό ενδιαφέρον και με την βοήθεια της, μπορεί να τον βοηθήσει να πλοηγηθεί σε φημισμένες γειτονιές της Αθήνας, να αντλήσει πληροφορίες πατώντας πάνω στα σημεία ενδιαφέροντος (Points of Interest) αλλά και να εμφανιστεί ένα μήνυμα ότι ο χρήστης έφτασε στο σημείο ενδιαφέροντος.

1.2 Ιστορική Αναδρομή

Αναμφίβολα η τεχνολογία και ιδίως οι έξυπνες συσκευές έχουν ξεκινήσει μια τεράστια επέλαση στην καθημερινότητα και δείχνουν ότι έχουν έρθει για να μείνουν. Μέχρι και το 2012 καταγράφονται πωλήσεις τάξεως 1 δις των smartphones σε παγκόσμια κλίμακα. Νούμερο που συνεχώς αυξάνεται μέχρι και σήμερα. Ας γνωρίσουμε λίγο πιο συγκεκριμένα τα smartphones: είναι μια κατηγορία κινητών τηλεφώνων και συσκευών υπολογιστών πολλαπλών χρήσεων. Διακρίνονται από τα κινητά τηλέφωνα με ισχυρότερες δυνατότητες υλικού(hardware) και τα εκτεταμένα λειτουργικά συστήματα κινητής τηλεφωνίας, τα οποία διευκολύνουν το ευρύτερο λογισμικό, το διαδίκτυο (συμπεριλαμβανομένης της περιήγησης στο διαδίκτυο μέσω κινητής ευρυζωνικότητας) και λειτουργίες πολυμέσων (συμπεριλαμβανομένων μουσικής, βίντεο, φωτογραφικών μηχανών και παιχνιδιών). Δεν λείπουν φυσικά οι βασικές λειτουργίες τηλεφώνου, όπως φωνητικές κλήσεις και μηνύματα κειμένου. Τα smartphones περιλαμβάνουν συνήθως διάφορους αισθητήρες που μπορούν να αξιοποιηθούν από το λογισμικό τους, όπως ένα μαγνητόμετρο (magnometers), αισθητήρες εγγύτητας (proximity sensors), βαρόμετρο(barometer), γυροσκόπιο(gyroscope) και επιταχυνσιόμετρο (accelerometer) και υποστηρίζουν πρωτόκολλα ασύρματης επικοινωνίας όπως Bluetooth, Wi-Fi και δορυφορική πλοήγηση(GPS).

Ιδιαίτερα δημοφιλές Λειτουργικό Σύστημα για smartphones αποτελεί το Android.

Συγκεκριμένα, είναι ένα λειτουργικό σύστημα που αναπτύχθηκε από την Google. Βασίζεται σε μια τροποποιημένη έκδοση του πυρήνα του Linux και άλλων λογισμικών ανοιχτού κώδικα και έχει σχεδιαστεί κυρίως για φορητές συσκευές αφής, όπως smartphones και tablet. Οι παραλλαγές του Android χρησιμοποιούνται επίσης σε κονσόλες παιχνιδιών, ψηφιακές φωτογραφικές μηχανές, υπολογιστές και άλλα ηλεκτρονικά.

Αρχικά αναπτύχθηκε από την Android Inc., την οποία αγόρασε η Google το 2005, παρουσιάστηκε το 2007,ενώ η πρώτη εμπορική συσκευή που φορούσε Android διατέθηκε τον Σεπτέμβριο του 2008. Το λειτουργικό σύστημα έχει περάσει από πολλές μεγάλες κυκλοφορίες, με την τρέχουσα έκδοση να είναι 9 "Pie" , που κυκλοφόρησε τον Αύγουστο του 2018. Ο βασικός πηγαίος κώδικας του Android είναι γνωστός ως Android Open Source Project (AOSP) και έχει πρωτίστως άδεια χρήσης Apache (Apache License).

Εικόνα 1.Λογότυπο Λογισμικού Android

1.3 Οργάνωση Πτυχιακής

Στα κεφάλαια που ακολουθούν θα αναλυθούν εκτενέστερα εκτός από τις υπάρχουσες λειτουργίες που προσφέρει το Android οι επιπλέον τεχνολογίες που χρησιμοποιήθηκαν, πως ενσωματώθηκαν στην εφαρμογή, οι ανάγκες αλλά και τα προβλήματα που προέκυψαν κατά την δημιουργία ώστε τελικά να δημιουργηθεί.

ΚΕΦΑΛΑΙΟ 2

ΠΕΡΙΓΡΑΦΗ

Σε αυτό το κεφάλαιο θα γίνει μια περιγραφή του προβλήματος που υπάρχει και πως το αντιμετωπίζει η εφαρμογή αλλά και μια σύντομη ματιά στην αρχιτεκτονική του συνολικού συστήματος μαζί με κάποιες σημαντικές τεχνολογίες που χρησιμοποιήθηκαν.

2.1 Περιγραφή προβλήματος

Η Ελλάδα και συγκεκριμένα η Αθήνα αποτελεί ένα δημοφιλές τουριστικό προορισμό λόγω γεωγραφικής τοποθέτησης αλλά και της ιδιαίτερης ιστορίας που την διακατέχει. Σε συνδυασμό με το γεγονός ότι δεν υπάρχουν αρκετές εφαρμογές στο επίσημο κατάστημα του Android-Play Store αλλά και ιστοσελίδες που να αναδεικνύουν την πόλη της Αθήνας, δημιουργήθηκε η ιδέα για υλοποίηση μια εφαρμογής που βασίζεται στην ιδέα περί ξενάγησης στην πόλη της Αθήνας αλλά και σε οπουδήποτε μέρος με τουριστικό ενδιαφέρον.

2.2 Σύντομη Αρχιτεκτονική Συστήματος

Η εφαρμογή αποτελείται από δύο κύρια κομμάτια το *frontend* που στην περίπτωση μας είναι η mobile εφαρμογή που αλληλεπιδρά ο χρήστης και το *backend* όπου αποτελείται από ένα Web Service ώστε να τροφοδοτεί την εφαρμογή με πληροφορίες όπως οι γεωγραφικές συντεταγμένες των σημείων ενδιαφέροντος, όνομα σημείων και ένα κείμενο με πληροφορίες για τα διάφορα σημεία. Συγκεκριμένα έχουμε:

- **Frontend**: Η εφαρμογή είναι γραμμένη σε Java και τα layouts σε XML.Επιπλέον γίνεται χρήση του Google Maps API για την εμφάνιση των Markers και της πλοήγησης του χρήστη. Πατώντας την επιθυμητή διαδρομή στέλνεται ένα HTTP request στο API ώστε να πάρει πίσω τις πληροφορίες της εκάστοτε διαδρομής. Ο Server στέλνει πίσω τις πληροφορίες σε JSON πάνω από HTTP όπου με την βοήθεια του Retrofit(HTTP client) και του parser Jackson γίνεται αποκωδικοποίηση των JSON πληροφοριών. Στο κεντρικό παράθυρο που εμφανίζονται τα σημεία ενδιαφέροντος πάνω στον χάρτη του Google Maps,μπορούμε να πατήσουμε τα διάφορα Points of Interests (POIs) για πληροφορίες. Επιπλέον καθώς φτάνουμε κοντά σε κάποιο POI εμφανίζεται ένα μήνυμα ότι φτάσαμε στον προορισμό μας.

- **Backend**: Το REST API που εξυπηρετεί την mobile εφαρμογή είναι και αυτό γραμμένο σε Java σε συνδυασμό με το framework Spring Boot, JPA όπου διευκολύνει την ανάπτυξη του API. Για Βάση Δεδομένων έχει χρησιμοποιηθεί η MySQL όπου έχουμε αποθηκεύσει και οργανώσει τα δεδομένα μας.

ΚΕΦΑΛΑΙΟ 3

ΣΧΕΔΙΑΣΜΟΣ

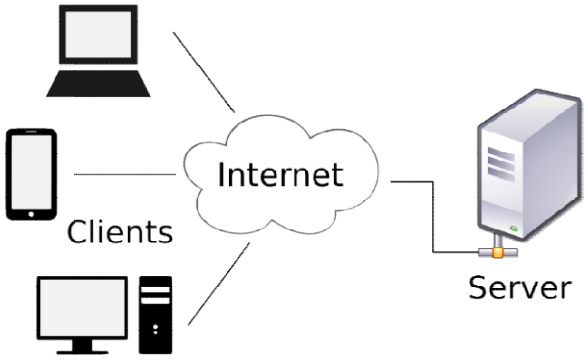
Σε αυτό το κεφάλαιο θα μιλήσουμε αναλυτικά για την αρχιτεκτονική του συστήματος, τις τεχνολογίες που το συνοδεύουν μαζί με τα κατάλληλα διαγράμματα.

3.1 Αρχιτεκτονική Client-Server

Το μοντέλο Client-Server(Πελάτη-Διακομιστή) είναι μια κατανεμημένη δομή εφαρμογής που χωρίζει εργασίες(tasks) ή φόρτο εργασίας(workload) μεταξύ των παρόχων ενός πόρου ή υπηρεσίας, που ονομάζεται Servers(διακομιστές) και οι αιτούντες υπηρεσία, που καλούνται Clients(πελάτες).

Ειδικότερα, ένα σύστημα client-server είναι ένα σύστημα στο οποίο το δίκτυο ενώνει διάφορους υπολογιστικούς πόρους, ώστε οι clients (ή αλλιώς front end) να μπορούν να ζητούν υπηρεσίες από έναν server (ή αλλιώς back end), ο οποίος προσφέρει πληροφορίες ή επιπρόσθετη υπολογιστική ισχύ. Με άλλα λόγια, στο client-server μοντέλο, ο client θέτει μια αίτηση και ο server επιστρέφει μια ανταπόκριση ή κάνει μια σειρά από ενέργειες.

Ο server μπορεί να ενεργοποιείται άμεσα για την αίτηση αυτή ή να προσθέτει την αίτηση σε μια ουρά. Η άμεση ενεργοποίηση για την αίτηση μπορεί, για παράδειγμα, να σημαίνει ότι ο server υπολογίζει έναν αριθμό και τον επιστρέφει αμέσως στον client. Η τοποθέτηση της αίτησης σε μια ουρά μπορεί να σημαίνει ότι η αίτηση πρέπει να τεθεί σε αναμονή για να εξυπηρετηθεί. Ο server τοποθετεί την αίτηση σε μια ουρά μαζί με αιτήσεις εκτυπώσεων και από άλλους clients. Μετά επεξεργάζεται την αίτηση με βάση την σειρά προτεραιότητας, η οποία , σε αυτή την περίπτωση, καθορίζεται από τη σειρά με την οποία ο server παρέλαβε την απαίτηση.



Εικόνα 2.Client-Server Model

Αναλυτικά,ο **client** είναι ο αιτών των υπηρεσιών. Ο client δεν μπορεί παρά να είναι ένας υπολογιστής(σήμερα είτε και κινητό,tablet). Οι υπηρεσίες που ζητούνται από τον client μπορεί να υπάρχουν στους ίδιους σταθμούς εργασίας ή σε απομακρυσμένους σταθμούς εργασίας που συνδέονται μεταξύ τους μέσω ενός δικτύου. Ο client ξεκινάει πάντα την επικοινωνία.

Τα συστατικά του client είναι πολύ απλά. Μια client μηχανή πρέπει να μπορεί να κάνει τα ακόλουθα:

- Να τρέχει το λογισμικό των γραφικών διεπαφών χρηστών (GUIs).
- Να δημιουργεί τις αιτήσεις για πληροφορίες και να τις στέλνει στον server.
- Να αποθηκεύει τις επιστρεφόμενες πληροφορίες.

Ο server απαντάει στις αιτήσεις που γίνονται από τους clients. Ένας client μπορεί να ενεργεί ως server εάν λαμβάνει και επεξεργάζεται αιτήσεις όπως ακριβώς και τις στέλνει (για παράδειγμα, ένας σταθμός εργασίας που χρησιμοποιείται και ως server εκτυπώσεων από άλλους). Οι server δεν ξεκινάνε τις επικοινωνίες -περιμένουν τις αιτήσεις των clients.

Επιστρέφοντας στο παράδειγμα του server εκτυπώσεων ενός δικτύου, ο client ζητάει από τον server να εκτυπώσει ένα κείμενο σε έναν συγκεκριμένο εκτυπωτή και ο server προσθέτει την εκτύπωση σε μια ουρά και ενημερώνει τον client όταν το κείμενο εκτυπωθεί επιτυχημένα. Η διαδικασία του client μπορεί να ανήκει φυσικά στον ίδιο σταθμό εργασίας με την διαδικασία του server. Στο παράδειγμα εδώ, μια εντολή εκτύπωσης μπορεί να εκδίδεται στον server του σταθμού εργασίας του δικτύου, χρησιμοποιώντας την διαδικασία του server εκτυπώσεωνσε αυτόν τον σταθμό εργασίας.

Τα συστατικά του server είναι πολύ απλά. Μια server μηχανή πρέπει να μπορεί να κάνει τα ακόλουθα :

- Να αποθηκεύει, να ανακτά και να προστατεύει πληροφορίες.
- Να επιθεωρεί τις αιτήσεις των clients.
- Να δημιουργεί εφαρμογές διαχείρισης πληροφοριών, όπως δημιουργία αντιγράφων, ασφάλεια κτλ.
- Να διαχειρίζεται πληροφορίες.

3.2 Client-Server Επικοινωνία

Γενικά, ένα Service(υπηρεσία) είναι μια αφαιρετική έννοια που αναφέρεται στους πόρους του υπολογιστή και ένας client δεν χρειάζεται να ασχολείται με τον τρόπο με τον οποίο ο server εκτελεί την εκπλήρωση του αιτήματος και την παράδοση της απάντησης. Ο client πρέπει μόνο να κατανοήσει την απόκριση με βάση το γνωστό πρωτόκολλο εφαρμογής, δηλαδή το περιεχόμενο και τη μορφοποίηση των δεδομένων για την ζητούμενη υπηρεσία.

Οι clients και οι servers ανταλλάσσουν μηνύματα σε ένα πρότυπο μηνυμάτων αίτησης-απόκρισης. Ο πελάτης στέλνει ένα αίτημα και ο διακομιστής επιστρέφει μια απάντηση. Αυτή η ανταλλαγή μηνυμάτων αποτελεί παράδειγμα επικοινωνίας μεταξύ των διαδικασιών.

Για να επικοινωνούν, οι υπολογιστές πρέπει να έχουν μια κοινή γλώσσα και πρέπει να ακολουθούν κανόνες έτσι ώστε τόσο ο client όσο και ο server να γνωρίζουν τι να περιμένουν.

Η γλώσσα και οι κανόνες επικοινωνίας ορίζονται σε ένα πρωτόκολλο επικοινωνιών. Όλα τα πρωτόκολλα client-server λειτουργούν στο application layer(επίπεδο εφαρμογής).

Το application layer ορίζει τα βασικά πρότυπα του διαλόγου. Για να επισημοποιήσει ακόμη περισσότερο την ανταλλαγή δεδομένων, ο server μπορεί να ορίζει ένα application programming Interface (διεπαφή προγραμματισμού εφαρμογών) ή αλλιώς API.Το API είναι ένα επίπεδο αφαίρεσης για την πρόσβαση σε μια υπηρεσία. Περιορίζοντας την επικοινωνία σε μια συγκεκριμένη μορφή περιεχομένου, διευκολύνει το parsing(μετατροπή). Η αφαιρετικότητα αυτή, διευκολύνει την ανταλλαγή δεδομένων μεταξύ των πλατφορμών.

3.3 Web Service

Ως Web Service ορίζουμε την υπηρεσία που επιτρέπει σε μια ηλεκτρονική συσκευή να μιλά μέσω διαδικτύου με μία άλλη.

Μια Web Τεχνολογία, όπως το HTTP που σχεδιάστηκε αρχικά για την επικοινωνία από άνθρωπο σε υπολογιστή, χρησιμοποιείται για επικοινωνία από μηχανή σε μηχανή, ειδικότερα για μεταφορά μορφών αρχείων που μπορούν να διαβαστούν από τη μηχανή, όπως XML και JSON.

Στην πράξη, ένα Web Service παρέχει συνήθως μια αντικειμενοστραφή διεπαφή βασισμένη στον ιστό(object-oriented web-based interface) σ’ένα Database Server, που χρησιμοποιείται για παράδειγμα από έναν άλλο web server ή από μια εφαρμογή για κινητά, που παρέχει μια διεπαφή χρήστη στον τελικό χρήστη.

Πολλοί οργανισμοί που παρέχουν δεδομένα σε μορφοποιημένες σελίδες HTML θα παρέχουν επίσης ότι τα δεδομένα στον server τους ως XML ή JSON, συχνά μέσω ενός web service για να επιτρέψουν την εξαγωγή πληροφοριών.

- **REST (Representational State Transfer)**: είναι ένα αρχιτεκτονικό στυλ λογισμικού που ορίζει ένα σύνολο περιορισμών που πρέπει να χρησιμοποιηθούν για τη δημιουργία Web Service. Τα Web Services που συμμορφώνονται με το αρχιτεκτονικό στυλ REST, ονομάζονται RESTful Web Services (RWS), παρέχουν δια-λειτουργικότητα μεταξύ συστημάτων υπολογιστών στο Διαδίκτυο. Τα RESTful Web Services επιτρέπουν στα αιτούμενα συστήματα να έχουν πρόσβαση και να χειρίζονται κειμενικές αναπαραστάσεις των πόρων του Διαδικτύου χρησιμοποιώντας ένα ομοιόμορφο και προκαθορισμένο σύνολο λειτουργιών.

Σε ένα RWS, τα αιτήματα που υποβάλλονται στο URI (Uniform Resource Identifiers) ενός πόρου θα πάρουν μια απάντηση με ένα payload σε format HTML, XML, JSON ή σε κάποια άλλη μορφή. Η απόκριση μπορεί να επιβεβαιώσει ότι έχουν γίνει ορισμένες αλλαγές στον αποθηκευμένο πόρο και η απάντηση μπορεί να παρέχει συνδέσεις υπερκειμένου (hypertext links) με άλλους σχετικούς πόρους ή συλλογές πόρων. Όταν χρησιμοποιείται το HTTP πρωτόκολλο, όπως συμβαίνει συνήθως, οι διαθέσιμες λειτουργίες (μέθοδοι HTTP) είναι GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS και TRACE.

- **Σχέση μεταξύ URI και HTTP μεθόδων**

Τα Web Services που υιοθετούν την αρχιτεκτονική REST ονομάζονται αλλιώς και RESTful APIs. Συγκεκριμένα ένα HTTP RESTful API ορίζεται με:

- ένα βασικό URI,όπως π.χ http://api.example.com/collection/
- τυπικές μεθόδους HTTP (π.χ. GET, POST, PUT, PATCH και DELETE).
- Ένα media type που ορίζει στοιχεία δεδομένων μεταβατικής κατάστασης (π.χ. Atom, microformats, application / vnd.collection + json κ.λ.π). Η τρέχουσα αναπαράσταση αναφέρει στον client πώς να συνθέτει αιτήματα για μεταβάσεις σε όλες τις επόμενες διαθέσιμες καταστάσεις της εφαρμογής.

Οι παρακάτω πίνακες δείχνει πως οι HTTP μέθοδοι χρησιμοποιούνται σε ένα RESTful API.

URI	GET	POST	PUT	DELETE
Collection Resource όπως `https://api.example.com/`**`collection`**`/`	Ανάκτηση του resource collection μέσω response body.	Δημιουργία ενός μέλους που ανήκει στο resource collection βάση του request μας.	Αντικατάσταση όλων των αναπαραστάσεων των μελών του resource collection βάση του request μας.	Διαγραφή όλων των αναπαραστάσεων των μελών του resource collection.

URI	GET	POST	PUT	DELETE
Member Resource όπως `https://api.example.com/collection/` **item1**	Ανάκτηση μοναδικού μέλους item1 μέσω response body.	Δημιουργία ενος μέλους που ανήκει στο resource collection βάση του request μας.	Αντικατάσταση όλων των αναπαραστάσεων του member resource(item 1) βάση του request μας.	Διαγραφή όλων των αναπαραστάσεων των μελών του member resource(εδώ item1).

- **JSON (Javascript Object Notation):** είναι μια μορφή αρχείου ανοικτού προτύπου (open-standard), δημιουργία του Douglas Crockford, που χρησιμοποιεί κείμενο αναγνώσιμο από άνθρωπο για τη μετάδοση data objects(αντικειμένων δεδομένων) που αποτελούνται από ζεύγη attributes-value(χαρακτηριστικών-τιμών) και array data types (ή Serializable μεταβλητές). Πρόκειται για μια πολύ κοινή μορφή δεδομένων που χρησιμοποιείται για την ασύγχρονη επικοινωνία προγράμματος περιήγησης-server, συμπεριλαμβανομένης της αντικατάστασης της XML σε ορισμένα συστήματα τύπου AJAX.

  Το JSON είναι μια μορφή δεδομένων ανεξάρτητη από τη γλώσσα. Προέρχεται από το JavaScript, αλλά από το 2017 πολλές γλώσσες προγραμματισμού περιλαμβάνουν κώδικα για την παραγωγή και ανάλυση δεδομένων τύπου JSON. Ο επίσημος τύπος μέσου διαδικτύου για το JSON είναι application / json. Τα ονόματα αρχείων JSON χρησιμοποιούν την επέκταση .json.

