Τμήμα Μηχανικών
Πληροφορικής ΑΤΕΙΘ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή Android για εύρεση και βαθμολογία παιδότοπων

Του φοιτητή
Μπανάτσα Χρίστου
Αρ. Μητρώου: 123955

Επιβλέπων καθηγητής
Ευκλείδης Κεραμόπουλος

Θεσσαλονίκη 2016-17

ΠΡΟΛΟΓΟΣ

Οι κινητές συσκευές αποτελούν σήμερα αναπόσπαστο κομμάτι της καθημερινότητας μας. Μας διευκολύνουν στην επικοινωνία με άλλους ανθρώπους, στη ψυχαγωγία και πολλές φορές αποτελούν το μέσο στη λύση διαφόρων προβλημάτων. Το κύριο χαρακτηριστικό τους είναι η φορητότητα και το μικρό μέγεθος τους στο οποίο περιλαμβάνονται πολλοί τύποι αισθητήρων και εξαρτημάτων(GPS, γυροσκόπιο, φωτογραφική μηχανή, δακτυλικό αποτύπωμα κ.α). Ο συνδυασμός λοιπόν των δύο παραπάνω χαρακτηριστικών αποτελεί το ιδανικό κρητίριο στη λύση πολλών καθημερινών προβλημάτων. Αυτό έχει σαν αποτέλεσμα, να κεντρίσει το ενδιαφέρον πολλών προγραμματιστών στο να αναπτύξουν τις ιδέες/λύσεις τους σε εφαρμογές για κινητές συσκευές.

Για το παραπάνω λόγο λοιπόν, η εφαρμογή της συγκεκριμένης πτυχιακής εργασίας έχει επιλεγεί να αναπτυχθεί για κινητές συσκευές και πιο συγκεκριμένα για τη πλατφόρμα του Android.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία έχει ως κύριο στόχο την ανάπτυξη μιας εφαρμογής σε περιβάλλον Android. Η εφαρμογή επιτρέπει στον χρήστη να βρίσκει/προσθέτει και να βαθμολογεί παιδότοπους στη περιοχή του καθώς και σε άλλες περιοχές.

Τα βασικά στοιχεία αυτής της πτυχιακής εργασίας επικεντρώνονται σε δύο μέρη. Το πρώτο περιστρέφεται γύρω απο τις λειτουργίες που αφορούν τον εξυπηρετητή(server) της εφαρμογής, εδώ κρατείται η βάση δεδομένων και γίνονται όλοι οι απαραίτητοι υπολογισμοί ώστε το αποτέλεσμα τους να παρουσιαστεί στην συνέχεια στον πελάτη(client), δηλαδή στην Android εφαρμογή. Η Android εφαρμογή αποτελεί και το δεύτερο μέρος, ασχολείται με την αισθητική παρουσίαση των αποτελεσμάτων που θα λάβει από τον εξυπηρετητή και οριοθετεί την ευχρηστία και την εύκολη πρόσβαση μέσω της διεπαφής χρήστη(UI).

Συμπερασματικά λοιπόν πρόκειται για μια εφαρμογή η οποία από άποψη χρηστών στοχεύει γονείς, ώστε να βρούν και να βοηθήσουν και άλλους γονείς να βρούν τους κατάλληλους παιδότοπους για τα παιδιά τους. Ενώ από άποψη τεχνολογιών στοχεύει σε δημοφιλείς νέες τεχνολογίες οι οποίες αποτελούν ένα ενδιαφέρον κομμάτι για μελέτη.

ABSTRACT

The current thesis has as main objective the development of an Android application. The application allows a user to find/add and rate playgrounds in him location as well as in others locations.

The basic elements of this thesis focus in two parts. The first one rotates around the usages of the server, here is held the database and take place all the necessary calculations so that their result be presented into client, the Android application. The second part is about Android application, is concerned with the aesthetics presentation of the server results and limits user-friendliness and easy access via the user interface(UI).

In conclusion, is about an application which from a user's point of view, it aims parents to find and help other parents to find the best playgrounds for their children. In the other side, from a technology point of view, it aims popular and new technologies which they consist an interesting part to study.