- **Βασικά JSON Data Types**

NUMBER	STRING	BOOLEAN	ARRAY	OBJECT	NULL

- NUMBER: προσημασμένος δεκαδικός αριθμός που μπορεί να περιέχει ένα κλασματικό τμήμα και μπορεί να χρησιμοποιεί εκθετική σημείωση e, αλλά δεν μπορεί να περιλαμβάνει μη αριθμούς όπως NaN. Η μορφή δεν κάνει διάκριση μεταξύ ακέραιου και δεκαδικού απλής ακρίβειας(float). Η JavaScript χρησιμοποιεί μια μορφή κινητής υποδιαστολής διπλής ακρίβειας για όλες τις αριθμητικές τιμές του, αλλά άλλες γλώσσες που εφαρμόζουν JSON ενδέχεται να κωδικοποιούν διαφορετικά τους αριθμούς.
- STRING: μια ακολουθία μηδενικών ή περισσότερων χαρακτήρων Unicode. Τα Strings(συμβολοσειρές) οριοθετούνται με διπλά εισαγωγικά(“ ”) και υποστηρίζουν backslash(/) χαρακτήρες διαφυγής.
- BOOLEAN: true ή false τιμές
- ARRAY: μια λίστα με μηδενικές ή περισσότερες τιμές, καθεμιά από τα οποία μπορεί να είναι οποιουδήποτε τύπου. Οι πίνακες χρησιμοποιούν τετραγωνικές αγκύλες ([ ]) και τα στοιχεία διαχωρίζονται με κόμμα.
- OBJECT: μια συλλογή ζευγών με names (ονόματα) και values (τιμές) όπου τα ονόματα (ονομάζονται επίσης κλειδιά) είναι strings. Δεδομένου ότι τα objects προορίζονται να αντιπροσωπεύουν associative arrays,συνιστάται, αν και δεν απαιτείται, ότι κάθε key είναι μοναδικό μέσα σε ένα object. Τα objects οριοθετούνται με κυματιστές αγκύλες({ }) και χρησιμοποιούν κόμματα για να

διαχωρίζουμε κάθε ζεύγος, ενώ μέσα σε κάθε ζεύγος ο χαρακτήρας άνω-κάτω τελεία (:) χωρίζει το key ή το name από το value του.

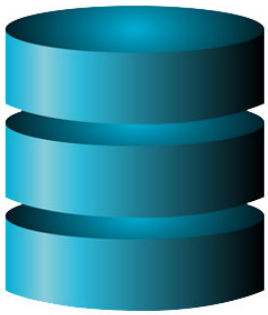
- NULL: κενή τιμή

π.χ JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

3.4 Βάση Δεδομένων (DATABASE)

Βάση Δεδομένων(Β.Δ) είναι ένα σύνολο δεδομένων που σχετίζονται μεταξύ τους και μπορούν να προσπελαστούν από ένα Η/Υ. Ουσιαστικά, η Β.Δ αναπαριστά ένα μέρος του πραγματικού κόσμου. Η διαδικασία ανάπτυξης μιας Β.Δ περιλαμβάνει ένα αριθμό σταδίων. Συγκεκριμένα, μια Β.Δ σχεδιάζεται, δημιουργείται, τροφοδοτείται με δεδομένα και χρησιμοποιείται.



Εικόνα 3. Σύμβολο Β.Δ

Για την διευκόλυνση ανάπτυξης και διαχείριση μιας Β.Δ αναπτύχθηκε εξειδικευμένο λογισμικό, το οποίο ονομάζεται Σύστημα Διαχείρισης Βάσεων Δεδομένων(Database Management System-DBMS).

Με τον όρο διαχείριση υπονοούμε την δημιουργία, εισαγωγή, διαγραφή, τροποποίηση και αναζήτηση των δεδομένων.

Οι χρήστες μιας Β.Δ μπορούν να έχουν πρόσβαση στη Β.Δ μόνο μέσω των υπηρεσιών που προσφέρει ένα DBMS και εφόσον έχουν το δικαίωμα για αυτό.

Τα σύγχρονα DBMS προσφέρουν πρόσβαση στη Β.Δ με διάφορους τρόπους επικοινωνίας. Οι πιο διαδεδομένοι είναι:

- Με χρήση γραφικού περιβάλλοντος(GUI)
- Με χρήση μιας γλώσσας προγραμματισμού που είναι εξειδικευμένη για την διαχείριση των δεδομένων, με πιο δημοφιλή την SQL.
- Με χρήση μιας εξειδικευμένης εφαρμογής που έχει αναπτυχθεί με μια γενική γλώσσα προγραμματισμού όπως C/C++ ή Java.

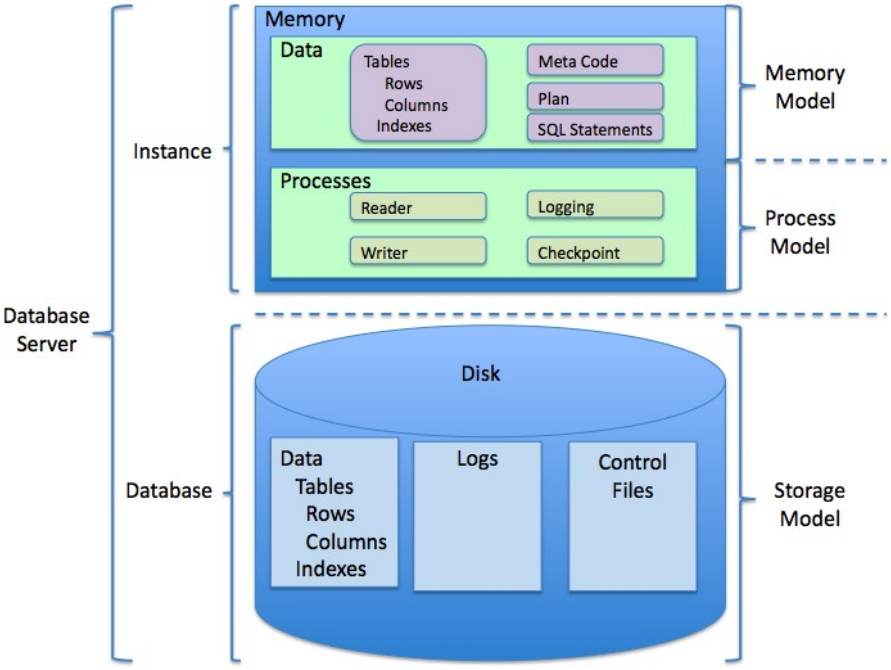
Οι επιστήμονες υπολογιστών επιπλέον ταξινομούν τα Συστήματα Διαχείρισης Βάσεων Δεδομένων σύμφωνα με τα μοντέλα βάσης δεδομένων που υποστηρίζουν. Οι σχεσιακές βάσεις δεδομένων (Relational Database Management System-RDBMS) κυριάρχησαν στη δεκαετία του '80. Η μοντελοποίηση Δεδομένων γίνεται με γραμμές και στήλες σε μια σειρά πινάκων και η συντριπτική πλειονότητα χρησιμοποιούν SQL για τη γραφή και την αναζήτηση δεδομένων. Στη δεκαετία του 2000, οι μη σχεσιακές βάσεις

δεδομένων έγιναν δημοφιλής, και αναφέρονται ως NoSQL επειδή χρησιμοποιούν διαφορετικές γλώσσες ερωτημάτων (query languages).

3.4.1 ΣΧΕΣΙΑΚΟ ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΗΣΗΣ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ (RDBMS)

Ένα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων (RDBMS) είναι ένα σύστημα διαχείρισης βάσεων δεδομένων (DBMS) βασισμένο στο σχεσιακό μοντέλο δεδομένων. Οι περισσότερες σημερινές Β.Δ βασίζονται σε αυτό το μοντέλο.
Τα RDBMS αποτελούν μια κοινή επιλογή για την αποθήκευση πληροφοριών σε Β.Δ που χρησιμοποιούνται για τις οικονομικές καταγραφές, την παραγωγή και τις υλικοτεχνικές πληροφορίες, τα δεδομένα προσωπικού χαρακτήρα και άλλες εφαρμογές από τη δεκαετία του '80.

Οι σχεσιακές βάσεις δεδομένων έχουν συχνά αντικαταστήσει τις ιεραρχικές βάσεις δεδομένων και τις βάσεις δεδομένων δικτύου επειδή ήταν ευκολότερες στην εφαρμογή και τη διαχείριση.



Εικόνα 4.Επισκόπηση ενός RDBMS

3.4.2 SQL (Structured Query Language)

Είναι μια domain-specific γλώσσα που χρησιμοποιείται στον προγραμματισμό και έχει σχεδιαστεί για τη διαχείριση δεδομένων που διατηρούνται σε ένα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων (RDBMS) ή για επεξεργασία ροής(stream processing) σε ένα σύστημα διαχείρισης ροής σχεσιακών δεδομένων (RDSMS). Είναι ιδιαίτερα χρήσιμο για τον χειρισμό δομημένων δεδομένων όπου υπάρχουν σχέσεις μεταξύ διαφορετικών οντοτήτων / μεταβλητών των δεδομένων. Η SQL προσφέρει δύο βασικά πλεονεκτήματα έναντι των παλαιότερων API ανάγνωσης / εγγραφής όπως ISAM ή VSAM. Πρώτον, εισήγαγε την έννοια της πρόσβασης σε πολλά αρχεία με μία μόνο εντολή. και δεύτερον, εξαλείφει την ανάγκη καθορισμού τρόπων επίτευξης μιας εγγραφής, π.χ. με ή χωρίς index(δείκτη).

Η SQL αναπτύχθηκε αρχικά στην IBM από τον Donald D. Chamberlin και τον Raymond F. Boyce αφού έμαθε σχετικά με το σχεσιακό μοντέλο από τον Ted Codd στις αρχές της δεκαετίας του 1970. Αυτή η έκδοση, που αρχικά ονομάστηκε SEQUEL (Structured English Query Language), σχεδιάστηκε για να διαχειριστεί και να ανακτήσει τα δεδομένα που ήταν αποθηκευμένα στο αρχικό σύστημα διαχείρισης της βάσης δεδομένων της IBM, System R, το οποίο είχε αναπτύξει μια ομάδα στο IBM San Jose Research Laboratory κατά τη δεκαετία του 1970.

Έγινε πρότυπο του Αμερικανικού Εθνικού Ινστιτούτου Προτύπων (ANSI) το 1986 και του Διεθνούς Οργανισμού Τυποποίησης (ISO) το 1987. Έκτοτε, το πρότυπο έχει αναθεωρηθεί ώστε να περιλαμβάνει ένα ευρύτερο σύνολο χαρακτηριστικών.

```
1  SELECT * FROM tour_app.poi;
2  SELECT id,lat,lng,Tour FROM tour_app.poi WHERE Tour=1;
3  SELECT poi.TourID,poi.lat,poi.lng,tour.tour_name,poi.tour_description
4  FROM tour_app.poi
5  INNER JOIN tour_app.tour ON poi.Tour=tour.id
6  WHERE TourID=1;
7  SELECT poi.TourID,poi.lat,poi.lng,tour.tour_name,poi.tour_description
8  FROM tour_app.poi
9  INNER JOIN tour_app.tour ON poi.Tour=tour.id
10 WHERE TourID=2;
```

Εικόνα 5. Παράδειγμα SQL

3.4.3 ORM (Object-Relational Mapping)

Είναι μια τεχνική προγραμματισμού για τη μετατροπή δεδομένων μεταξύ ασύμβατων συστημάτων τύπου χρησιμοποιώντας αντικειμενοστραφείς γλώσσες προγραμματισμού. Αυτό δημιουργεί, μια εικονική αντικειμενοστραφής Β.Δ (virtual object database) που μπορεί να χρησιμοποιηθεί από τη γλώσσα προγραμματισμού.

Υπάρχουν διαθέσιμα δωρεάν και εμπορικά πακέτα που εκτελούν αντικειμενοστραφή χαρτογράφηση, παρόλο που ορισμένοι προγραμματιστές επιλέγουν να κατασκευάσουν τα δικά τους εργαλεία ORM.

Στον αντικειμενοστραφή προγραμματισμό, τα καθήκοντα διαχείρισης δεδομένων ενεργούν σε αντικείμενα που είναι σχεδόν πάντα μη-κλιμακωτά(non-scalar). Για παράδειγμα, μια καταχώρηση βιβλίου διευθύνσεων που αντιπροσωπεύει ένα μόνο άτομο μαζί με μηδέν ή περισσότερους αριθμούς τηλεφώνου και μηδέν ή περισσότερες διευθύνσεις.

Αυτό θα μπορούσε να διαμορφωθεί σε μια εφαρμογή αντικειμενοστραφής από ένα class(αντικείμενο) "Person" με ιδιότητες / πεδία για να κρατήσει κάθε στοιχείο δεδομένων που περιλαμβάνει: το όνομα του ατόμου, μια λίστα αριθμών τηλεφώνου και μια λίστα διευθύνσεων. Η λίστα των αριθμών τηλεφώνου θα περιέχει τα ίδια "αντικείμενα Phone, Number" και ούτω καθεξής.

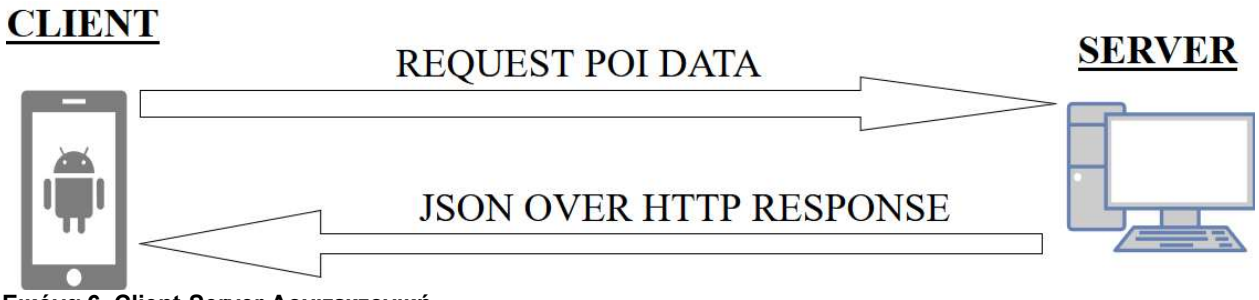
Η καταχώρηση του βιβλίου διευθύνσεων αντιμετωπίζεται ως ένα μοναδικό αντικείμενο από τη γλώσσα προγραμματισμού (μπορεί να γίνει αναφορά από μία μόνο μεταβλητή που περιέχει δείκτη στο αντικείμενο, για παράδειγμα).

Έπειτα ακολουθώντας τις objected oriented τεχνικές, διάφορες μέθοδοι μπορούν να συσχετιστούν με το αντικείμενο, όπως μια μέθοδο για την επιστροφή του προτιμώμενου αριθμού τηλεφώνου, της διεύθυνσης κατοικίας κλπ.

3.5 Συνολική Επισκόπηση Αρχιτεκτονικής

Η αρχιτεκτονική που χρησιμοποιήθηκε όπως προαναφέρθηκε είναι Client-Server.

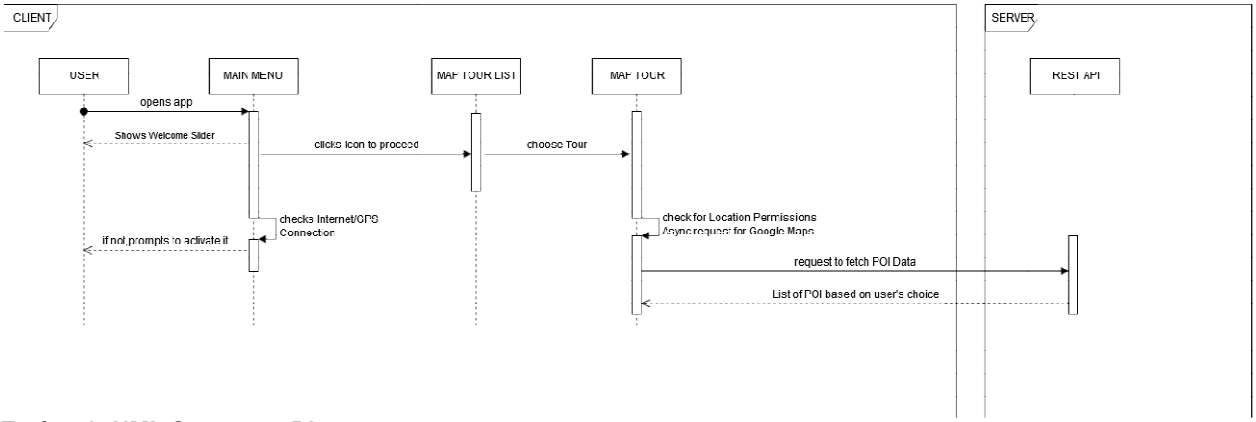
Στην προκειμένη περίπτωση τον ρόλο του client εδώ έχει το κινητό όπου είναι εγκατεστημένη η εφαρμογή και τον ρόλο του Server έχει ο τοπικός Η/Υ. Πάνω σε αυτόν είναι η Database και το API για την εξυπηρέτηση του Client.



Εικόνα 6. Client-Server Αρχιτεκτονική

Το κινητό κάνει request τα σημεία ενδιαφέροντος(Points of Interests) και λαμβάνει πίσω τα δεδομένα από την Database σε μορφή JSON.Ας δούμε αναλυτικά πως γίνετε στά σχετικά διαγράμματα.

Εικόνα 7. Επισκόπηση Τεχνολογιών



Εικόνα 8. UML Sequence Diagram

Οι εικόνες 7,8 δίνουν μια συνολική επισκόπηση το τι συμβαίνει.

Παρακάτω θα δούμε αναλυτικά συμβαίνει στο backend κομμάτι και τι στο frontend.

ΚΕΦΑΛΑΙΟ 4

ΤΕΧΝΟΛΟΓΙΕΣ

Σε αυτό το κεφάλαιο θα γίνει μια σύντομη περιγραφή με τις τεχνολογίες που χρησιμοποιήθηκαν και πώς ενσωματώνονται στην εφαρμογή.

4.1 IDE (Integrated Development Environment)

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) είναι μια εφαρμογή λογισμικού που παρέχει ολοκληρωμένες δυνατότητες στους προγραμματιστές για την ανάπτυξη λογισμικού.

Ένα IDE αποτελείται συνήθως από έναν source code editor (επεξεργαστής πηγαίου κώδικα), automation tools (εργαλεία αυτοματισμού κατασκευής) και έναν debugger (εργαλείο εντοπισμού σφαλμάτων).

Οι περισσότεροι από τους σύγχρονους IDE έχουν έξυπνη ολοκλήρωση κώδικα. Ορισμένοι IDE, όπως το NetBeans και το Eclipse, περιέχουν έναν compiler (μεταγλωττιστή), έναν interpreter(διερμηνέα) ή και τα δύο.

4.1.1 Android Studio

Το Android Studio είναι το επίσημο ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για το λειτουργικό σύστημα Android της Google, το οποίο βασίζεται στο λογισμικό *IntelliJ IDEA* της JetBrains και έχει σχεδιαστεί ειδικά για ανάπτυξη Android.

Είναι διαθέσιμο για λήψη σε λειτουργικά συστήματα που βασίζονται σε Windows, macOS και Linux.

Αποτελεί αντικαταστάτη του Eclipse Android Development Tools (ADT) ως το πρωταρχικό IDE για την ανάπτυξη εφαρμογών Android.

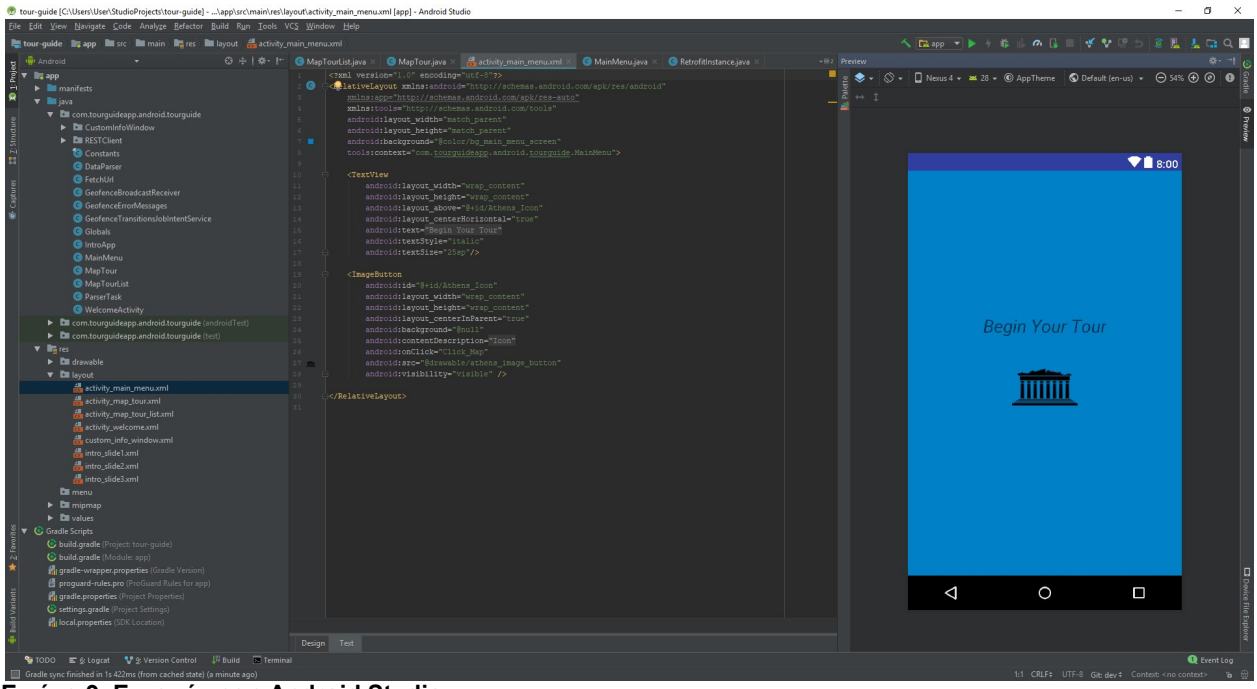
Οι ακόλουθες λειτουργίες παρέχονται στην τρέχουσα σταθερή έκδοση:

- Build Automation με το Gradle.
- Refactoring ειδικά για Android οδηγώντας σε γρήγορες επιδιορθώσεις κώδικα.
- Εργαλεία Lint για να παρακολούθηση της απόδοσης, τη χρηστικότητα, τη συμβατότητα έκδοσης και άλλα προβλήματα που μπορεί να προκύψουν.
- Ενσωμάτωση ProGuard για δυνατότητα υπογραφής εφαρμογών(app-signing).
- Templates για προκαθορισμένα Android designs και components.

- Layout Editor που επιτρέπει στους χρήστες να φτιάχνουν με ευκολία UI, δυνατότητα προεπισκόπησης των layouts σε διάφορες διαστάσεις οθόνης.
- Υποστήριξη για την ανάπτυξη εφαρμογών Android Wear.
- Ενσωματωμένη υποστήριξη(Built-In) για την πλατφόρμα Google Cloud, επιτρέποντας την ενσωμάτωση των εφαρμογών Cloud Messages και το Google App Engine.
- Εικονική συσκευή Android (Emulator) για την εκτέλεση και τον εντοπισμό σφαλμάτων σε εφαρμογές στο στούντιο Android.

Το Android Studio υποστηρίζει όλες τις ίδιες γλώσσες προγραμματισμού IntelliJ (και CLion) π.χ. Java, C ++ και άλλες με επεκτάσεις, όπως η Go.

Από την έκδοση Android Studio 3.0 ή νεότερες εκδόσεις, υποστηρίζουν και την γλώσσα kotlin.



Εικόνα 9. Επισκόπηση Android Studio

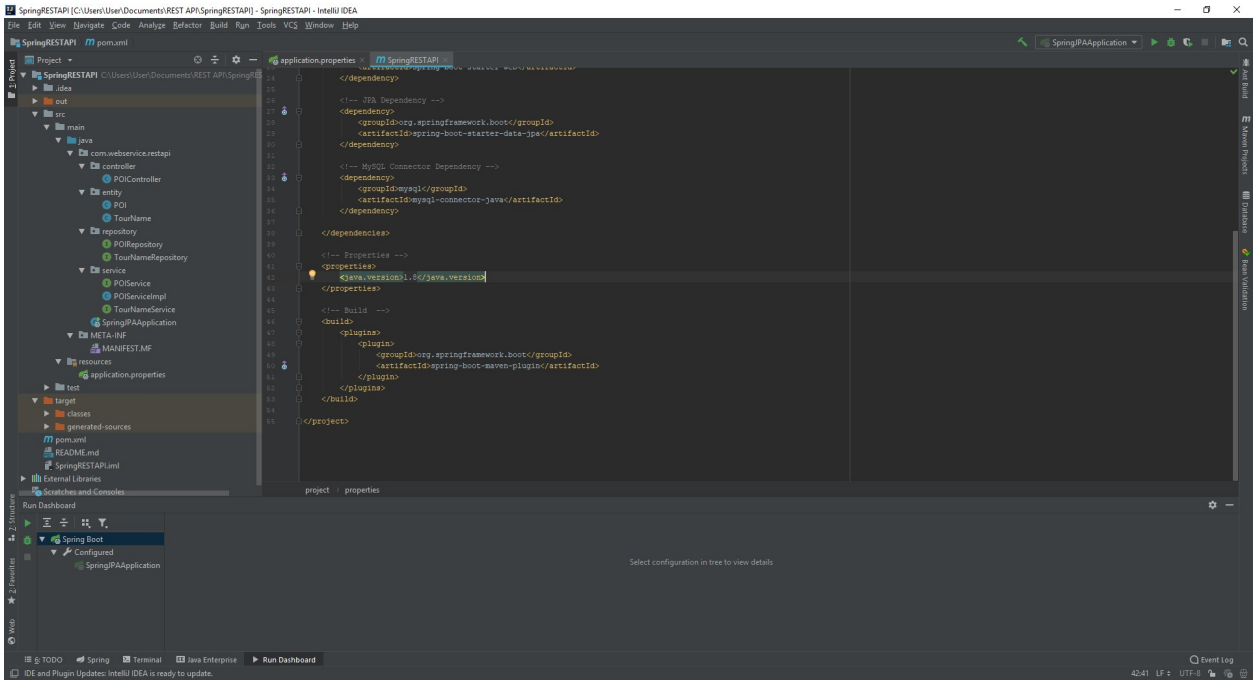
4.1.2 Intellij IDEA

Το IntelliJ IDEA είναι ένα Java IDE για την ανάπτυξη λογισμικού υπολογιστών. Αναπτύσσεται από το JetBrains (παλαιότερα γνωστή ως IntelliJ) και είναι διαθέσιμο ως ελεύθερη έκδοση Community edition κάτω από την άδεια Apache 2 Licensed αλλά και σε ιδιωτική εμπορική έκδοση- Ultimate Edition. Και οι δύο μπορούν να χρησιμοποιηθούν για εμπορική ανάπτυξη εφαρμογών.

Οι ακόλουθες λειτουργίες που παρέχονται είναι:

- Coding Assistance όπως συμπλήρωση κώδικα, refactoring και συμβουλές για γρήγορη επιδιόρθωση κώδικα.
- Built-In Tools όπως το Gradle, Maven. Επιπλέον υποστηρίζει και VCS(Version Control Systems) όπως το Git,SVN.
- Plugin Support για επιπλέον λειτουργίες του IDE.

Υποστηριζόμενες γλώσσες είναι φυσικά η Java,Kotlin,Groovy και άλλες μέσω plugins.



Εικόνα 10. Επισκόπηση Intellij IDEA

4.2 Frameworks

4.2.1 Spring Framework

Εικόνα 11. Λογότυπο Spring

Το Spring Framework είναι ένα application framework και IoC (Inversion of Control) container για την πλατφόρμα Java.

Τα βασικά χαρακτηριστικά του framework μπορούν να χρησιμοποιηθούν από οποιαδήποτε εφαρμογή Java, αλλά υπάρχουν extensions (επεκτάσεις) για την κατασκευή Web Applications πάνω από την πλατφόρμα Java EE (Enterprise Edition).

Αν και το framework δεν επιβάλλει συγκεκριμένο μοντέλο προγραμματισμού, έχει γίνει δημοφιλές στην κοινότητα Java ως προσθήκη ή ακόμα και αντικατάσταση του μοντέλου Enterprise JavaBeans (EJB). Το Spring είναι open-source.

Το Spring περιέχει αρκετά Modules οπότε παρακάτω αναλύουμε τα βασικότερα:

- Spring Core Container: Βασικό Module που προσφέρει τα Spring Containers (Bean Factory και Application Context).
- Υποστήριξη Aspect-oriented Programming.
- Authentication & Authorization: παραμετροποιημένα security processes που υποστηρίζουν διάφορα standards, πρωτόκολλα και εργαλεία μέσω του Spring Security.
- Data Access: υποστήριξη RDBMS μέσω της Java κάνοντας χρήση του JDBC (Java Database Connectivity), ORM τεχνικών αλλά και NoSQL DB.
- Υποστήριξη Inversion of Control (IoC) container: παραμετροποίηση των applications components και lifecycle management των Java Objects μέσω Dependency Injection.
- Model-View-Controller (MVC) Μοντέλο: HTTP και servlet framework για παραμετροποίηση Web Apps και RESTful Web Services.
- Remote Access framework: παραμετροποίηση RPC μοντέλου για marshalling των Java Objects μέσω δικτύου χρησιμοποιώντας JRMI (Java Remote Method Invocation),CORBA (Common Object Request), και SOAP (Simple Object Access Protocol).
- Transaction Management: ενοποίηση διαφόρων transaction APIs και καθοδήγηση transactions για Java Objects
- Testing: υποστήριξη κλάσεων για unit και integration tests

4.2.2 Spring Boot

Το Spring Boot είναι μια “ετοιμοπαράδοτη” λύση και ανήκει στη λογική του convention-over-configuration του Spring για τη δημιουργία αυτόνομων εφαρμογών που βασίζονται στο Spring και οι οποίες χωρίς πολλά configuration να τρέξουν εύκολα.

Είναι σεταρισμένο με το “opinionated view” του Spring όπου γίνεται χρήση του καλύτερου configuration από την ίδια την πλατφόρμα περιλαμβάνοντας τις αναγκαίες third-party βιβλιοθήκες ώστε να τρέξει η εφαρμογή χωρίς άλλη καθυστέρηση.

Χαρακτηριστικά που περιλαμβάνει:

- Δημιουργία stand-alone(αυτόνομων) Spring εφαρμογών.
- Ενσωματωμένος Application Server Tomcat ή Jetty (δεν χρειάζεται να παραχθούν αρχεία WAR).
- Παρέχετε αρχείο POM(Project Object Model) για απλοποίηση στη διαμόρφωση του Maven.
- Αυτόματη ρύθμιση του Spring όποτε είναι δυνατόν.
- Παρέχονται λειτουργίες για τσεκάρισμα της εφαρμογής, όπως metrics,health checks και εξωτερικό configuration.

4.2.3 JPA

Το Java Persistence API (JPA) είναι ένα API που περιγράφει τη διαχείριση σχεσιακών δεδομένων σε εφαρμογές που χρησιμοποιούν πλατφόρμα Java είτε Standard Edition είτε Enterprise Edition.

Το “persistence“ καλύπτει τρεις τομείς:

- Το ίδιο το API, που ορίζεται στο πακέτο javax.persistence
- Η γλώσσα JPQL(Java Persistence Query Language)
- Αντικειμενικά / σχεσιακά metadata

4.2.4 Spring Data JPA

Μια αφαιρετική υλοποίηση του repository,που αποτελεί βασικό δομικό στοιχείο του Domain-Driven Design που βασίζεται στο Spring framework. Υποστηρίζει όλες τις διαθέσιμες υλοποιήσεις JPA και υποστηρίζει CRUD λειτουργίες καθώς και έτοιμα queries προς την DB.

4.3 Database

4.3.1 MySQL

Εικόνα 12. Λογότυπο MySQL

MySQL είναι ένα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων ανοικτού κώδικα (RDBMS). Το όνομά είναι ένας συνδυασμός του "My", το όνομα της συνιδρυτής της κόρης του Michael Widenius, και της "SQL", της συντομογραφίας για τη δομημένη γλώσσα ερωτημάτων.

Η MySQL είναι δωρεάν λογισμικό ανοιχτού κώδικα υπό τους όρους της Γενικής Δημόσιας Άδειας GNU και διατίθεται επίσης με διάφορες άδειες ιδιοκτησίας. Η MySQL ανήκε και χρηματοδοτήθηκε από τη σουηδική εταιρεία MySQL AB, η οποία αγοράστηκε από την Sun Microsystems (τώρα Oracle Corporation).

Η MySQL χρησιμοποιείται από πολλά Web Apps που βασίζονται σε Β.Δ, όπως το Drupal, το Joomla, το phpBB και το WordPress. Η MySQL χρησιμοποιείται επίσης από πολλούς δημοφιλείς ιστότοπους, όπως το Google (αν και όχι για αναζητήσεις), Facebook, Twitter, Flickr, και YouTube.

Η MySQL προσφέρεται σε δύο διαφορετικές εκδόσεις: την MySQL Community Server ανοιχτού κώδικα και την ιδιόκτητη Enterprise Server. Η MySQL Enterprise Server διαφοροποιείται από μια σειρά από ιδιόκτητες επεκτάσεις(extensions) οι οποίες εγκαθιστούν ως server plugins,αλλά κατά τα άλλα μοιράζονται το ίδιο σύστημα αρίθμησης εκδόσεων και είναι κατασκευασμένα από την ίδια βάση κώδικα με την community.

Σημαντικές λειτουργίες που διατίθενται στην MySQL 5.6:

- Πρότυπο ANSI SQL 99
- Cross-Platform υποστήριξη
- Stored Procedures
- Triggers
- Cursors
- Information Schema
- Performance Schema για παρακολούθηση απόδοσης των queries
- Transactions με save points στην default InnoDB Storage Engine
- SSL Support
- Query caching
- Sub-SELECTs
- Built-In Replication
- Full-Text Indexing & Searching
- Unicode Support
- Πολλαπλά Storage Engines
- Native Storage engines όπως InnoDB, MyISAM, Merge, CSV, NDB Cluster
- Commit grouping για πολλαπλά transactions

4.3.2 MySQL Workbench

Το *MySQL Workbench* είναι το επίσημο ολοκληρωμένο γραφικό περιβάλλον για την MySQL.

Αναπτύχθηκε από την MySQL AB και επιτρέπει στους χρήστες να διαχειρίζονται γραφικά τις βάσεις δεδομένων MySQL και να σχεδιάζουν οπτικά τις δομές των ΒΔ.

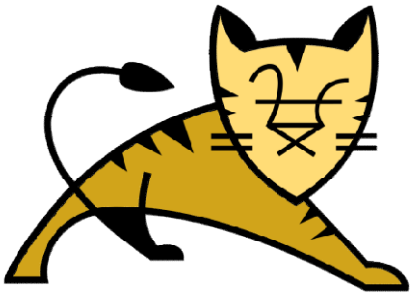
Το MySQL Workbench αντικαθιστά το προηγούμενο πακέτο λογισμικού, MySQL GUI Tools.

Ο MySQL Workbench επιτρέπει στους χρήστες να διαχειρίζονται τη σχεδίαση και το σχεδιασμό βάσεων δεδομένων, την ανάπτυξη SQL (αντικαθιστώντας το MySQL Query Browser) και τη διαχείριση της Β.Δ (αντικατάσταση του MySQL Administrator), παρόμοια με άλλα πακέτα άλλων κατασκευαστών.

Το MySQL Workbench είναι διαθέσιμο σε δύο εκδόσεις, την ελεύθερη και ανοιχτή έκδοση Community Edition και την ιδιόκτητη Standard Edition που επεκτείνει και βελτιώνει το σύνολο χαρακτηριστικών της Community Edition.

4.4 Application Server

4.4.1 Apache Tomcat



Εικόνα 13. Λογότυπο Tomcat Apache

Ο Apache Tomcat, συχνά αναφέρεται ως Tomcat Server, είναι ένα Java Servlet Container ανοικτού κώδικα που αναπτύχθηκε από το Apache Software Foundation (ASF). Ο Tomcat υλοποιεί διάφορες προδιαγραφές της Java EE, όπως το Java Servlet, Java Server Pages (JSP), το Java EL και το Web Socket, και παρέχει έναν Java HTTP Web Server στο οποίο μπορεί να εκτελεστεί κώδικας Java.

Το Tomcat αναπτύσσεται και συντηρείται από μια ανοιχτή κοινότητα προγραμματιστών υπό την αιγίδα του Apache Software Foundation, το οποίο κυκλοφορεί υπό την άδεια Apache License 2.0 και είναι λογισμικό ανοικτού κώδικα.

Ο Tomcat ξεκίνησε ως servlet reference από τον James Duncan Davidson, αρχιτέκτονα λογισμικού της Sun Microsystems. Αργότερα βοήθησε να γίνει το έργο open source και να διαδραματίσει βασικό ρόλο στη δωρεά του από την Sun Microsystems στο Apache Software Foundation.

Χαρακτηριστικά των νεότερων εκδόσεων Tomcat :

Το Tomcat 7.x υλοποιεί τις προδιαγραφές Servlet 3.0 και JSP 2.2 . Απαιτεί Java έκδοση 1.6, παρόλο που οι προηγούμενες εκδόσεις έχουν τρέξει σε Java 1.1 έως 1.5. Οι εκδόσεις 5 έως 6 είδαν βελτιώσεις στο συλλογή garbage collection, JSP parsing, τις επιδόσεις και την επεκτασιμότητα.

Το Tomcat 8.x υλοποιεί τις προδιαγραφές Servlet 3.1 και JSP 2.4 . Το Apache Tomcat 8.5.x προορίζεται να αντικαταστήσει το 8.0.x και περιλαμβάνει νέα χαρακτηριστικά που από τον Tomcat 9.0.x.

Ο TomCat εμπεριέχεται στο Spring Framework με έκδοση 8.3.

4.5 Google Maps

Εικόνα 14. Λογότυπο Google Maps

Είναι μια υπηρεσία χαρτογράφησης που αναπτύχθηκε από την Google. Προσφέρει δορυφορικές εικόνες, αεροφωτογραφία, χάρτες δρόμων, πανοραμική θέα σε δρόμους 360 ° (Street View), συνθήκες κυκλοφορίας σε πραγματικό χρόνο (Google Traffic) και σχεδιασμό διαδρομών για περπάτημα, ποδήλατα, αυτοκίνητα ή τα μέσα μαζικής μεταφοράς.

Τα Google Maps ξεκίνησαν ως desktop App σε C ++ από την Where 2 Technologies. Τον Οκτώβριο του 2004, η εταιρεία εξαγοράστηκε από την Google, η οποία την μετέτρεψε σε Web App. Μετά από πρόσθετες εξαγορές μιας εταιρείας γεωγραφικής απεικόνισης δεδομένων και ενός αναλυτή κυκλοφορίας σε πραγματικό χρόνο, κυκλοφόρησαν τον Φεβρουάριο του 2005. Το frontend της υπηρεσίας χρησιμοποιεί JavaScript, XML και Ajax. Τα Google Maps προσφέρουν ένα API που επιτρέπει στους χάρτες να ενσωματωθούν σε ιστότοπους τρίτων και προσφέρει εντοπισμό για αστικές επιχειρήσεις και άλλους οργανισμούς σε πολλές χώρες σε όλο τον κόσμο.

Τα Google Maps για συσκευές *Android* και *iOS* κυκλοφόρησαν τον Σεπτέμβριο του 2008 και διαθέτουν πλοήγηση GPS μαζί με ειδικά χαρακτηριστικά υποστήριξης στάθμευσης. Τον Αύγουστο του 2013, αναδείχθηκε η πιο δημοφιλής εφαρμογή για smartphones στον κόσμο, με το 54% των παγκόσμιων κατόχων smartphone να το χρησιμοποιούν τουλάχιστον μία φορά.

- **Google Maps API**

Η Google κυκλοφόρησε το API του Google Maps τον Ιούνιο του 2005 για να επιτρέψει στους προγραμματιστές να ενσωματώσουν τους Χάρτες Google στους ιστότοπους τους.

Ήταν μια δωρεάν υπηρεσία που δεν απαιτεί κλειδί API μέχρι τον Ιούνιο του 2018 (οι αλλαγές άρχισαν να ισχύουν στις 16 Ιουλίου), όταν ανακοινώθηκε ότι ένα κλειδί API συνδεδεμένο με έναν λογαριασμό Google Cloud με ενεργοποιημένη χρέωση θα πρέπει να έχει πρόσβαση στο API .

Χρησιμοποιώντας το API της Google, είναι δυνατή η ενσωμάτωση των Google Maps σ' έναν εξωτερικό ιστότοπο ή εφαρμογή, από τον οποίο μπορούμε να παρουσιάσουμε δεδομένα που αντλούμε από το API.

Παρόλο που αρχικά χρησιμοποιήθηκε μόνο ένα API JavaScript, το API Χαρτών επεκτάθηκε για να συμπεριλάβει μια υπηρεσία για την ανάκτηση εικόνων στατικών χαρτών και Web Services για geocoding, οδηγίες για οδήγηση κλπ.

Πάνω από 1.000.000 ιστότοποι χρησιμοποιούν το API Χαρτών Google, καθιστώντας το πιο ισχυρό API για Web App Development.

Το Google Maps API είναι ελεύθερο για εμπορική χρήση, υπό την προϋπόθεση ότι ο ιστότοπος στον οποίο χρησιμοποιείται είναι δημόσιος και δεν χρεώνει για πρόσβαση και δεν παράγει περισσότερες από 25.000 προσβάσεις χάρτη ημερησίως.

Οι ιστότοποι που δεν πληρούν αυτές τις απαιτήσεις μπορούν να αγοράσουν την άδεια Google Maps API για business .

Από τις 21 Ιουνίου 2018, η Google αύξησε τις τιμές του API Χαρτών και απαιτεί billing profile (προφίλ χρέωσης).

ΚΕΦΑΛΑΙΟ 5

ΥΛΟΠΟΙΗΣΗ

Στο κεφάλαιο αυτό θα αναλύσουμε σε βάθος τον κώδικα και τις διαδικασίες για την ενσωμάτωση των απαραίτητων third-party libraries, APIs. Αρχικά θα γίνει αναφορά στο backend κομμάτι και έπειτα στο frontend.

5.1 Backend

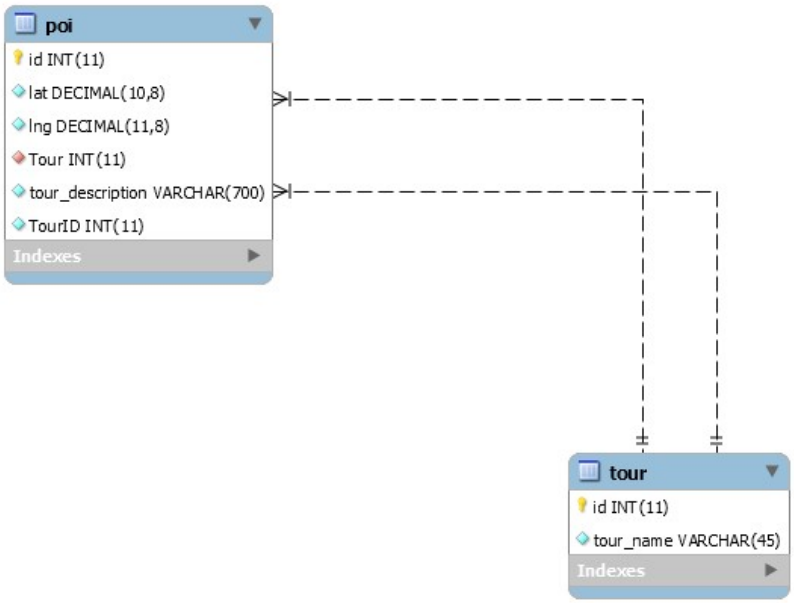
- Βρίσκεται στον τοπικό Η/Υ και συνίσταται από την MySQL Βάση Δεδομένων μας, μαζί με το REST API.

Η Β.Δ περιέχει 2 πίνακες:

i) τον πίνακα POI με τα εξής στοιχεία: id,lat,lng,Tour,tour_description και TourID, όπου περιέχουν τις γεωγραφικές συντεταγμένες (lat,lng),σε ποιο tour ανήκουν (Tour) και μια περιγραφή για το εκάστοτε σημείο ενδιαφέροντος (tour description)

ii) τον πίνακα Tour με τα στοιχεία: id και tour_name, όπου περιέχει τα ονόματα των σημείων ενδιαφέροντος (tour_name).

Η σχέση των 2 πινάκων είναι 1-1.