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα πτυχιακή εργασία πραγματοποιήθηκε στο Αλεξάνδρειο τεχνολογικό εκπαιδευτικό ίδρυμα Θεσσαλονίκης στο τμήμα Μηχανικών Πληροφορικής κατά το έτος 2016-2017.

Αρχικά, χρωστάω ένα μεγάλο ευχαριστώ στον καθηγητή μου, κ. Ευκλείδη Κεραμόπουλο, που μου έδωσε τη δυνατότητα να μπορέσω να πραγματοποιήσω τη συγκεκριμένη πτυχιακή εργασία καθώς και για τη στήριξη του όλο αυτό το καιρό. Επίσης, θα ήθελα να ευχαριστήσω τον Νεκτάριο Καρανίκα για τις πολύτιμες συμβουλές του στο σχεδιαστικό κομμάτι(UI) της Android εφαρμογής και άλλους φίλους οι οποίοι δέχτηκαν και με βοήθησαν να τεσταριστεί η σταθερότητα της εφαρμογής.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου, Κοτλίδα Βασιλική και Μπανάτσα Δημήτριο οι οποίοι με στήριξαν όλα αυτά τα χρόνια και στους οποίους οφείλω όλη τη διαδρομή των σπουδών μου μέχρι σήμερα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ ΚΑΙ ΠΙΝΑΚΩΝ

Εισαγωγή

Σκοπός της συγκεκριμένης πτυχιακής εργασίας είναι ο σχεδιασμός και η δημιουργία μιας εφαρμογής με τις πιο σύγχρονες τεχνολογίες που χρησιμοποιούνται σήμερα, τόσο στην ανάπτυξη των REST API όσο και στην ανάπτυξη Android εφαρμογών.

Η συγκεκριμένη πτυχιακή εργασία πρόκειται για μία client-server (πελάτης-εξυπηρετητής) εφαρμογή, που αποτελείται από δύο μέρη. Τα πρώτα κεφαλάια(1ο, 2ο, 3ο) αναφέρονται στο κομμάτι του εξυπηρετητή(REST API) και τα υπόλοιπα(4ο, 5ο, 6ο, 7ο) στο κομμάτι του πελάτη(Android εφαρμογή). Και στα δύο μέρη περιγράφονται οι αρχιτεκτονικές και τεχνολογίες που χρησιμοποιήθηκαν για να ολοκληρωθεί αυτή εφαρμογή. Επιπλέον, στο κομμάτι του εξυπηρετητή γίνεται ανάλυση της χρήσης και λειτουργίας του REST API ενώ στο κομμάτι του πελάτη γίνεται ανάλυση της λειτουργίας και χρήσης της εφαρμογής.

Στο τέλος, υπάρχει ένα ειδικό παράρτημα που περιέχει τη δομή της βάσης δεδομένων που χρησιμοποιεί ο εξυπηρετηρής(REST API).

Καλή ανάγνωση!

1. REST API

1.1 Εισαγωγή

Ενα REST(REpresentational State Transfer) API είναι ένας απο τους πιο δημοφιλείς τρόπους για την ανάπτυξη ενός Web API σήμερα. Έχει αντικαταστήσει τα Web services(SOAP) σε πολύ μεγάλο βαθμό καθώς είναι πολύ πιο εύκολο στην υλοποίηση και στη χρήση του μιας και δε στηρίζεται σε κάποιο αυστηρό πρωτόκολλο αλλά σε HTTP ερωτήματα. Στο παρόν κεφάλαιο θα περιγραφεί τι είναι ένα REST API, γιατί προτιμάται και πως λειτουργεί.

1.2 Τι είναι

Δίνωντας έναν γρήγορο ορισμό θα λέγαμε ότι ένα REST API είναι ένα πρόγραμμα(κώδικας) που υπάρχει και εκτελείται σε κάποιο μηχάνημα (εξυπηρετητής) το οποίο χρησιμοποιεί HTTP ερωτήματα(GET, POST, PUT, DELETE) για να τροποποιεί τα δεδομένα του.

1.3 Γιατί REST

Οι παρακάτω αρχές κάνουν εναν REST API να είναι απλό, ελαφρύ και γρήγορο:

- Οι πόροι κάθε REST API προσδιορίζονται απλά μέσω URL's, έτσι ένας πελάτης(client) το μόνο που χρειάζεται να γνωρίζει για να χρησιμοποιήσει ένα Rest API και να έχει πρόσβαση στους πόρους του είναι τα διαθέσιμα URL's του.
- Ομοιόμορφη διασύνδεση, οι πόροι τροποποιούνται μέσω τεσσάρων HTTP λειτουργιων(POST, PUT, GET, DELETE).
- Αυτοπροσδιοριζόμενα μηνύματα, οι πόροι μπορούν να παρουσιαστούν σε πολλές μορφές και όχι μόνο σε μια συγκεκριμένα, έτσι οι ίδιοι πόροι ενός

εξυπηρετητή μπορούν να χρησιμοποιούνται απο πολλούς πελάτες οι οποίοι τους παρουσιάζουν σε άλλη μορφή ο καθένας.

1.4 Πως λειτουργεί

Ο εξυπηρετητής(Server) διαθέτει πόρους(resources) οι οποίοι προσδιορίζονται από τα URL τους. Οι πελάτες(clients) μπορούν να έχουν πρόσβαση σε αυτούς τους πόρους στέλνωντας ένα HTTP ερώτημα(GET, POST, PUT, DELETE) σε κάποιο απο τα διαθέσιμα URL's, η απάντηση που θα πάρουν εξαρτάται από το HTTP ερώτημα που έστειλαν και η μορφή της απάντησης μπορεί να είναι: *text/json/xml* ή και αλλες.

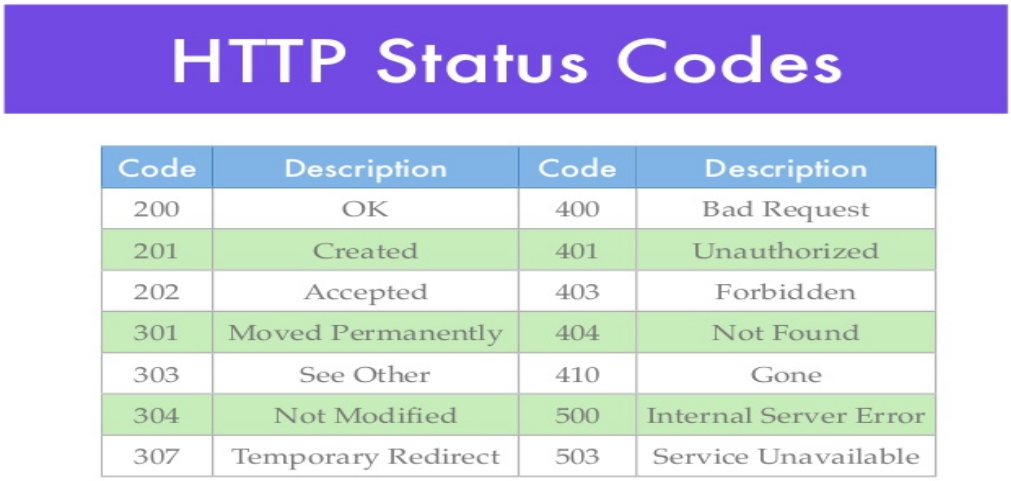
Οι πελάτες στα *POST*, *PUT* HTTP ερωτήματα μπορούν να περιέχουν και κάποιο σώμα/φορτίο(body/payload) εφόσον αυτό απαιτείται απο το εκάστοτε URL. Η μορφή του σώματος/φορτίου καθορίζεται απο τον εξυπηρετητή και μπορεί να είναι: *text/json/xml* ή και αλλες.