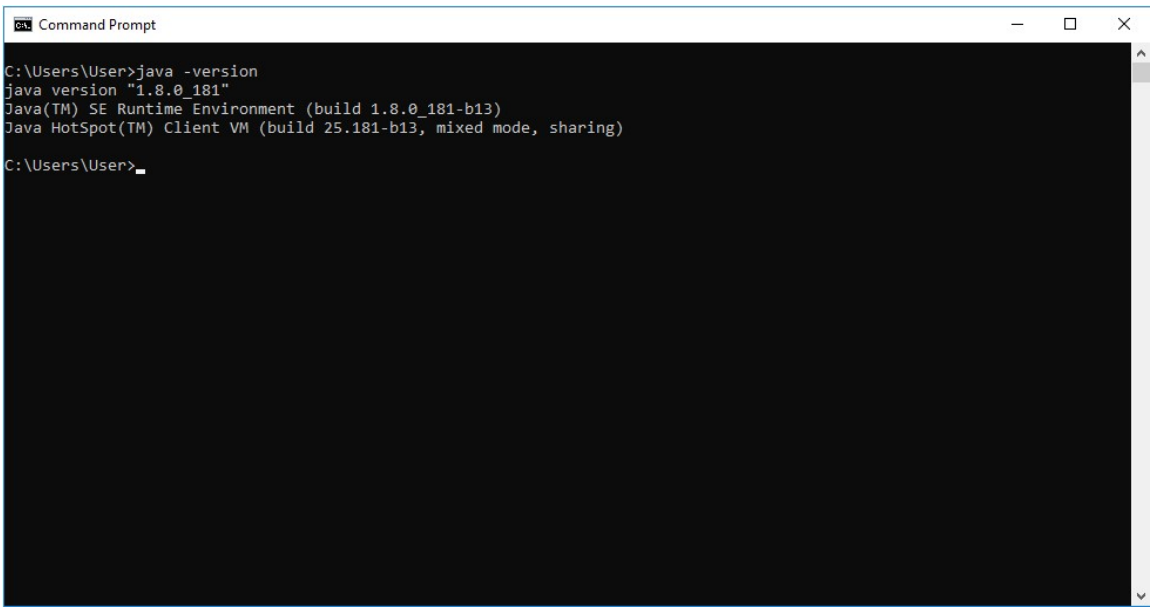


Εικόνα 15. MySQL Schema

Όπως έχει προαναφερθεί το backend έχει γραφτεί σε γλώσσα Java με την βοήθεια του Spring Framework στο Intellij IDEA Ultimate Edition.

Πριν προχωρήσουμε πρέπει να βεβαιωθούμε ότι έχουμε εγκατεστημένη την γλώσσα java στον Η/Υ. Ανοίγουμε το cmd και εκτελούμε την ακόλουθη

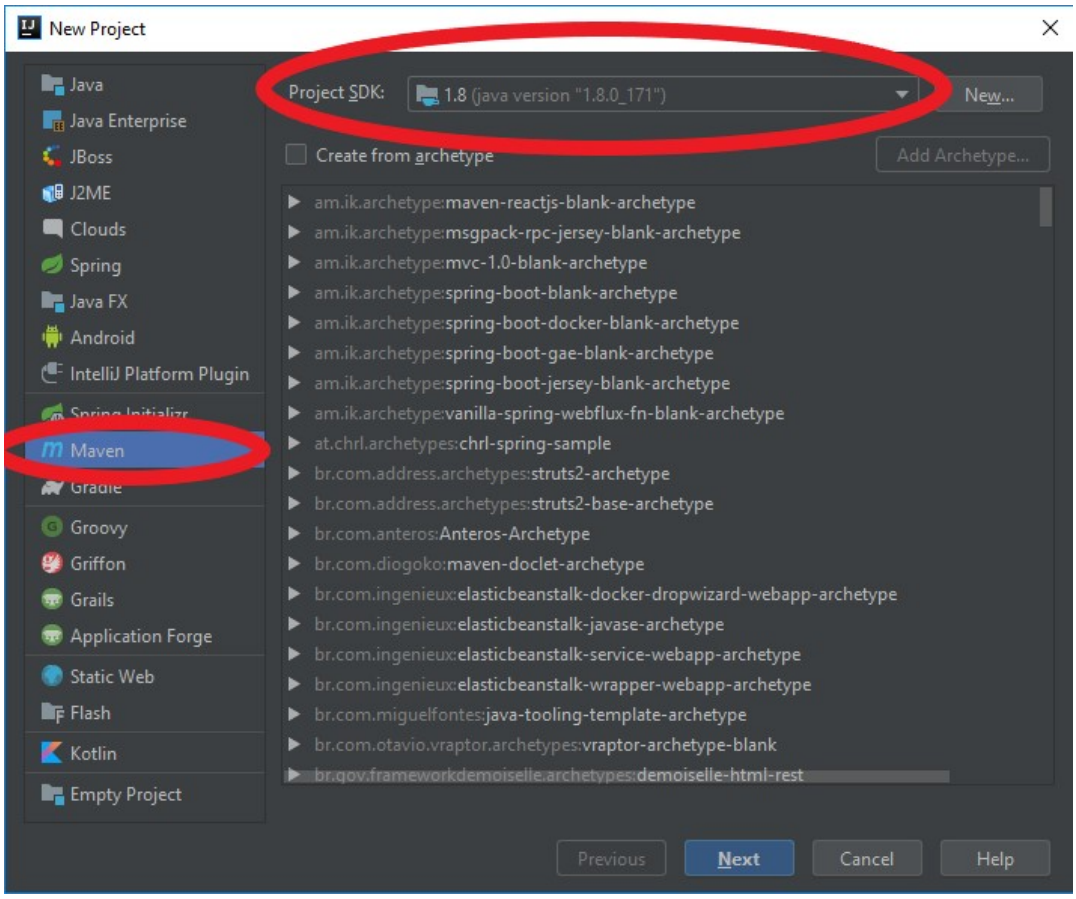
εντολή: java -version



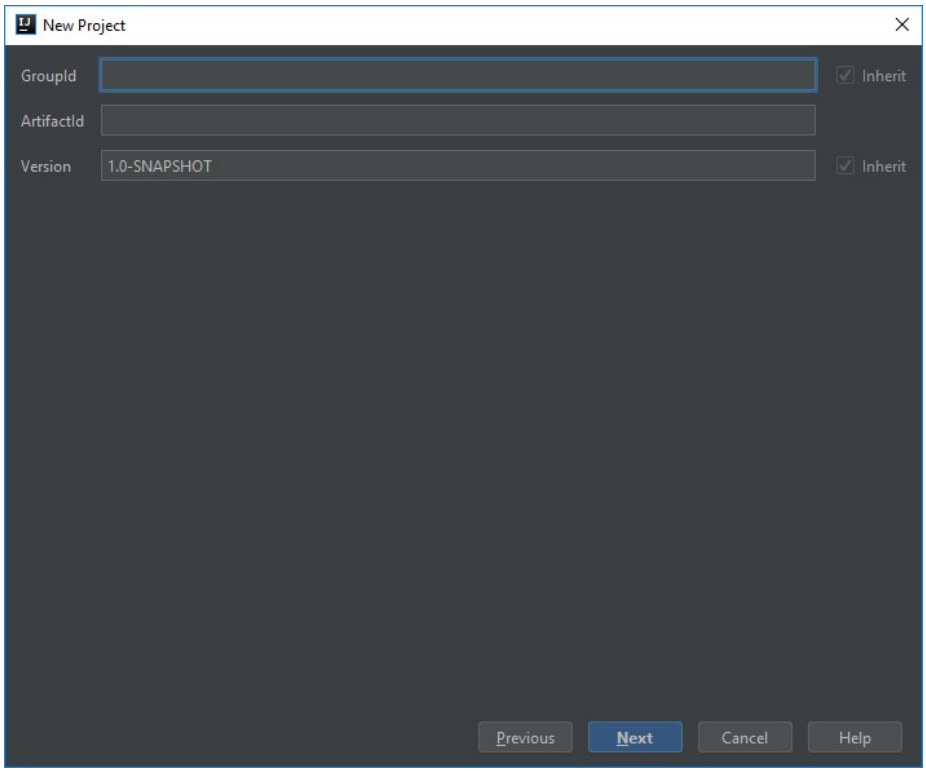
5.1.1 Δημιουργία Project στο IntelliJ IDEA

Αφότου ανοίξουμε το IDE δημιουργούμε ένα καινούργιο Project στο File→ New→Project…

Επιλέγουμε Maven και στο Project SDK επιλέγουμε την έκδοση java μας.



Έπειτα πατάμε Next και εμφανίζεται το παρακάτω παράθυρο.

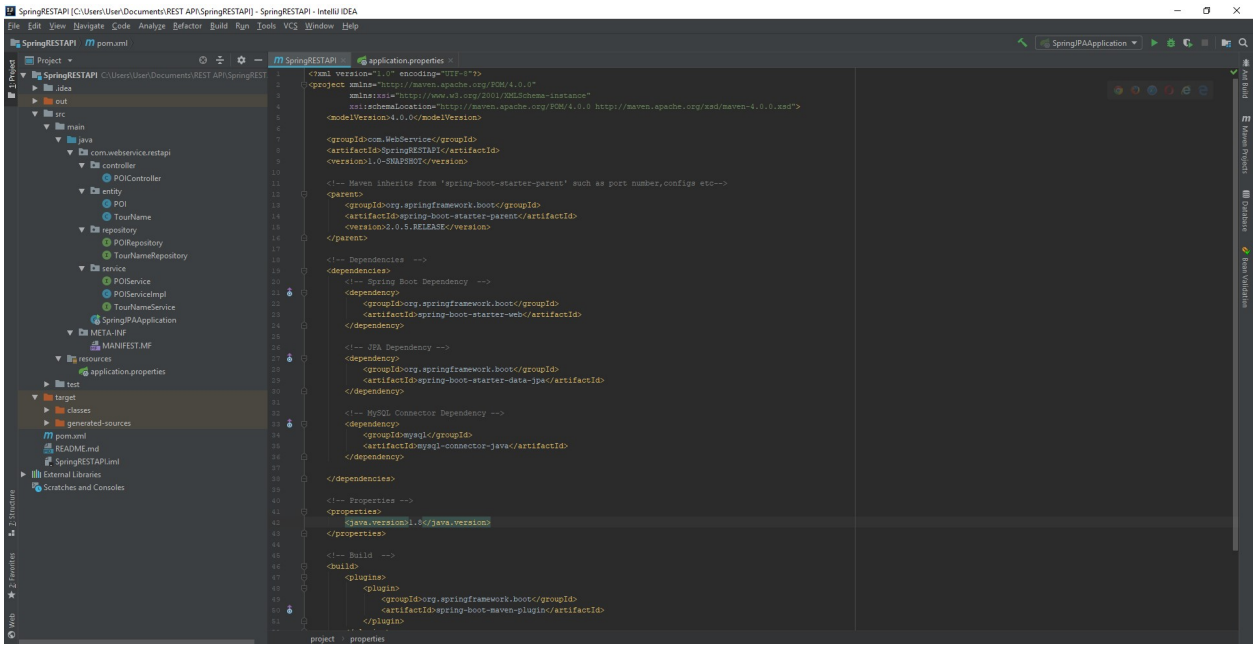


Στο GroupId συμπληρώνουμε με com.WebService

και στο ArtifactID συμπληρώνουμε με SpringRESTAPI.

Πατάμε next και δημιουργούμε το project μας.

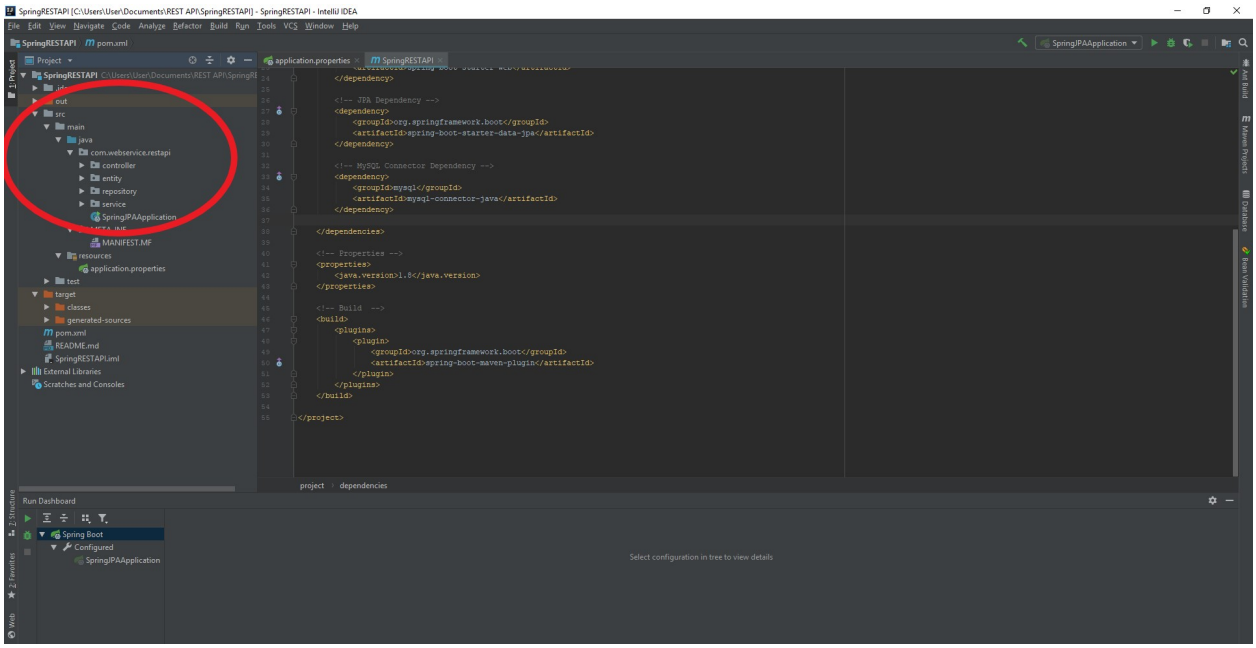
Στην διαδικασία αυτή δημιουργούμε και ένα αρχείο POM.xml όπου αποτελεί οδηγό για το πως είναι δομημένο το project μας άλλα και τα dependencies που θα χρειαστούμε.



Στην παραπάνω εικόνα βλέπουμε το pom.xml με τα generated ονόματα που βάλαμε αλλά και την Java έκδοση που θα τρέξει η εφαρμογή.

Επιπλέον με το tag dependency μπορούμε να ορίσουμε τα επιπλέον dependencies που θα χρειαστούμε όπως Spring Boot, Spring Data JPA και MySQL Connector για να μπορούμε να συνδεθούμε με την Β.Δ

Αφότου συμπληρωθούν όλα πηγαίνουμε στο File→Synchronize(είτε CTRL+ALT+Y) ώστε να κατέβουν τα dependencies που ζητήσαμε.



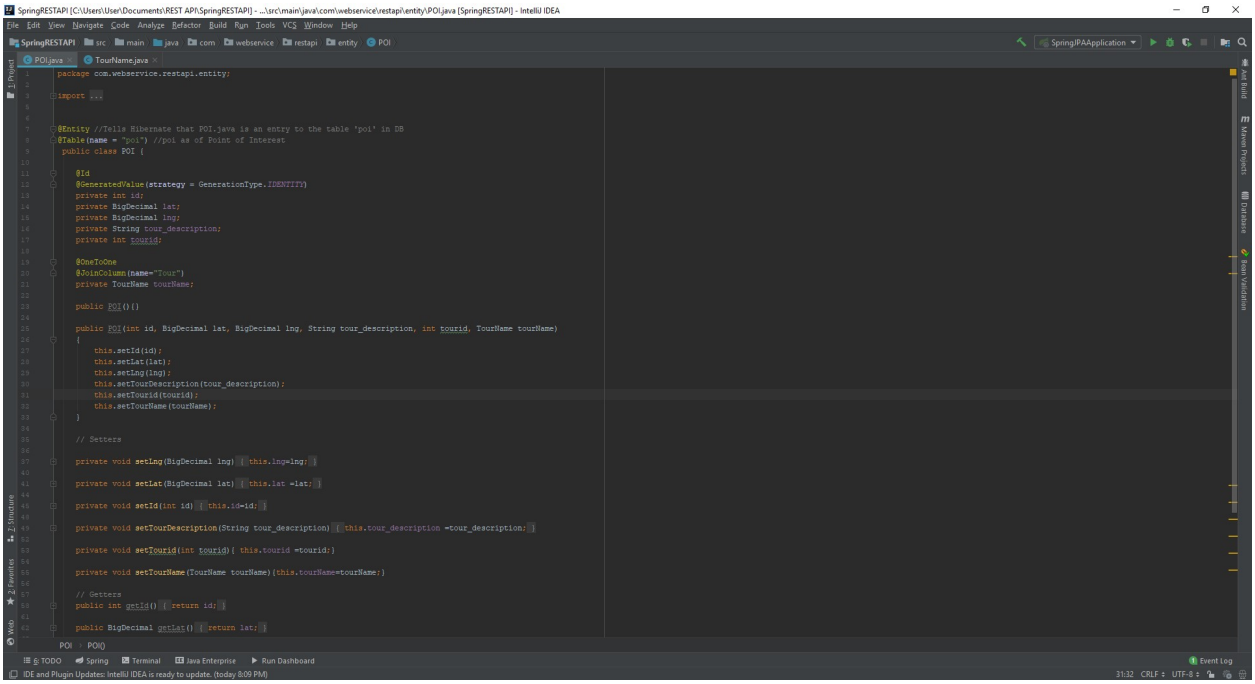
Όπως φαίνεται και στην παραπάνω εικόνα έχουμε οργανώσει τις κλάσεις σε κάποια κύρια packages που αντιπροσωπεύουν κύριες αρμοδιότητες του Spring.

Τα packages είναι τα εξής:

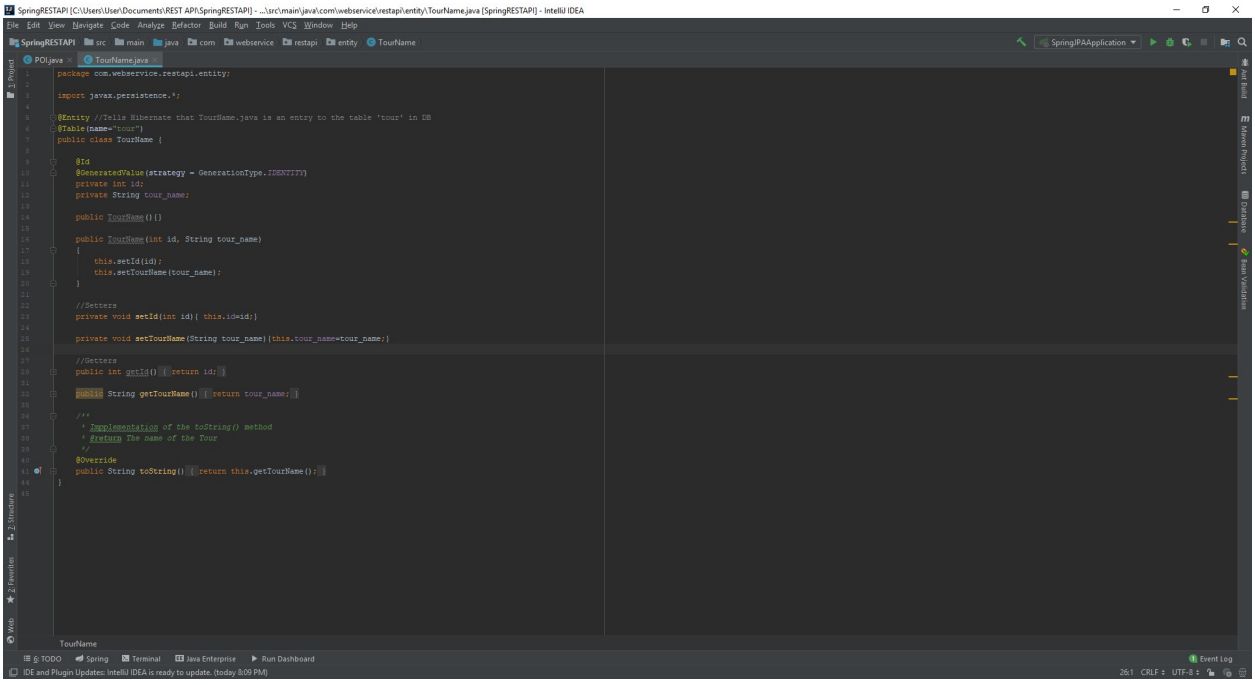
- Controller: περιέχει την κλάση POIController όπου είναι υπεύθυνη για να τις URL κλήσεις που θα κάνει η mobile εφαρμογή.
- Entity: περιέχει τις κλάσεις POI και TourName όπου είναι υπεύθυνη για κάνει τις αντιστοίχηση των πινάκων στην Β.Δ με τις αντικειμενοστραφής κλάσεις μας.
- Repository: περιέχει το Interface POIRepository όπου είναι υπεύθυνο για τα queries στην Β.Δ
- Service: περιέχει το Interface POIService και την κλάση POIServiceImpl για την υλοποίηση των queries στην Β.Δ

5.1.2 POI.java και TourName.java

Αρχικά θα αναλύσουμε τι γίνετε στις κλάσεις POI και TourName.



```java
package com.webservice.restapi.entity;

import ...

@Entity //Tells Hibernate that POI.java is an entry to the table 'poi' in DB
@Table(name = "poi") //poi as of Point of Interest
public class POI {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private BigDecimal lat;
    private BigDecimal lng;
    private String tour_description;
    private int tourid;

    @OneToOne
    @JoinColumn(name="Tour")
    private TourName tourName;

    public POI() {}

    public POI(int id, BigDecimal lat, BigDecimal lng, String tour_description, int tourid, TourName tourName)
    {
        this.setId(id);
        this.setLat(lat);
        this.setLng(lng);
        this.setTourDescription(tour_description);
        this.setTourid(tourid);
        this.setTourName(tourName);
    }

    // Setters

    private void setLng(BigDecimal lng) { this.lng=lng; }

    private void setLat(BigDecimal lat) { this.lat =lat; }

    private void setId(int id) { this.id=id; }

    private void setTourDescription(String tour_description) { this.tour_description =tour_description; }

    private void setTourid(int tourid) { this.tourid =tourid;}

    private void setTourName(TourName tourName){this.tourName=tourName;}

    // Getters
    public int getId() { return id; }

    public BigDecimal getLat() { return lat; }
```



```java
package com.webservice.restapi.entity;

import javax.persistence.*;

@Entity //Tells Hibernate that TourName.java is an entry to the table 'tour' in DB
@Table(name="tour")
public class TourName {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String tour_name;

    public TourName() {}

    public TourName(int id, String tour_name)
    {
        this.setId(id);
        this.setTourName(tour_name);
    }

    //Setters
    private void setId(int id){ this.id=id;}

    private void setTourName(String tour_name){this.tour_name=tour_name;}

    //Getters
    public int getId() { return id; }

    public String getTourName() { return tour_name; }

    /**
     * Implementation of the toString() method
     * @return The name of the Tour
     */
    @Override
    public String toString() { return this.getTourName(); }
}
```

Οι 2 αυτές Κλάσεις χρησιμεύουν ώστε να κάνουμε την “αντιστοίχηση” από τα αντίστοιχα entities που βρίσκονται στην Β.Δ σε java κλάσεις ή POJO(Plain Old Java Objects) classes όπως αλλιώς ονομάζονται.