Η Χρηση των HTTP ερωτημάτων είναι απλή:

- **POST** ------> Δημιουργία νέου πόρου
- **GET** --------> Ανάκτηση διαθέσιμου πόρου
- **PUT** --------> Ενημέρωση υπάρχωντος πόρου
- **DELETE** --> Διαγραφή πόρου

Σε κάθε ερώτημα που στέλνει ένας πελάτης, για να σιγουρευτεί ότι το ερωτημά του ολοκληρώθηκε επιτυχώς ή όχι, στην απάντηση που θα λάβει απο τον εξυπηρετητή θα υπάρχει πάντα ένα πεδίο *STATUS* το οποίο θα περιέχει ένα HTTP status code.

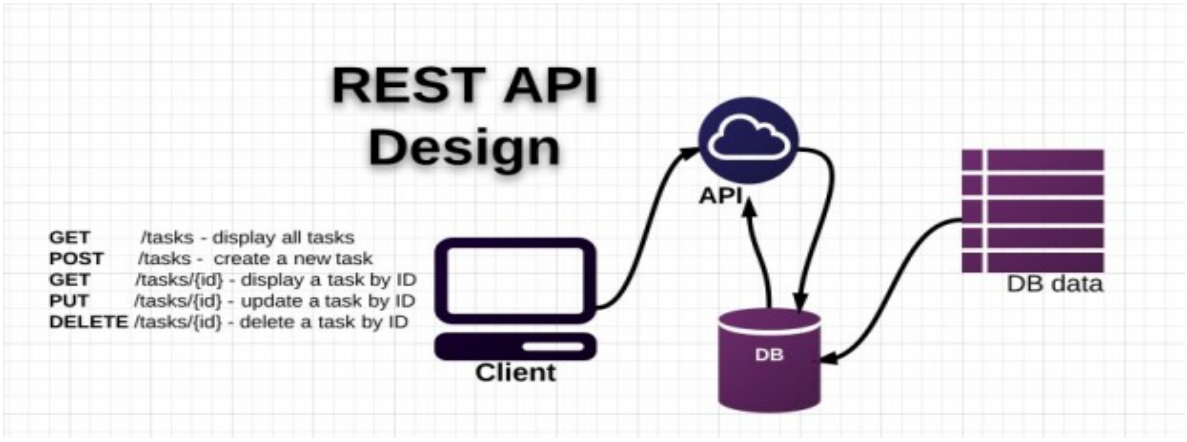
Τα πιο συνήθη HTTP status code και η ερμηνεία τους παρουσιάζονται στη παρακάτω εικόνα