Το Spring γενικότερα κάνει εκτεταμένη χρήση των annotations(@) που επιτελούν συγκεκριμένες λειτουργίες.

Ας δούμε συγκεκριμένα την POI κλάση.

Βλέπουμε ότι χρησιμοποιούμε τα 2 annotations @Entity και @Table όπου δηλώνουμε ότι πρόκειται για κλάση τύπου entity και το table αναφέρει το όνομα του πίνακα που έχουμε στην Β.Δ που είναι poi.

Έπειτα έχουμε το @Id όπου δηλώνει ότι η μεταβλητή που ακολουθεί μετά είναι το Primary Key μας και το @GeneratedValue με strategy identity όπου αφήνει Β.Δ να αναλάβει το auto-increment του PK.

Τέλος έχουμε το @OneToOne και @JoinColumn που χρησιμεύουν ώστε να δείξουμε οτι οι πίνακες της Β.Δ έχουν σχέση 1-1 και το join θα γίνει βάση της Tour στήλης.

Ακολουθούν ο constructor για λόγους αρχικοποίησης και τα κλασσικά getters/setters για τις μεταβλητές.

Τα ίδια συμβαίνουν για την κλάση TourName με εξαίρεση το join.

Η λογική είναι ότι όσα tables έχουμε στην Β.Δ τόσα entities θα πρέπει να δημιουργήσουμε ώστε να γίνει η αντιστοίχηση.

5.1.3 POIService.java και POIServiceImp.java

```java
/** Interface to declare our common Methods */
public interface POIService {

    List<POI> findAll();
    List<POI> findByTourid(int tourid);
}
```

```java
/** Implementing the Methods from POIService */
@Service // stereotype for service layer,same with component annotation
public class POIServiceImpl implements POIService {

    @Autowired
    private POIRepository poiRepository;

    @Override
    public List<POI> findAll() { return poiRepository.findAll(); }

    @Override
    public List<POI> findByTourid(int tourid) { return poiRepository.findByTourid(tourid); }
}
```

Στις παραπάνω εικόνες βλέπουμε ένα interface το POIService και την κλάση POIServiceIml που κάνει το implement.

Συγκεκριμένα, στο POIService δηλώνουμε τις μεθόδους findAll και findByTourID όπου επιστρέφουν μια λίστα τύπου POI. Η πρώτη επιστρέφει ότι έχουμε στους 2 πίνακες και η 2η μέθοδος επιστρέφει όλα τα tour βάση του ID (π.χ. 1,2,3).

Οι 2 αυτές μέθοδοι αντιπροσωπεύουν τα query που θα κάναμε στην βάση άλλα σε γλώσσα java. Την λειτουργία αυτή αναλαμβάνει το Spring Data JPA αφού αυτόματα αντιστοιχίζει το keyword IS,EQUALS ενός native query(π.χ. where tourid=1?) με το findby που κάνουμε χρήση.

Το POIServiceImpl αναλαμβάνει να υλοποιήσει τις μεθόδους του interface.

Ειδική αναφορά στο @Autowired που ακολουθείτε από το από την μεταβλητή POIRespository καθώς αναλαμβάνει αυτόματα να σκανάρει όλες τις κλάσεις που περιέχουν annotations όπως controller,repository και χωρίς περαιτέρω configuration να δημιουργήσει τα Beans.

Παρακάτω βλέπουμε και το interface POIRespository με annotation @Respository που είναι υπεύθυνο για τα query στην Β.Δ να κληρονομεί από το JpaRespository όπου δηλώνουμε την κλάση POI και data type του PK.

Ακολουθεί η μέθοδος findbyTourid με την μεταβλητή tourid όπου κάνουμε αναφορά στην κλάση POI.

5.1.4 POIController.java

```
/** Implementing the Methods from POIService */
@Service // stereotype for service layer,same with component annotation
public class POIServiceImpl implements POIService {

    @Autowired
    private POIRepository poiRepository;

    @Override
    public List<POI> findAll() { return poiRepository.findAll(); }

    @Override
    public List<POI> findByTourid(int tourid) { return poiRepository.findByTourid(tourid); }
}
```

Στην παραπάνω εικόνα βλέπουμε την κλάση POIController όπου είναι υπεύθυνη να "παράξει" τα δεδομένα της Β.Δ σε JSON μορφή. Για αυτό χρησιμοποιούμε και το @RestController.

Κάνουμε χρήση του @Autowired για το poiService για να μπορέσει να σκανάρει τις κλάσεις όπως αναφέρθηκε πιο πριν.

Ακολουθούν οι μέθοδοι index() και showTourID με annotation @ResponseBody όπου σημαίνει ότι η επιστρεφόμενες τιμές αποτελούν το body ενός HTTP response.

Τα annotations @RequestMapping μας βοηθούν να φτιάξουμε το URI.

Ορίζουμε το resource μας /poi για το index και /poi{tourid} για το showTourID,

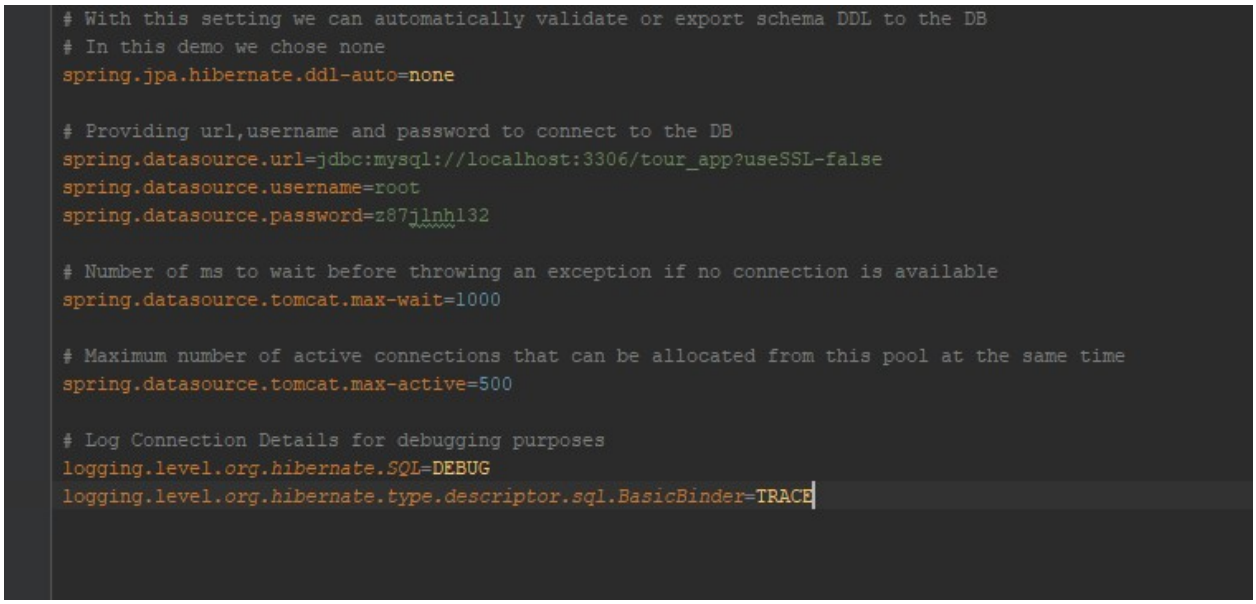
HTTP Method Get και τα δεδομένα που θα επιστρέφουν θα είναι σε JSON μορφή.

```
package com.webservice.restapi;

import ...

@SpringBootApplication
public class SpringJPAApplication {

    public static void main(String[] args) { SpringApplication.run(SpringJPAApplication.class,args); }
}
```

Στην παραπάνω εικόνα βλέπουμε την κεντρική main κλάση με @SpringBootApplication όπου είναι υπεύθυνο για το σκανάρισμα των components.

Πραγματοποιείται με την μέθοδο run().

5.1.5 app.properties



Στην παραπάνω εικόνα βλέπουμε τα app.properties όπου ορίζουμε την σύνδεση με την Β.Δ.

Συγκεκριμένα με το spring.jpahibernate.ddl.-auto ορίζουμε αν θέλουμε να “πειράξουμε” προγραμματιστικά την Β.Δ με κάποιο update κλπ.

Στην περίπτωση μας δεν θέλουμε οπότε βάζουμε none.

Με τα spring.datasource.url ορίζουμε το URL για την Β.Δ όπου βρίσκεται στον τοποικό Η/Υ στην 3306 port.

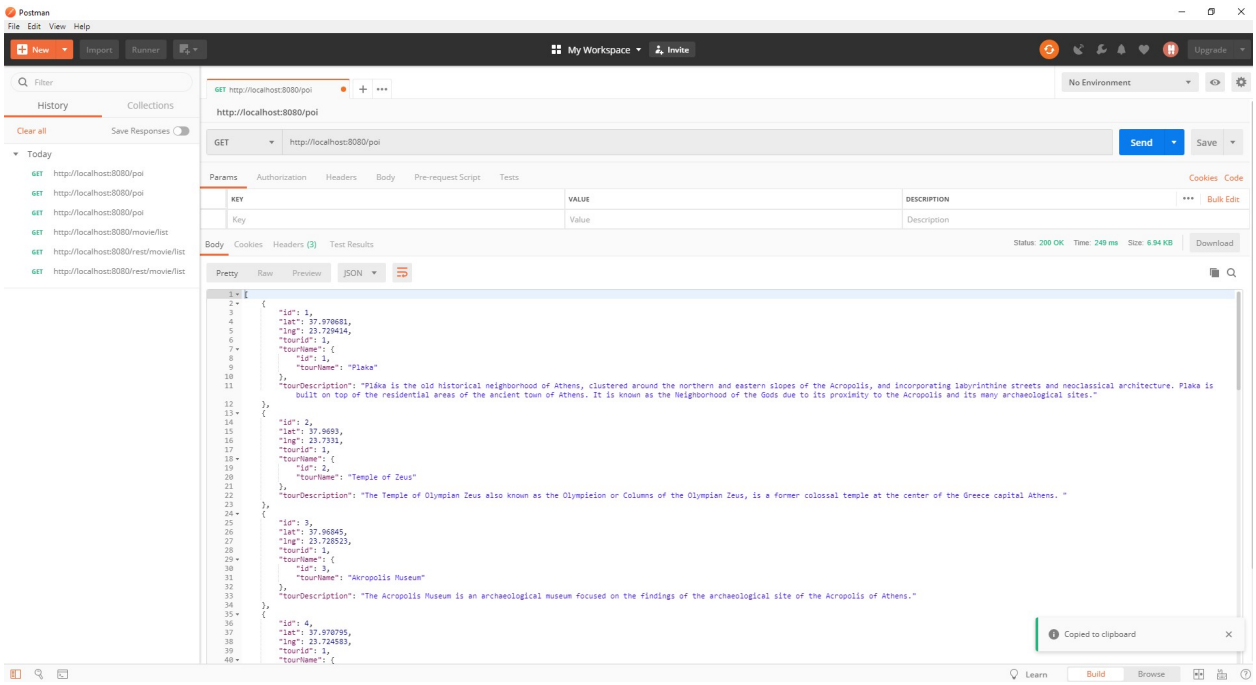
Στο spring.datasource.username και password συπληρώνουμε τα credentials που έχουμε ορίσει από την Β.Δ.

Τα max-wait και max-active είναι στην 1η περίπτωση πόσα ms πρέπει να περιμένει ο tomcat για να εμφανίσει error σε περίπτωση προβλήματος με την σύνδεση στη Β.Δ και στην 2η περίπτωση πόσα connections επιτρέπονται.

Τέλος τα logs είναι για λόγους debugging.

5.1.6 POSTMAN-ENDPOINT TEST

Η δοκιμή της εφαρμογής μπορεί να γίνει με μια εφαρμογή που ονομάζεται postman όπου πατώντας το URL π.χ. http://localhost:8080/poi δοκιμάζουμε τα endpoints που έχουμε φτιάξει. Βλέπουμε ότι επιστρέφει σωστά όλες τις πληροφορίες για τα Tour από την Β.Δ σε μορφή JSON.

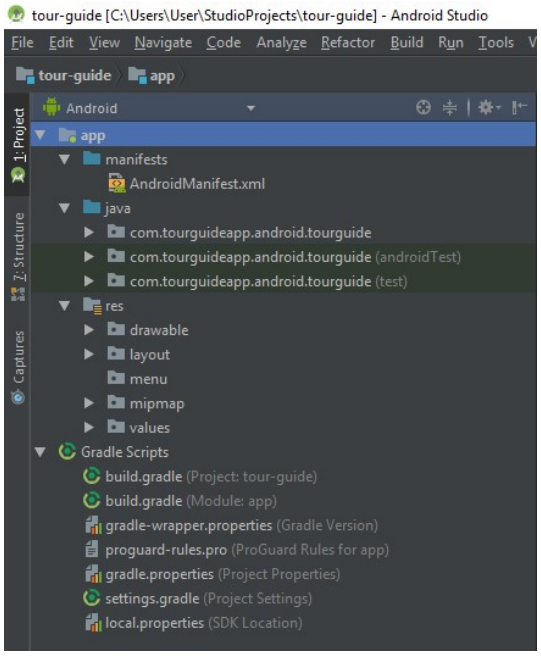


5.2 Frontend

- Το κομμάτι του frontend συνιστά η Android Εφαρμογή, είναι γραμμένη και αυτή σε Java στο Android Studio IDE. Αρχικά θα δείξουμε κάποια σημαντικά dependencies και το integration με το Google Maps API και έπειτα τον κώδικα.

Το Android Studio όπως προαναφέρθηκε βασίζεται στο Intellij IDEA οπότε η διαδικασία δημιουργίας του project ακολουθεί τα ίδια βήματα. Εξαίρεση αποτελεί το γεγονός ότι γίνεται χρήση του Gradle ως Build Tool και όχι του Maven.

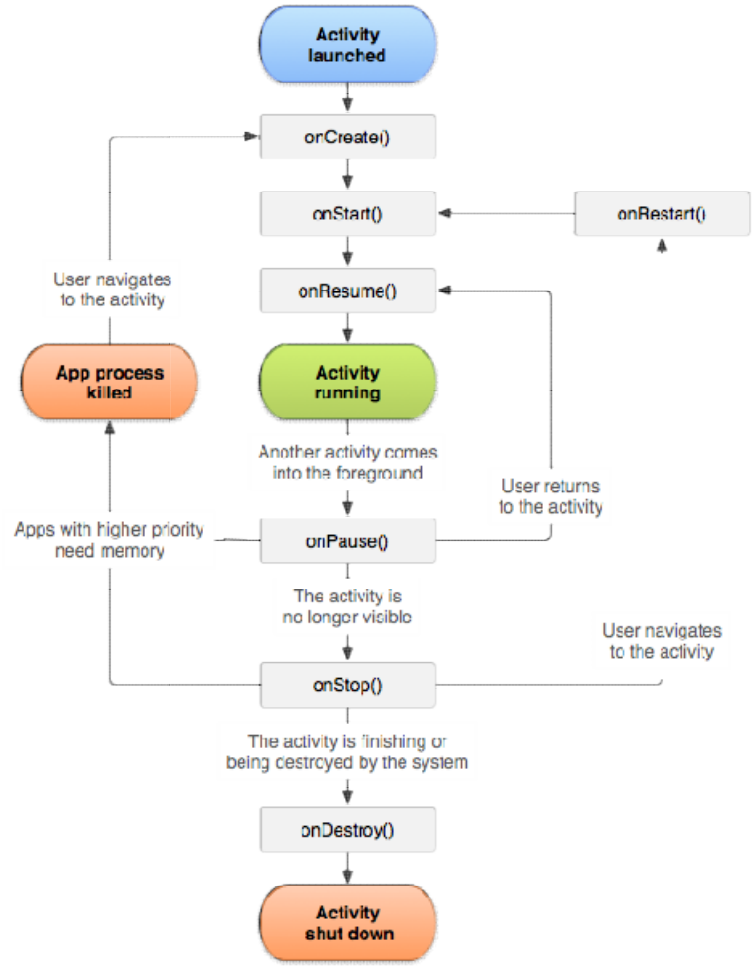
Πριν προχωρήσουμε στην επεξήγηση του κυρίως κώδικα να κάνουμε μια αναφορά στην οργάνωση των φακέλων στο Android Studio αλλά και στο lifecycle μοντέλο που ακολουθεί.



Η ιεραρχική δομή που ακολουθεί το android φαίνεται στην παραπάνω εικόνα. Οι 2 Βασικοί Φάκελοι είναι η εφαρμογή μας app και τα Gradle Scripts.

Στον φάκελο app βρίσκουμε τον φάκελο java όπου περιέχει τα Packages που δημιουργούμε τα java αρχεία μας,res που περιέχουν τους φακέλους drawable, layouts και values όπου μέσα περιέχουν αρχεία xml ώστε να χτίσουμε το UI μας, τις εικόνες που χρειαζόμαστε, τις μορφοποιήσεις της Οθόνης ενώ στο Gradle Scripts οδηγίες προς το Build Tool ως προς στο πως θα γίνει Packaging η εφαρμογή μας.

5.2.1 Android Lifecycle



Εικόνα 16. UI Callbacks

Καθώς ένας χρήστης περιηγείται στην εφαρμογή μας, τα Activity Instances περνούν μέσα από μια σειρά από διαφορετικές καταστάσεις στον κύκλο ζωής τους. Η κλάση "Activity" (Δραστηριότητα) παρέχει μια σειρά από callbacks που επιτρέπουν στο ίδιο το Activity να γνωρίζει ότι έχει αλλάξει μια κατάσταση: ότι το σύστημα δημιουργεί, σταματάει ή επαναλαμβάνει μια δραστηριότητα ή καταστρέφει τη διαδικασία στην οποία βρίσκεται η δραστηριότητα.

Τα κυριότερα callbacks φαίνονται στην παραπάνω εικόνα.

5.2.2 Dependencies

Εφόσον δημιουργήσουμε το Project γίνονται generate κάποια αρχεία,2 είναι τα πιο σημαντικά που θα μας βοηθήσουν στην δόμηση του project,1ο είναι το Android.Manifest και 2ον είναι το build.gradle (module :app) όπου ορίζουμε κάποια βασικά Dependencies που θα μας χρειαστούν.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 26
    defaultConfig {
        applicationId "com.tourguideapp.android.tourguide"
        minSdkVersion 16
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    implementation 'com.android.support:appcompat-v7:26.1.0'
    /* Essential Google Maps dependencies */
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    implementation 'com.google.android.gms:play-services-maps:11.8.0'
    implementation 'com.google.android.gms:play-services-location:11.8.0'
    implementation 'com.google.android.gms:play-services-places:11.8.0'
    implementation 'com.google.maps.android:android-maps-utils:0.5'
    implementation 'com.googlecode.json-simple:json-simple:1.1' //JSON support
    implementation 'com.parse:parse-android:1.12.0'
    /* Essential REST Client dependencies */
    implementation 'com.squareup.retrofit2:retrofit:2.5.0' //Retrofit
    /* Jackson JSON Converter and Modules */
    implementation 'com.squareup.retrofit2:converter-jackson:2.2.0'
    implementation 'com.fasterxml.jackson.core:jackson-databind:2.8.5'
    implementation 'com.fasterxml.jackson.core:jackson-core:2.8.5'
    implementation 'com.fasterxml.jackson.core:jackson-annotations:2.8.5'
    implementation 'com.squareup.okhttp3:logging-interceptor:3.6.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.1'
    implementation 'com.android.support:design:26.1.0'

}
```

Στην παραπάνω εικόνα βλέπουμε το αρχείο build.gradle(module :app) όπου ορίζουμε μέσα στα dependencies τις Βιβλιοθήκες που θα χρειαστούμε ώστε να κάνουμε χρήση των Google Maps,του HTTP client Retrofit και του parser Jackson.

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

    <!-- Permissions used: Network,Location Services,Waking up phone when locked -->
    <!--
         The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
         Google Maps Android API v2, but you must specify either coarse or fine
         location permissions for the 'MyLocation' functionality.
    -->
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
```

```xml
        <!-- Intro Activity to show Slider -->
        <activity
            android:name=".WelcomeActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

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

        <!-- Main Menu -->
        <activity
            android:name=".MainMenu"
            android:screenOrientation="portrait">
        </activity>
        <!--
             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="AIzaSyDVxhFApFBJtoa0ALPTdSC1Zg0VK9555fc" />

        <!-- Main Activity which shows the Map -->
        <activity
            android:name=".MapTour"
            android:label="Map"
            android:screenOrientation="portrait"
            android:parentActivityName=".MapTourList">
        </activity>

        <!-- User's choice of preferable Route -->
        <activity
            android:name=".MapTourList"
            android:label="Choose your Tour"
            android:screenOrientation="portrait"
            android:parentActivityName=".MainMenu">
        </activity>
```

Στις πιο πάνω εικόνες βλέπουμε το Android.Manifest και τα permissions που χρειαζόμαστε από τον χρήστη.

5.2.3 Manifest.xml

Κάθε εφαρμογή πρέπει να έχει ένα αρχείο AndroidManifest.xml (με ακριβώς αυτό το όνομα) που βρίσκεται στον φάκελο Manifests. Το Manifest περιγράφει βασικές πληροφορίες σχετικά με την εφαρμογή στο Build Tool του Android, στο Λ.Σ του Android και στο Google Play.

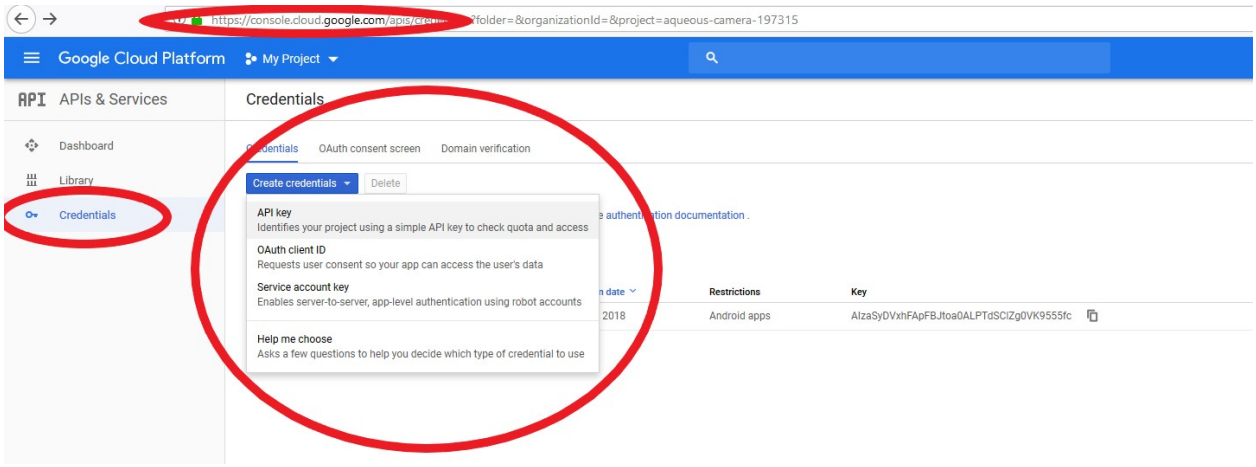
Μεταξύ άλλων, το manifest υποχρεούται να δηλώσει τα εξής:

- Το όνομα του πακέτου της εφαρμογής, το οποίο συνήθως αντιστοιχεί στο χώρο ονομάτων του κώδικα. Το Build Tool του Android χρησιμοποιούν αυτό για να προσδιορίσουν τη θέση των οντοτήτων του κώδικα κατά το packaging της εφαρμογής.
- Τα components της εφαρμογής, τα οποία περιλαμβάνουν όλα τα activities, τα services, τους broadcast receivers και τα content providers. Κάθε component πρέπει να ορίζει βασικές ιδιότητες όπως το όνομα της κλάσης Kotlin ή Java. Μπορεί επίσης να δηλώσει δυνατότητες όπως τι τύπους συσκευών που μπορεί να χειριστεί και intent filters που περιγράφουν πώς μπορεί να ξεκινήσει ένα component.
- Τα Permissions (δικαιώματα) που χρειάζεται η εφαρμογή για πρόσβαση σε προστατευμένα μέρη του συστήματος ή άλλων εφαρμογών. Επίσης, δηλώνει τα δικαιώματα που πρέπει να έχουν άλλες εφαρμογές, αν θέλουν να έχουν πρόσβαση σε περιεχόμενο από αυτήν την εφαρμογή.
- Τα hardware και software χαρακτηριστικά που απαιτεί η εφαρμογή, τα οποία επηρεάζουν τις συσκευές που μπορούν να εγκαταστήσουν την εφαρμογή από το Google Play.

  Το Android Studio, δημιουργεί το manifest για εμάς και τα περισσότερα από τα απαραίτητα elements προστίθενται καθώς δημιουργούμε την εφαρμογή.

Ιδιαίτερη προσοχή να δοθεί στο tag meta-data που περιέχει το API key που έχει γίνει generate από το Google API Console.

5.2.4 Δημιουργία API Key



Παραπάνω φαίνεται ένα στιγμιότυπο από την δημιουργία.

Συγκεκριμένα έχουμε:

1. Μεταβαίνουμε στο Google Cloud Platform.

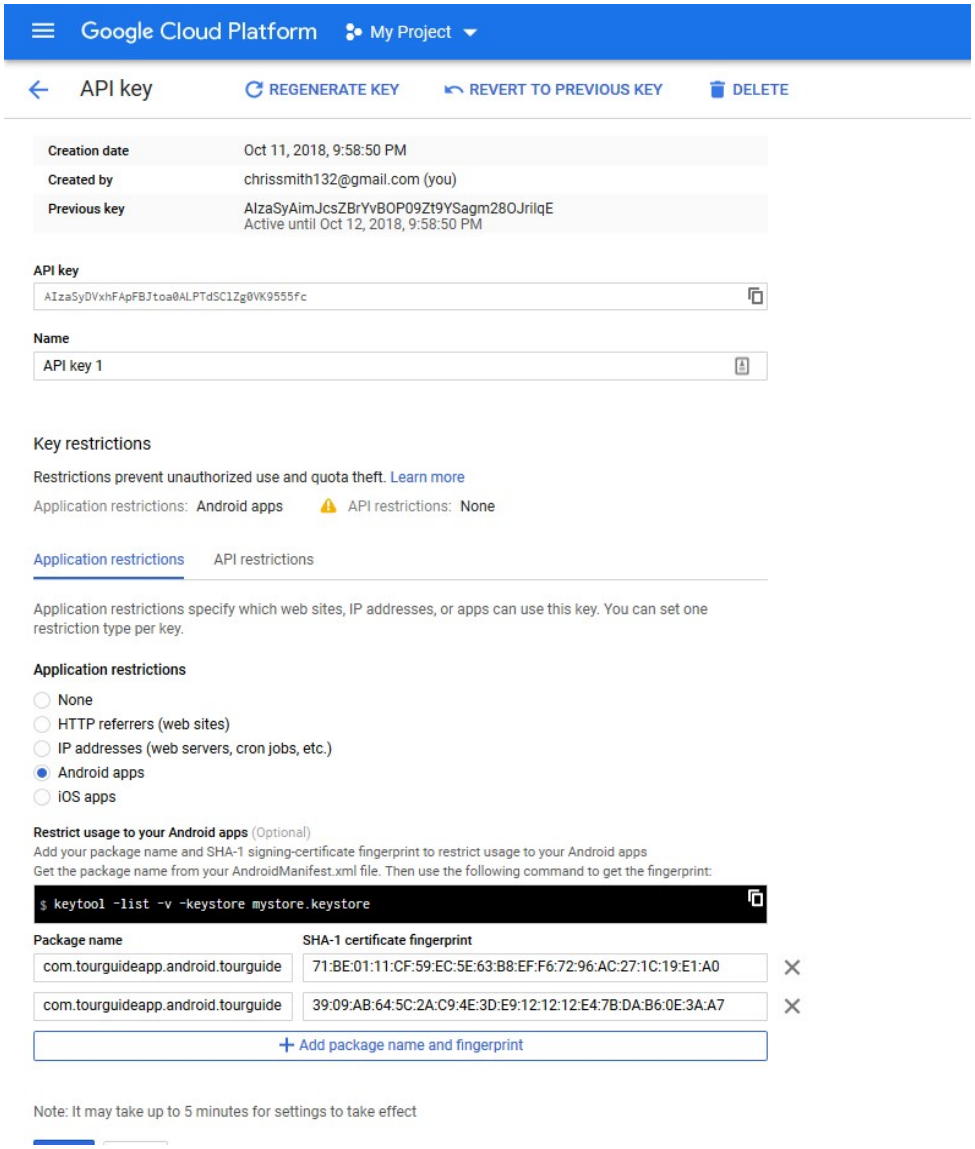
2. Από το αναπτυσσόμενο μενού Project, επιλέγουμε ή δημιουργούμε το project για το οποίο θέλουμε να προσθέσουμε ένα κλειδί API.

3. Από το μενού πλοήγησης, επιλέξτε APIs & Services> Credentials.

4. Στη σελίδα "Credentials", κάνουμε κλικ στην επιλογή Create Credentials> Key API.

5. Το παράθυρο διαλόγου που δημιουργήθηκε με το κλειδί API εμφανίζει το νέο δημιουργημένο κλειδί API.
   Στο παράθυρο διαλόγου, κάντε κλικ στην επιλογή Περιορισμοί κλειδιών.
6. Στη σελίδα κλειδιού API, στην ενότητα Βασικοί περιορισμοί, ορίστε τους περιορισμούς εφαρμογής.
   Ορίζουμε Android Apps
7. Κάνουμε κλικ στην επιλογή Αποθήκευση.

Στο τέλος θα καταλήξουμε να έχουμε ότι φαίνεται στην παρακάτω εικόνα.



Το API key που φαίνεται είναι αυτό που βάζουμε στο tag meta-data στο manifest ώστε να έχουμε πρόσβαση στα Google Maps.

Τέλος θα πρέπει να συμπληρώσουμε το package name της εφαρμογής μας και το SHA-1 fingerprint ώστε να λειτουργήσουν τα Google Maps.

5.2.5 MainMenu.java

Το αρχικό activity που δημιουργούμε είναι ένα αρκετά απλό καθώς σετάρουμε στην βασική μέθοδο onCreate() να δείχνει ένα UI με ένα image button στην μέση όπου δημιουργήθηκε με σκοπό να τσεκάρει απλά αν ο χρήστης έχει ήδη ανοιχτό Wi-Fi, Mobile Data ή GPS. Σε περίπτωση μου δεν είναι ανοιχτό κάποια από τα τρία εμφανίζεται ένα μήνυμα μορφής Alert Dialog ώστε να προτρέψει τον χρήστη να ενεργοποιήσει κάποιο από αυτά ώστε να λειτουργήσει η εφαρμογή.

```
package com.tourguideapp.android.tourguide;

import ...

/** Main Menu of the App  */
public  class MainMenu extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_menu);
    }

   // Check on Startup of the App if Wifi or Mobile Data is enabled
    @Override
    public void onStart()
    {
        super.onStart();
        // check if Network Connection or GPS is enabled
        if(!isNetworkConnected() && !isGPSEnabled())
        {
            // Create an Alert Dialog Message
            AlertDialog.Builder builder=new AlertDialog.Builder( context: this);
            builder.setMessage("Internet Connection or GPS is Required," +
                    "Please enable your WiFi/Mobile Data or GPS")
                    .setCancelable(false)
                    // kill the app
                    .setPositiveButton( text: "Retry", (dialogInterface, i) -> {
                            finish();

                    });
            AlertDialog alert=builder.create();
            alert.show();
        }


    }
```

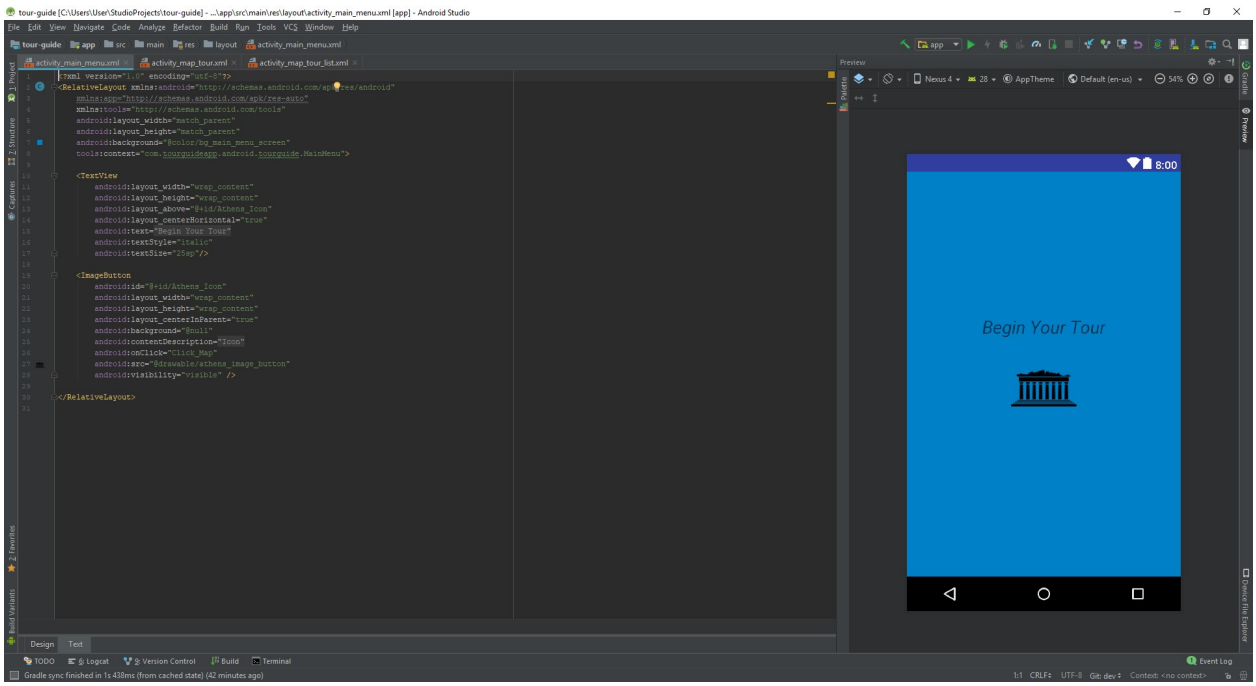
Ιδιαίτερη αναφορά για το callback onCreate() που θα μας συνοδέψει σε όλα τα activities καθώς μέσα στην μέθοδο αυτή κάνουμε τις αρχικοποιήσεις μας και σετάρουμε το UI από τα αρχεία XML.

```
/* Implementing the method Click_Map
  When the user clicks on the map icon
  The Map Layout is showed
 */
public void Click_Map(View view)
{
    Intent MapChoice=new Intent( packageContext: this,MapTourList.class);
    startActivity(MapChoice);
}

/* Check if the device is connected either to WiFi or Mobile Data  */
private boolean isNetworkConnected()
{
    ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
    if (activeNetwork != null)  // connected to the internet
    {
        if (activeNetwork.getType() == ConnectivityManager.TYPE_WIFI)
        {
            // connected to WiFi
            return true;

        }
        else if (activeNetwork.getType() == ConnectivityManager.TYPE_MOBILE)
        {
            // connected to the mobile provider's data plan
            return true;
        }
    } else {
        // not connected to the internet
        return false;
    }
    return false;
}

private boolean isGPSEnabled()
{
    LocationManager location_manager=(LocationManager) getSystemService(Context.LOCATION_SERVICE);
    boolean gpsProvider=location_manager.isProviderEnabled(LocationManager.GPS_PROVIDER);
    return gpsProvider;
}
```

Οι 2 μέθοδοι που εκπληρώνουν την δουλειά για τσεκάρισμα isNetworkConnected() και isGPSEnabled().

Το Click_Map() χρησιμεύει ώστε να πατώντας πάνω στο image-button να περάσουμε στο επόμενο Activity που είναι η λίστα με τις προκαθορισμένες διαδρομές μας.

5.2.6 activity_main_menu.xml



Όπως προαναφέρθηκε τα αρχεία XML βοηθούν ώστε να φτιάξουμε το GUI των activity μας.

Το android XML περιέχει ειδικές δεσμευμένες λέξεις για την δημιουργία των layouts όπως Relative, Linear, Constraint αλλά και ειδικά elements όπως textview, buttons κλπ. Έτσι μας διευκολύνει στην ανάπτυξη του UI layout μας.

Παραπάνω φαίνεται το Κεντρικό Μενού μας που είναι οργανωμένο σαν Relative με 2 στοιχεία ένα textview που απεικονίζει το String Begin Your Tour και ένα image button που απεικονίζει την ακρόπολη.

5.2.7 MapTourList.java

```
public class MapTourList extends AppCompatActivity
{
    Button BtnFirstTour,BtnSecondTour,BtnThirdTour;
    double Starting_LocPoint_Lat;
    double Starting_LocPoint_Lng;
    double Destination_LocPoint_Lat;
    double Destination_LocPoint_Lng;

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

        // Declaring the Buttons for the MapTourList Layout
        BtnFirstTour=findViewById(R.id.First_Tour);
        BtnSecondTour=findViewById(R.id.Second_Tour);
        BtnThirdTour=findViewById(R.id.Thid_Tour);

        // Declaring the "Up Button"
        if (getActionBar() != null) {
            getActionBar().setDisplayHomeAsUpEnabled(true);
        }

    }

    /** Implementing the "Up Button" to go back in Parent Activity */
    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId())
        {
            // Respond to the action bar's Up/Home button
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask( sourceActivity: this);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

Στο activity αυτό ορίζουμε την λίστα με τις διαδρομές μας όπου πατώντας μια από τις διαδρομές αντλούμε τα δεδομένα από την Β.Δ ώστε να εμφανιστεί ο χάρτης στο επόμενο activity.

Συγκεκριμένα εδώ στην onCreate() ορίζουμε τα 3 image-buttons και το κουμπί “πίσω” που εμφανίζεται στο πάνω μέρος της οθόνης.

```
/** Clicking the Image-Button shows the Google Map
 *  and fetches async Lat/Lng Starting and Destination coordinates from the API
 * */
public void Click_Map_Tour(View view)
{
    //Creating an Intent to go from MapTourList to MapTour
    Intent TourChoice = new Intent( packageContext: this, MapTour.class);

    //Creating a bundle to send the Longitude and Latitude of the Start/End Marker Points to MapTour
    Bundle lat_long=new Bundle();

    switch(view.getId())
    {
        // send Plaka-Monastiraki coordinates
        case R.id.First_Tour:
            FetchTourData( id: 1);
            lat_long.putParcelable("Start_Location",new LatLng(Starting_LocPoint_Lat,Starting_LocPoint_Lng));
            lat_long.putParcelable("Dest_Location",new LatLng(Destination_LocPoint_Lat,Destination_LocPoint_Lng));
            lat_long.putString("Tour_Name","First_Tour");
            TourChoice.putExtra( name: "Chosen_Tour",lat_long);
            startActivity(TourChoice);
            break;

        // send Syntagma-Thisio coordinates
        case R.id.Second_Tour:
            FetchTourData( id: 2);
            lat_long.putParcelable("Start_Location",new LatLng(Starting_LocPoint_Lat,Starting_LocPoint_Lng));
            lat_long.putParcelable("Dest_Location",new LatLng(Destination_LocPoint_Lat,Destination_LocPoint_Lng));
            lat_long.putString("Tour_Name","Second_Tour");
            TourChoice.putExtra( name: "Chosen_Tour",lat_long);
            startActivity(TourChoice);
            break;

        // send Akropolis-Zappeion coordinates
        case R.id.Thid_Tour:
            FetchTourData( id: 3);
            lat_long.putParcelable("Start_Location",new LatLng(Starting_LocPoint_Lat,Starting_LocPoint_Lng));
            lat_long.putParcelable("Dest_Location",new LatLng(Destination_LocPoint_Lat,Destination_LocPoint_Lng));
            lat_long.putString("Tour_Name","Third_Tour");
            TourChoice.putExtra( name: "Chosen_Tour",lat_long);
            startActivity(TourChoice);
            break;
    }

}
```

Στην click_Map_Tour() ορίζουμε ποια διαδρομή θα πατήσει ο χρήστης και ανάλογα ζητάμε το πρώτο και τελευταίο σημείο ενδιαφέροντος από την Β.Δ με την μέθοδο FetchTourData, ώστε στο επόμενο activity να βρεθούν τα ενδιάμεσα σημεία.

Επιπλέον δημιουργούμε και ένα bundle όπου αποτελεί σαν ένα προσωρινό αποθηκευτικό χώρο ώστε να αποθηκεύσουμε τα δεδομένα που λάβαμε (1ο και τελευταίο σημείο ενδιαφέροντος ή poi) και με την βοήθεια του intent να σταλθούν στο επόμενο activity.

Ιδιαίτερα η κλάση Intent παρέχει μια δυνατότητα εκτέλεσης σύνδεσης μεταξύ του κώδικα σε διάφορες εφαρμογές. Η πιο σημαντική χρήση του είναι η έναρξη των activities, όπου μπορεί να θεωρηθεί ως “κόλλα” μεταξύ activities.

Είναι βασικά μια δομή δεδομένων που περιέχει μια αφηρημένη περιγραφή μιας ενέργειας που πρέπει να εκτελεστεί.

```
/** Network Request to fetch data from API */
public void FetchTourData(int id)
{
    /* Create handle for the Retrofit-Instance */
    GetDataService service=RetrofitInstance.getRetrofitInstance().create(GetDataService.class);

    /* Call the method with parameter in the interface to get the POI data */
    Call<List<POI>> call=service.getPOIByTourID(id);

    /* Log the URL called */
    Log.wtf( tag: "URL Called", msg: call.request().url() + "");

    /* make an async call */
    call.enqueue(new Callback<List<POI>>() {
        @Override
        public void onResponse(Call<List<POI>> call, Response<List<POI>> response)
        {

            if(response.isSuccessful())
            {
                List<POI> poi = response.body();
                Log.i( tag: "POI_RESPONSE_SIZE", String.valueOf(poi.size()));
               for(POI p: poi) //iterate the poi List to fetch Lat/Lng
               {
                    Starting_LocPoint_Lat= poi.get(0).getLat();
                    Starting_LocPoint_Lng=poi.get(0).getLng();
                    Destination_LocPoint_Lat=poi.get(6).getLat();
                    Destination_LocPoint_Lng=poi.get(6).getLng();
               }


            }
            else
            {
                //Log the HTTP Response Code
                Log.d( tag: "RESPONSE_CODE", msg: response.code()+"");
            }

        }

        @Override
        public void onFailure(Call<List<POI>> call, Throwable t) {
            Toast.makeText(getApplicationContext(), text: "Unable to Fetch Data from Server",Toast.LENGTH_SHORT).show();
            Log.d( tag: "RESPONSE_FAILURE",t.getMessage());
        }
    });

}
```

Στην μέθοδο FetchTourData πραγματοποιείται ένα ασύγχρονο request στην Β.Δ ώστε να αντλήσουμε τα δεδομένα μας.

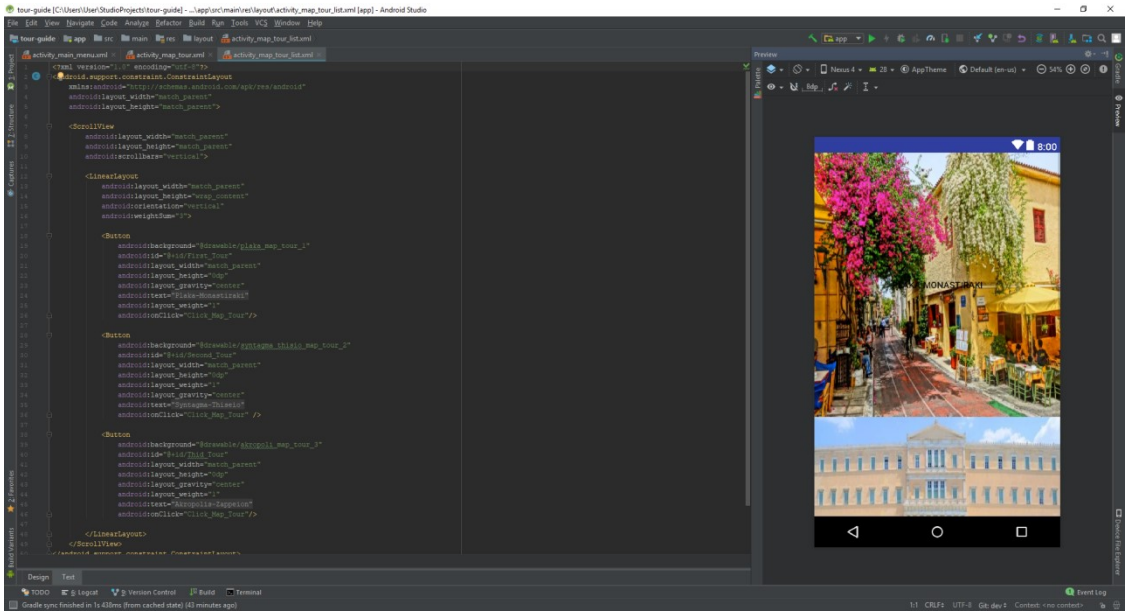
Σε αυτό μας βοηθάει το Retrofit όπου δημιουργούμε ένα Instance του και ένα ειδικό interface που ονομάζεται Call όπου περιέχει την ειδική μέθοδο enqueue ώστε να κάνουμε ασύγχρονο request.

Ιδιαίτερα έχουμε 2 ειδικές μεθόδους για αυτό το λόγο.

Στην onResponse() εφόσον γίνει επιτυχής το request λαμβάνουμε τα δεδομένα μας.

Στο onFailure() εφόσον έχουμε αποτυχημένη σύνδεση δημιουργούμε ένα μήνυμα ότι έχουμε αποτυχημένο request.

5.2.8 activity_map_tour_list.xml



Για το UI του MapTourList δημιουργούμε ένα scroll view με Linear Layout ώστε η οθόνη να είναι κυλιόμενη και ο χρήστης να επιλέξει την επιθυμητή διαδρομή.

Τέλος μέσα στο scroll-view έχουμε ορίσει 3 image-buttons που αντιπροσωπεύουν τις 3 προκαθορισμένες διαδρομές μας.

5.2.9 MapTour.java

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

    // Receive the Lat Long coordinates from MapTourList
    Bundle get_long_lang = getIntent().getParcelableExtra( name: "Chosen_Tour");
    if (get_long_lang != null) {
        Start_position = get_long_lang.getParcelable( key: "Start_Location");
        Dest_position = get_long_lang.getParcelable( key: "Dest_Location");
        correspond_waypoints = get_long_lang.getString( key: "Tour_Name");
        Log.i( tag: "Coordinates",  msg: "Coordinates OK!");
        start_point_lat = Start_position.latitude;
        start_point_lng = Start_position.longitude;
        dest_point_lat = Dest_position.latitude;
        dest_point_lng = Dest_position.longitude;

    } else {
        Toast.makeText( context: this,  text: "Something went Wrong", Toast.LENGTH_LONG).show();
        Log.d( tag: "Lat_Long_Bundle",  msg: "NULL Lat Long");
    }

    // Obtain the SupportMapFragment and get notified when the map is ready to be used
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
    mapFragment.getMapAsync( onMapReadyCallback: this);

    // Construct a FusedLocationProviderClient
    mFusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);

    // Initializing the ArrayList and adding the corresponding Waypoints to each of the Chosen Tour
    MarkerPoints = new ArrayList<>();

    switch (correspond_waypoints) {
        case "First_Tour":
            FetchTourData( id: 1);
            break;
        case "Second_Tour":
            FetchTourData( id: 2);
            break;
        case "Third_Tour":
            FetchTourData( id: 3);
            break;
    }
```

Το MapTour αποτελεί το τελευταίο και κεντρικό Activity όπου εμφανίζεται ο χάρτης και ο χρήστης περιηγείται στα διάφορα POIs.

Στο onCreate() κάνουμε τις απαραίτητες αρχικοποιήσεις που φαίνονται πιο πάνω όπως το να σετάρουμε το layout που φιλοξενεί τα Google Maps, λαμβάνουμε τα δεδομένα από το bundle από το προηγούμενο activity, κάνουμε async request για να εμφανιστεί ο χάρτης, εκκίνηση των LocationServices για το να μπορούμε να εντοπίζουμε την εκάστοτε τοποθεσία του χρήστη και εκτέλεση του FetchTourData όπως πριν ώστε να λάβουμε τα ενδιάμεσα POIs βάση ποιας διαδρομής επέλεξε ο χρήστης.

```java
/** Check Location Permissions on Startup */
@Override
public void onStart() {
    super.onStart();
    if (!checkPermissions()) {
        checkLocationPermission();
    }
}

/** Stop location updates when Activity is no longer active */
@Override
public void onPause() {
    super.onPause();
    if (mFusedLocationProviderClient != null) {
        mFusedLocationProviderClient.removeLocationUpdates(mLocationCallback);
    }
}

/** Resume Location Updates when Activity is active again */
@Override
public void onResume()
{
    super.onResume();
    if(mLastLocation!=null)
    {
        calculateDistance();
    }
    else
    {
        startLocationUpdates();
    }
}
```

Στην συνέχεια έχουμε τα 3 βασικά callbacks.

Στο onStart() ελέγχουμε με το checkLocationPermission αν ο χρήστης μας επιτρέπει να έχουμε πρόσβαση στην εκάστοτε τοποθεσία του.

Στο onPause() σταματάμε το LocationRequest εφόσον ο χρήστης φύγει από την εφαρμογή μας. Υπάρχει για λόγους εξοικονόμησης της μπαταρίας της συσκευής.

Στο onResume() επαναφέρουμε το Location Update και το CalculateDistance που μετράει κάθε στιγμή αν ο χρήστης βρίσκεται κοντά σε κάποιο POI.

```java
/** Network Request to fetch data from API */
public void FetchTourData(final int id)
{
    /* Create handle for the Retrofit-Instance */
    GetDataService service= RetrofitInstance.getRetrofitInstance().create(GetDataService.class);

    /* Call the method with parameter in the interface to get the POI data */
    Call<List<POI>> call=service.getPOIByTourID(id);

    /* Log the URL called */
    Log.wtf( tag: "URL Called", msg: call.request().url() + "");

    /* make an async call */
    call.enqueue(new Callback<List<POI>>() {
        @Override
        public void onResponse(Call<List<POI>> call, Response<List<POI>> response)
        {

            if(response.isSuccessful())
            {
                List<POI> poi = response.body();
                Log.i( tag: "POI_RESPONSE_SIZE", String.valueOf(poi.size()));
                for(int j=1; j<=5; ++j) //iterate the poi List to fetch Lat/Lng,TourName and Tour Description
                {
                    LatWaypoints=poi.get(j).getLat();
                    LngWaypoints=poi.get(j).getLng();
                    tourName=poi.get(j).getTourName();
                    tour_name=tourName.getTourName();
                    tourDescription=poi.get(j).getTourDescription();
                    MarkerPoints.add(new LatLng(LatWaypoints,LngWaypoints));
                    //GeofenceUniqueID.add(tour_name);
                    addMarkerPoints();
                    generatingGeofenceID();
                }


            }
            else
            {
                //Log the HTTP Response Code
                Log.d( tag: "RESPONSE_CODE", msg: response.code()+"");
            }

        }

        @Override
        public void onFailure(Call<List<POI>> call, Throwable t) {
            Toast.makeText(getApplicationContext(), text: "Unable to Fetch Data from Server",Toast.LENGTH_SHORT).show();
            Log.d( tag: "RESPONSE_FAILURE",t.getMessage());
        }
    });
```

Το FetchTourData() λειτουργεί όπως αναλύθηκε πιο πριν με μόνη διαφορά ότι λαμβάνει εκτός από τις γεωγραφικές συντεταγμένες και το όνομα του POI μαζί με το description ώστε παρακάτω να τα θέσουμε ως πληροφορίες πάνω στα διάφορα Markers.

```
/**
 * Manipulates the map once available.
 * This callback is triggered when the map is ready to be used.
 * This is where we can add markers or lines, add listeners or move the camera.
 * This method will only be triggered once the user has
 * installed Google Play services and returned to the app.
 */
@Override
public void onMapReady(GoogleMap googleMap)
{
    mMap = googleMap;

    // updating every 2 minutes the current location of the user
    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(120000); // two minute interval
    mLocationRequest.setFastestInterval(120000);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY);

    // granting permission and enabling to find the current location
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
    {
        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED)
        {
            //Location Permission already granted
            mFusedLocationProviderClient.requestLocationUpdates(mLocationRequest, mLocationCallback, Looper.myLooper());
            mMap.setMyLocationEnabled(true);
        }
        else
        {
            //Request Location Permission
            checkLocationPermission();
        }
    }
    else
    {
        mFusedLocationProviderClient.requestLocationUpdates(mLocationRequest, mLocationCallback, Looper.myLooper());
        mMap.setMyLocationEnabled(true);
    }
```

Το onMapReady() είναι ένα callback interface για όταν ο χάρτης είναι έτοιμος για χρήση.

Μόλις ένα instance του interface έχει οριστεί σε ένα MapFragment ή MapView αντικείμενο (βλέπε μέθοδος onCreate()), η onMapReady (GoogleMap) μέθοδος ενεργοποιείται όταν ο χάρτης είναι έτοιμος να χρησιμοποιηθεί και παρέχει έναν non-null instance του Google Map.

Επιπλέον μέσα στην μέθοδο κάνουμε τις απαραίτητες αρχικοποιήσεις όσον αφορά τα διάφορα Markers,γραμμές για ένωση μεταξύ των Markers και μετακίνηση της κάμερας του χρήστη πάνω στον χάρτη.

Συγκεκριμένα εδώ κάνουμε χρήση του LocationRequest ώστε κάθε 2 λεπτά να εντοπίζουμε την εκάστοτε τοποθεσία του χρήστη.

```
    // Adding and Styling the predefined Markers of the User's Tour choice
    Marker Start_Pos=mMap.addMarker(new MarkerOptions()
            .position(new LatLng(start_point_lat,start_point_lng))
            .title("Start")
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN))
    );
    Start_Pos.showInfoWindow();

    Marker End_Pos= mMap.addMarker(new MarkerOptions()
            .position(new LatLng(dest_point_lat,dest_point_lng))
            .title("End"));
    // End_Pos.showInfoWindow();

    // Adding the Waypoint Markers to MarkerPoints List
    addMarkerPoints();

    // Custom Info window in order to show the whole text of snippet
    CustomInfoWindowMap adapter=new CustomInfoWindowMap( context: MapTour.this);
    mMap.setInfoWindowAdapter(adapter);

    // Send LatLng and fetch directions for the Markers
  /*  String url = getUrl(Start_position,Dest_position);
    FetchUrl FetchUrl = new FetchUrl();
    FetchUrl.execute(url);*/

    // zoom Camera Map to markers
    mMap.animateCamera(CameraUpdateFactory.zoomTo( v: 11));
}
```

Στη συνέχεια ορίζουμε τα αρχικά και τελικά markers και έπειτα καλούμε το addMarkerPoints() ώστε να συμπληρωθούν τα ενδιάμεσα Markers.

Το CustomInfoWindowMap αποτελεί ιδιαίτερο object όπου κάνουμε implement ένα custom παράθυρο ώστε να χωράει όλο το description που έχουμε από την Β.Δ, στον marker όταν πατάει ο χρήστης πάνω του.

Τέλος ορίζουμε και ένα zoom ώστε η κάμερα του Google Maps να εστιάσει κοντά στα σημεία ενδιαφέροντος.

```
// Adding the Waypoint Markers to MarkerPoints List and connect them with polyline
protected void addMarkerPoints()
{
    for(int i=0; i<MarkerPoints.size(); ++i)
    {
        LatLng position=MarkerPoints.get(i);
        addMarkerToMap(position);
        mMap.addPolyline(new PolylineOptions().color(Color.RED)
                .addAll(
                        MarkerPoints
                ));
    }
}

// Add Waypoint Markers on Google Maps
protected void addMarkerToMap(LatLng latlng)
{
    marker = mMap.addMarker(new MarkerOptions()
            .position(latlng)
            .title(tour_name)
            .snippet(tourDescription)
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_ORANGE)));
    //marker.add(marker);
    //marker.showInfoWindow();

}

private void calculateDistance()
{
    float[] results = new float[1];
    float distance=0;
    for(int i=0; i<MarkerPoints.size(); i++)
    {
        mLastLocation.distanceBetween(this.mLastLocation.getLatitude(), this.mLastLocation.getLongitude(),MarkerPoints.get(i).latitude,MarkerPoints.get(i).longitude, results);
        distance = results[0];
        Log.i( tag: "DistanceBetweenLoc_POI",String.valueOf(distance));
        if(distance<=15)
        {
            Toast.makeText( context: this, text: "You Arrived!",Toast.LENGTH_SHORT).show();
        }
    }

}
```

Στην παραπάνω εικόνα βλέπουμε την υλοποίηση των 3 μεθόδων που χρησιμοποιήθηκαν νωρίτερα.

Συγκεκριμένα το addMarkerPoints() καλεί το addMarkersToMap για όλα τα POIs ώστε να φανούν στον χάρτη με την αντίστοιχη περιγραφή τους αλλά και να συνδεθούν με μια γραμμή.

Το calculateDistance μετράει κάθε στιγμή την απόσταση του χρήστη από το εκάστοτε σημείο ενδιαφέροντος ώστε να εμφανιστεί το μήνυμα ότι έφτασε στον προορισμό του.

```
private void startLocationUpdates() {
    /* Implementing onLocationResult which handles current place of the user */
        mLocationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            for (Location location : locationResult.getLocations()) {
                Log.i( tag: "MapsActivity",  msg: "Location: " + location.getLatitude() + " " + location.getLongitude());
                mLastLocation = location;

                //Place current Location Marker
                CurrentLocation = new LatLng(location.getLatitude(), location.getLongitude());
                calculateDistance();

                // Send LatLong of current position and draw the route between current and start location
            /* String url_2=getUrl(CurrentLocation,Start_position);
             FetchUrl FetchUrl = new FetchUrl();
             FetchUrl.execute(url_2);*/

                //move map camera
                mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(CurrentLocation,  v: 11));
                mMap.animateCamera(CameraUpdateFactory.zoomTo( v: 11));
            }
        }

    };
}
```

Στην παραπάνω εικόνα βλέπουμε την υλοποίηση του LocationUpdate που τροφοδοτεί το OnMapReady() ώστε να ανανεώνεται η εκάστοτε τοποθεσία του χρήστη.

Τέλος στις παρακάτω εικόνες φαίνεται η διαχείριση των permissions ώστε να ζητάμε από τον χρήστη ότι συναινεί να χρησιμοποιούμε την τοποθεσία του με τα αντίστοιχα Alert Dialogs.

```
/** Implementing onRequestPermissionsResult where we handle user's choice for permission  */
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String permissions[],
                                       @NonNull int[] grantResults)
{
    switch (requestCode)
    {
        case PERMISSIONS_REQUEST_CODE:
        {
            // permission was granted
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED)
            {
                if (ContextCompat.checkSelfPermission( context: this,
                        Manifest.permission.ACCESS_FINE_LOCATION)
                        == PackageManager.PERMISSION_GRANTED)
                {
                    mFusedLocationProviderClient.requestLocationUpdates(mLocationRequest, mLocationCallback, Looper.myLooper());
                    mMap.setMyLocationEnabled(true);
                }

            }
            else if(grantResults.length<= 0) // If request is cancelled, the grant result arrays are empty
            {
                Log.i( tag: "PERMISSSIONS REQUEST", msg: "User Interaction was cancelled");
            }
            else
            {
                // Permission was denied,
                // Disable the functionality that depends on this permission.
                Toast.makeText( context: this,  text: "Permission Denied", Toast.LENGTH_LONG).show();
                Log.i( tag: "PERMISSIONS REQUEST", msg: "User denied the functionality");
            }
            //return;
        }

    }

}
```

```
/** Return the current state of the permissions needed */
private boolean checkPermissions()
{
    int permissionState = ActivityCompat.checkSelfPermission( context: this,
            Manifest.permission.ACCESS_FINE_LOCATION);
    return permissionState == PackageManager.PERMISSION_GRANTED;
}

/** Implementing checkLocationPermission where user has to confirm for permission to use his Location  */
private void checkLocationPermission()
{
        boolean shouldProvideRationale =
                ActivityCompat.shouldShowRequestPermissionRationale( activity: this,
                        Manifest.permission.ACCESS_FINE_LOCATION);

        // Should we show an explanation?
        if (shouldProvideRationale)
        {

            // Show an explanation to the user *asynchronously* -- don't block
            // this thread waiting for the user's response! After the user
            // sees the explanation, try again to request the permission
            new AlertDialog.Builder( context: this)
                    .setTitle("Location Permission Needed")
                    .setMessage("This app needs the Location Permission, please accept to use Location Functionality")
                    .setPositiveButton( text: "OK", (dialogInterface, i) → {
                            // Prompt the user once explanation has been shown
                            ActivityCompat.requestPermissions( activity: MapTour.this,
                                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                                    PERMISSIONS_REQUEST_CODE);
                    })
                    .create()
                    .show();
        }
        else
        {
            // No explanation needed, we can request the permission
            ActivityCompat.requestPermissions( activity: this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    PERMISSIONS_REQUEST_CODE);
        }

}
```

5.2.10 RetrofitInstance.java

```java
/**
 * Create an instance of Retrofit Object
 * This sample uses JACKSON converter
 */

public class RetrofitInstance
{
    //My Machine's IP:Port of Tomcat for the sample
    private static final String Base_URL ="http://192.168.1.82:8080";
    private static Retrofit retrofit;


    public static Retrofit getRetrofitInstance()
    {
        /* For HTTP Error-Logs*/
        HttpLoggingInterceptor interceptor=new HttpLoggingInterceptor();
        interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        OkHttpClient httpClient=new OkHttpClient.Builder().addInterceptor(interceptor).build();

        if(retrofit==null)
        {
            retrofit=new retrofit2.Retrofit.Builder()
                    .baseUrl(Base_URL)
                    .client(httpClient)
                    .addConverterFactory(JacksonConverterFactory.create())
                    .build();
        }
        return retrofit;
    }
}
```

Το Retrofit είναι ένα type-safe HTTP client όπου μετατρέπει ένα HTTP API σε Java Interface.

Η μέθοδος FetchTourData() που είδαμε νωρίτερα κάνει χρήση ενός Retrofit Instance ώστε να κάνει ένα ασύγχρονο request στην Β.Δ.

Στην παραπάνω εικόνα βλέπουμε την δημιουργία της κλάσης Retrofit Instance όπου δημιουργούμε ένα object retrofit ορίζοντας IP και port του τοπικού Η/Υ μας για την σύνδεση με την Β.Δ

Τέλος κάνουμε χρήση και του Jackson Parser για την μετατροπή από JSON σε POJO κλάσεις.

Στις παρακάτω εικόνες δημιουργούμε τις POJO κλάσεις POI και TourName με τα αντίστοιχα getters/setters.

Και ο Jackson κάνει εκτεταμένη χρήση annotations όπως το JsonProperty που δείχνει οτι η μεταβλητή αποτελεί μια JSON οντότητα.

5.2.11 POI.java και TourName.java

```
/** Model Class to map JSON to POJO */

@JsonInclude(JsonInclude.Include.NON_NULL)
public class POI
{

    @JsonProperty("id")
    private int id;

    @JsonProperty("lat")
    private double lat;

    @JsonProperty("lng")
    private double lng;

   @JsonProperty("tourid")
   private int tourid;

   @JsonProperty("tourName")
   private TourName tourName;

   @JsonProperty("tourDescription")
   private String tourDescription;

    public POI(){}

   public POI(int id,double lat,double lng, int tourid, TourName tourName,String tourDescription)
    {
        this.setId(id);
        this.setLat(lat);
        this.setLng(lng);
        this.setTourID(tourid);
        this.setTourName(tourName);
        this.setTourDescription(tourDescription);

    }

    /** Setters */

    private void setTourID(int tourid) { this.tourid=tourid; }
    private void setLng(double lng) { this.lng=lng; }

    private void setLat(double lat) { this.lat =lat; }

    private void setId(int id) { this.id=id; }

    private void setTourName(TourName tourName) { this.tourName= tourName; }
```

```java
/** Model Class to map JSON to POJO */

@JsonInclude(JsonInclude.Include.NON_NULL)
public class TourName
{

    @JsonProperty("id")
    private int id;

    @JsonProperty("tourName")
    private String tourName;

    public int getId() { return id; }

    public TourName(){}

    public TourName(int id,String tourName)
    {
        this.setId(id);
        this.setTourName(tourName);
    }

    /** Setters */
    public void setId(int id) { this.id = id; }

    public void setTourName(String tourName) { this.tourName = tourName; }

    /** Getters */
    public int getID() { return id; }
    public String getTourName() { return tourName; }


}
```

5.2.12 GetDataService.java

```java
/** Interface which will be used to reach the endpoint */
public interface GetDataService
{
    /** Defining URL endpoints */

    /**
     * @return all POIs
     */
    @GET("poi/")
    Call<List<POI>> getAllPOIs();

    /**
     *@return POIs based on TourID
     **/
    @GET("poi/{tourid}")
    Call<List<POI>> getPOIByTourID(@Path("tourid") int tourid);
}
```

Εδώ βλέπουμε την μετατροπή από HTTP API σε Java Interface.

Η κλάση Call αποτελεί ειδικό interface για την κλήση της μεθόδου getPOIByTourID() όπου σε μας υλοποιείται στο FetchTourData().

ΚΕΦΑΛΑΙΟ 6

ΛΕΙΤΟΥΡΓΙΕΣ

Σε αυτήν την ενότητα θα παραθέσουμε screenshots από την εφαρμογή εν λειτουργία και ένα πραγματικό σενάριο χρήστη.

6.1 Διαδρομές

Για την δημιουργία της λίστας των διαδρομών δημιουργήθηκαν 3 διαδρομές που αποτελούν σημαντικά ιστορικά σημεία αλλά και τουριστικά.

Αυτά είναι:

1η Διαδρομή: Πλάκα-Μοναστηράκι περιλαμβάνει:

α) Αρχή: Πλάκα

β) Ναός Ολυμπίου Διός

γ) Μουσείο Ακρόπολης

δ) Ωδείο Ηρώδη του Αττικού

ε) Αρχαία Αγορά

ζ) Θησείο

η) Τέλος: Μοναστηράκι

2η Διαδρομή: Σύνταγμα-Θησείο περιλαμβάνει:

α) Αρχή: Πλατεία Συντάγματος

β) Πλατεία Κολωνακίου

γ) Λυκαβηττός

δ) Ακαδημία Αθηνών

ε) Δημοτική Αγορά

ζ) Αρχαιολογικός Χώρος Κεραμικού

η) Τέλος: Θησείο

3η Διαδρομή: Ακρόπολη-Ζάππειο περιλαμβάνει:

α) Αρχή: Ακρόπολη

β) Αναφιώτικα

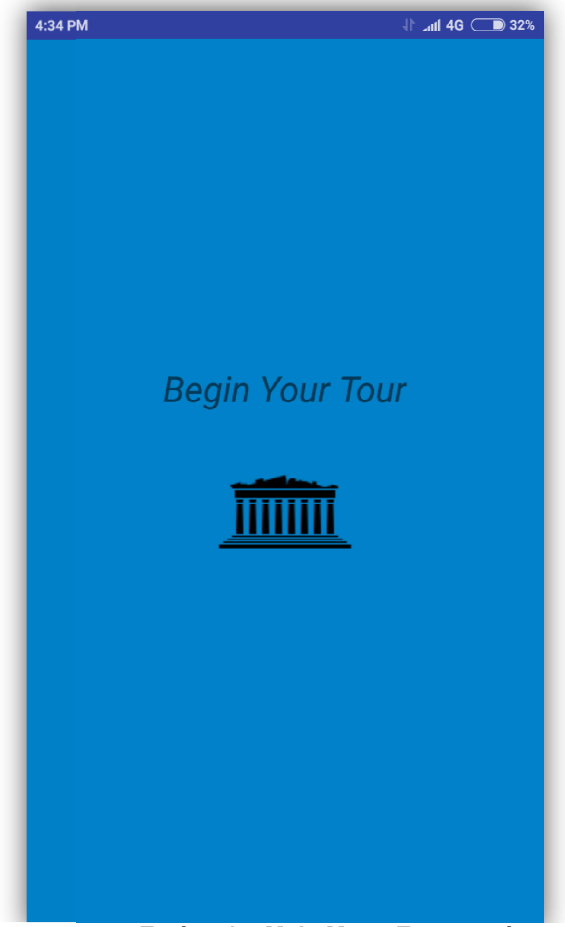
γ) Καλλιμάρμαρο Στάδιο

δ) Πολεμικό Μουσείο

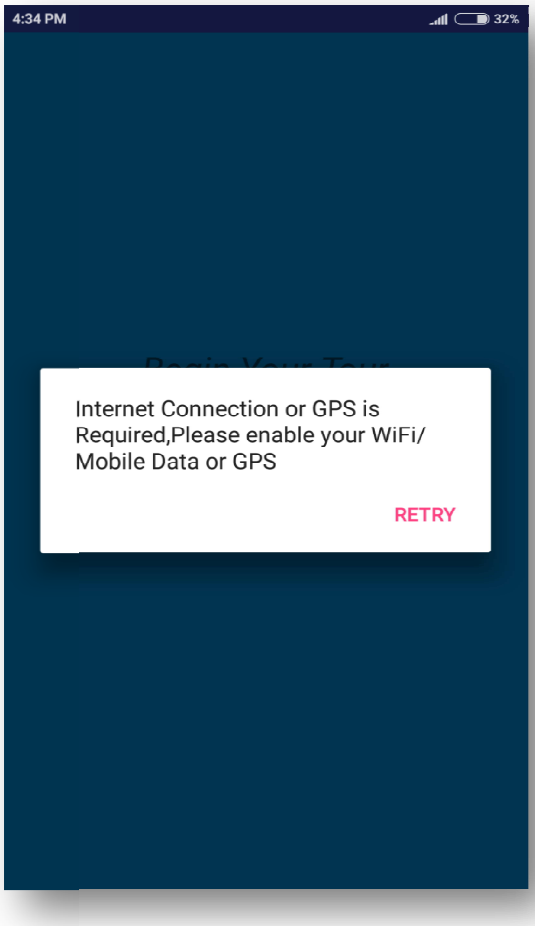
ε) Μουσείο Μπενάκη

ζ) Βοτανικό Μουσείο Εθνικού Κήπου

η) Τέλος: Ζάππειο



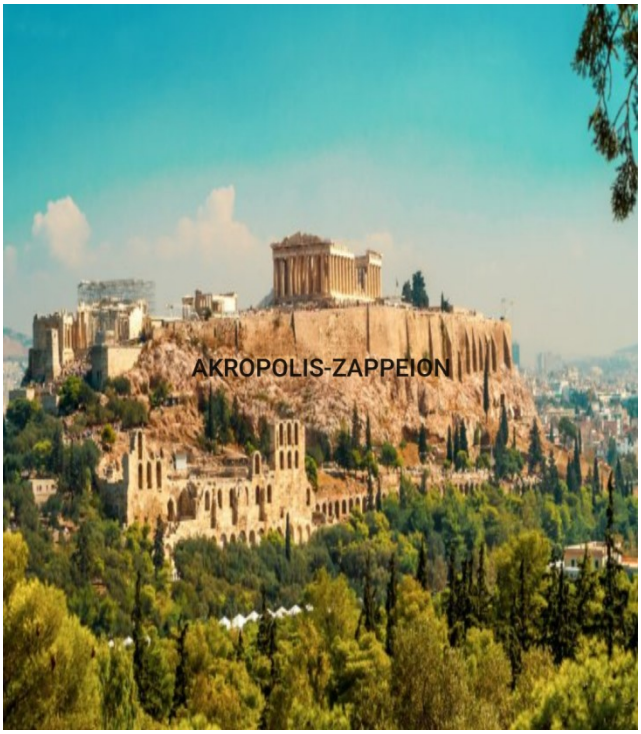
Εικόνα 17. Main Menu Εφαρμογής



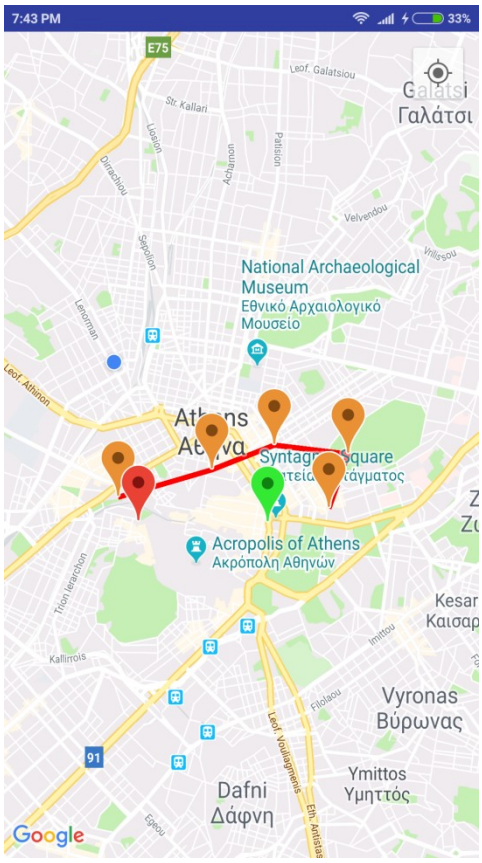
Εικόνα 18. Ενεργοποίηση Σύνδεσης σε περίπτωση αποτυχίας

Εικόνα 19. Image-Button Πλάκα-Μοναστηράκι

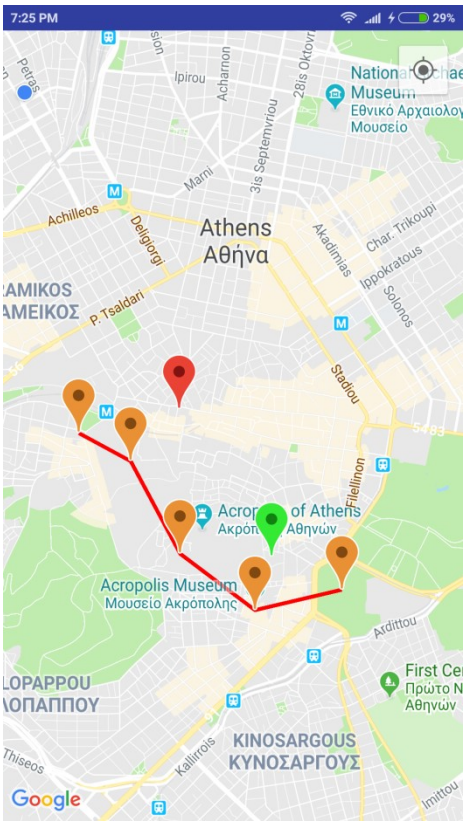
Εικόνα 20. Image-Button Σύνταγμα-Θησείο



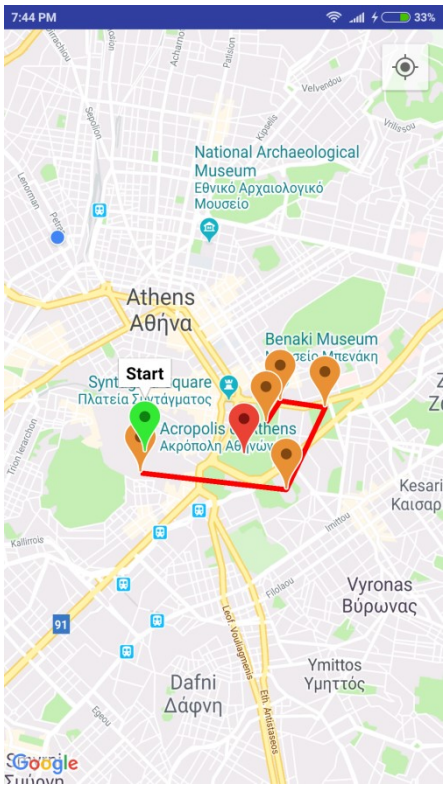
Εικόνα 21. Image-Button Ακρόπολη-Ζάππειο



Εικόνα 22. Διαδρομή Σύνταγμα-Θησείο



Εικόνα 23. Διαδρομή Πλάκα-Μοναστηράκι

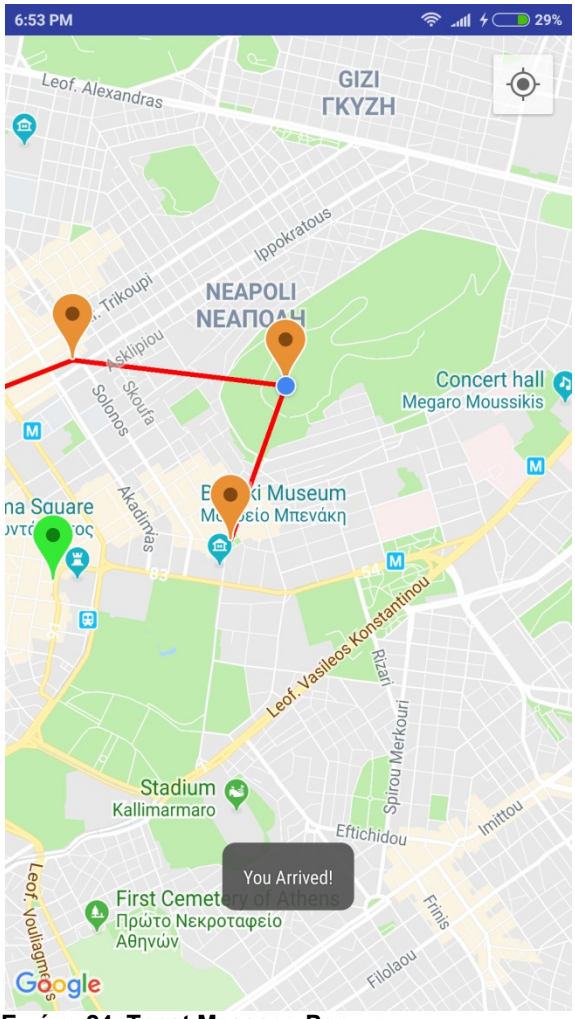


Εικόνα 23. Διαδρομή Ακρόπολη-Ζάππειο

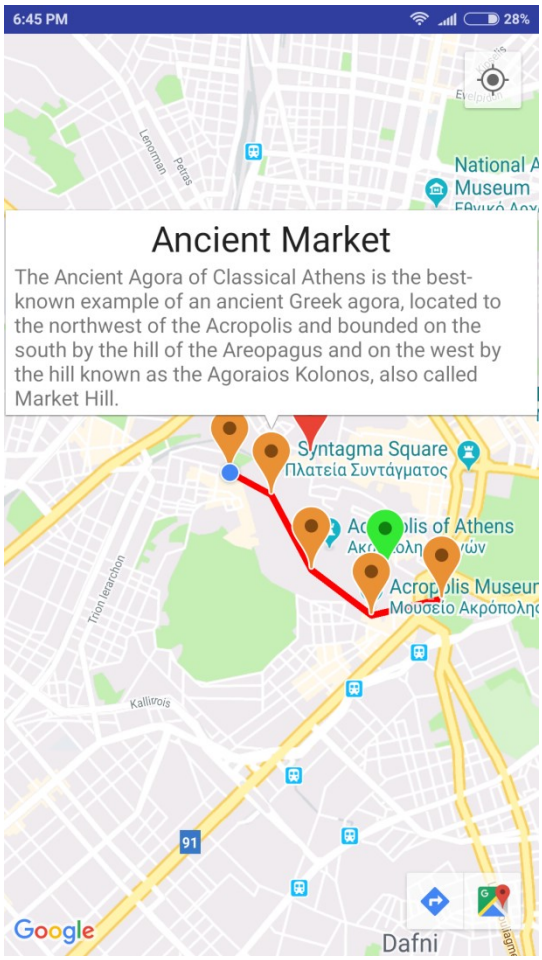
6.2 Testing της εφαρμογής

Για να τεστάρουμε ότι εφαρμογή λειτουργεί σωστά, χρησιμοποιήσαμε ένα άλλο App που ονομάζεται Fake GPS όπου κάνει mock την τοποθεσία μας με τα POI.

Δηλαδή συμπληρώνοντας τις γεωγραφικές συντεταγμένες του POI εμφανιζόμαστε σαν να είμαστε ήδη στο σημείο.



Εικόνα 24. Toast Message Pop-up

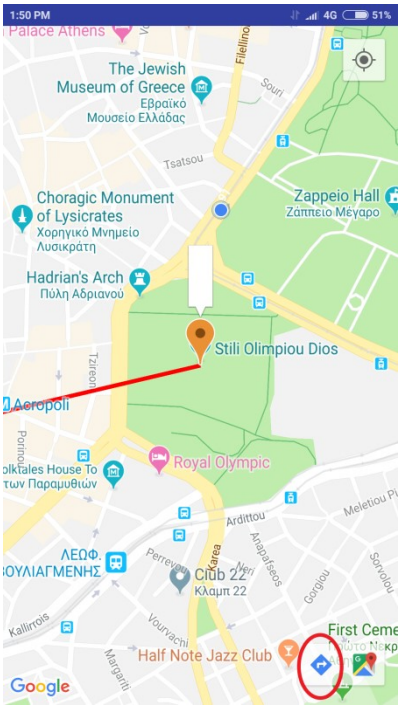


Εικόνα 26. Περιγραφή Αρχαίας Αγοράς

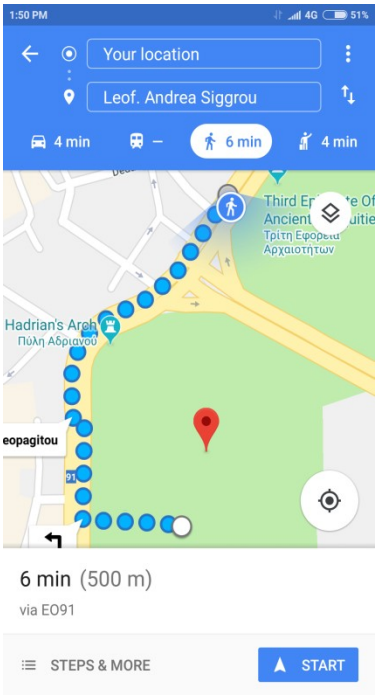
Εικόνα 27. Περιγραφή Πολεμικού Μουσείου

6.2.1 Αληθινό Σενάριο

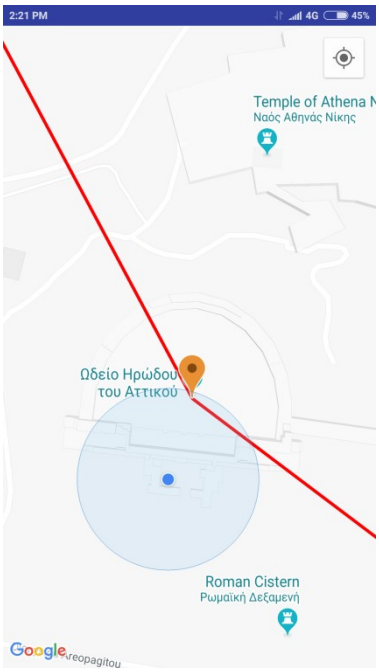
Θα πραγματοποιήσουμε μια δοκιμή όπου διατρέχουμε πραγματικά την 1η διαδρομή. Παρακάτω ακολουθούν ενδεικτικές εικόνες από την περιήγηση. Πατώντας πάνω στα POIs κάτω δεξιά εμφανίζεται ένα κουμπί-βέλος όπου πατώντας το μπορούμε να αντλήσουμε οδηγίες για το πώς να κατευθυνθούμε σε αυτά.



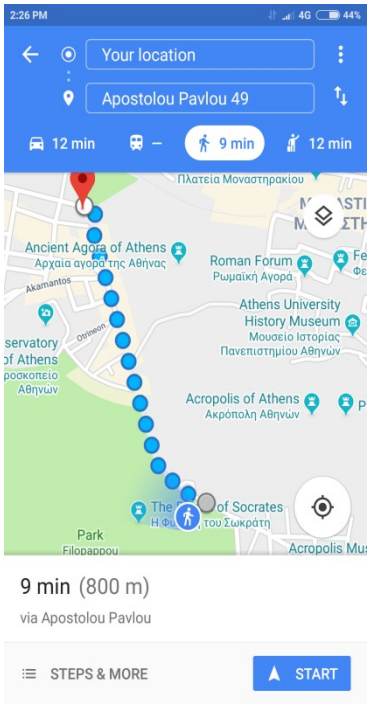
Εικόνα 28. Στήλες Ολυμπίου Διός



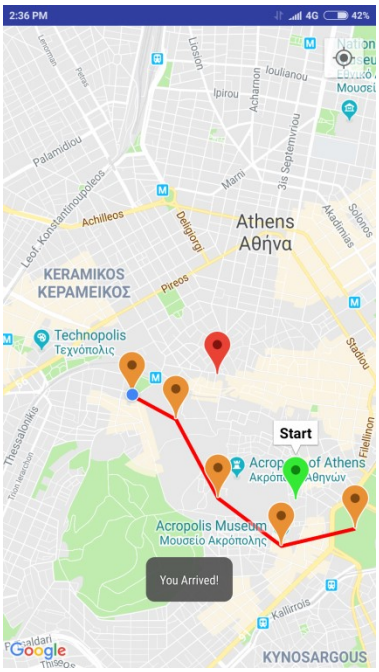
Εικόνα 29. Οδηγίες για κατεύθυνση προς τις Στήλες Ολ.Διός



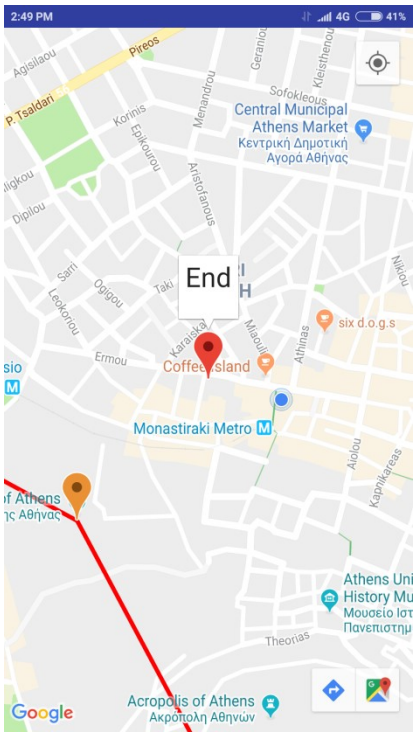
Εικόνα 30. Ωδείο Ηρώδη του Αττικού



Εικόνα 31. Οδηγίες κατεύθυνσης για Αρχαία Αγορά



Εικόνα 32. Θησείο



Εικόνα 33. Τέρμα-Μοναστηράκι

ΚΕΦΑΛΑΙΟ 7

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Ian Sommerville - Βασικές Αρχές Τεχνολογίας Λογισμικού
2. Ευάγγελος Κεχρής - Σχεσιακές Βάσεις Δεδομένων
3. Android Studio Install Instructions: https://developer.android.com/studio
4. Intellij IDEA Install Instructions: https://www.jetbrains.com/idea/download/#section=windows
5. MySQL WorkBench Install Instructions: https://dev.mysql.com/downloads/workbench/
6. Building REST Services with Spring: Tutorial · Building REST services with Spring
7. Spring Boot Documentation: Spring Boot Reference Guide
8. Accessing Data with MySQL: Getting Started · Accessing data with MySQL
9. JPA Reference: 2. JPA Repositories
10. Spring Boot REST API: Building a Spring Boot REST API — Part III: Integrating MySQL Database and JPA
11. JPA/Hibernate-One to one Mapping with MySQL: JPA / Hibernate One to One Mapping Example with Spring Boot | CalliCoder
12. Maps SDK for Android: https://developers.google.com/maps/documentation/android-sdk/intro
13. Custom Info Window: Info Windows | Maps SDK for Android | Google Developers
14. Drawing Routes between Markers: Google Maps Draw Route between two points using Google Directions in Google Map Android API V2
15. Google Maps Integration and Geofence: Implementing Map and Geofence Features in Android* Business Apps | Intel® Software
16. Directions API: https://developers.google.com/maps/documentation/directions/intro
17. Lat/Lng Reference: Get Lat Long from Address Convert Address to Coordinates
18. Android Checking Net Connection: https://www.androidbegin.com/tutorial/android-check-internet-connection/
19. Retrofit Client: Using Retrofit 2.x as REST client - Tutorial
20. Retrofit for Web Services: Using Retrofit 2 for web-services in Android with a simple demo project.
21. Retrofit-Jackson: Retrofit 2, Jackson parser and google map distancematrix api | Codexpedia
22. Android Networking: Android Networking II: OkHttp, Retrofit, Moshi and Picasso
23. Retrofit Capturing JSON Array: https://www.androidtutorialpoint.com/networking/retrofit-android-tutorial/
24. HTTP Calls with Retrofit: How to make HTTP calls on Android with Retrofit 2 – Oleg Šelajev – Medium
25. Android API Reference: https://developer.android.com/reference/
26. Technologies Info: https://en.wikipedia.org/wiki/Main_Page