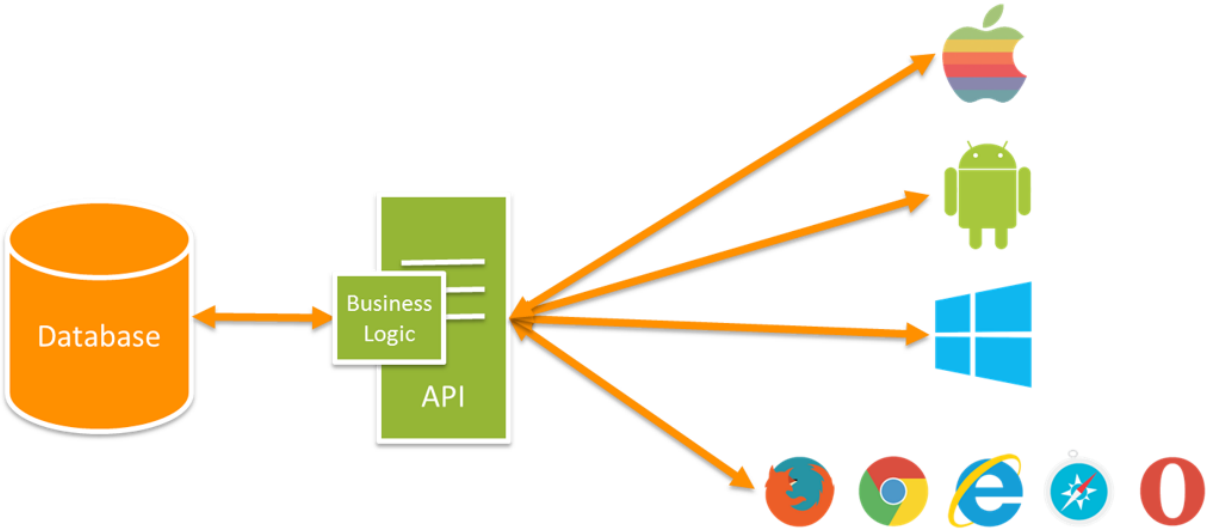
Εικόνα 1: HTTP status codes

1.5 Επίλογος

Στο παρών κεφάλαιο είδαμε τι είναι, πως λειτουργεί και γιατί προτειμάται ένα REST API. Για την καλύτερη κατανόηση ακουληθούν δύο εικόνες οι οποίες παρουσιάζουν ένα ολοκληρωμένο σύστημα βασισμένο σε ένα REST API.



Εικόνα 2: Ολοκληρωμένο σύστημα με REST API 1



Εικόνα 3: Ολοκληρωμένο σύστημα με REST API 2

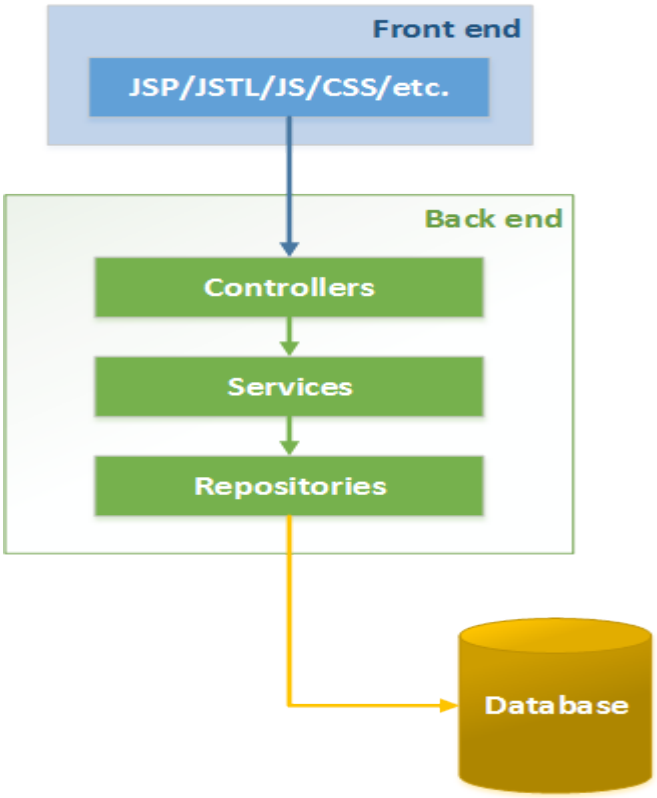
2. Αρχιτεκτονική και τεχνολογίες που χρησιμοποιήθηκαν

2.1 Εισαγωγή

Στο παρόν κεφάλαιο θα περιγραφεί η αρχιτεκτονική που χρησιμοποιήθηκε για την υλοποίηση του REST API καθώς και τι τεχνολογίες χρησιμοποιήθηκαν προκειμένου να γίνει κατανοητό στον αναγνώστη τόσο το θεωρητικό όσο και το πρακτικό μέρος.

2.2 Αρχιτεκτονική

Η αρχιτεκτονική που ακολουθήθηκε για την δημιουργία του REST API είναι όπως αυτή φένεται στην εικόνα 3 (τα ορθογώνια μέσα στο *Back end* σχήμα)



Εικόνα 4: Αρχιτεκτονική του Rest API

Όπως βλέπουμε έχουμε μια αρχιτεκτονική τριών επιπέδων:

1. Ο *Controller* είναι αυτός που περιέχει όλα τα διαθέσιμα URL που μπορεί να εξυπηρετήσει το API μας και διαχειρίζεται τα ερωτήματα των πελατών.
2. Το *Service* αναλαμβάνει να υλοποιήσει το Business logic του API.
3. Το *Repository* αναλαμβάνει να υλοποιήσει όλες τις συναλλαγές με τη βάση δεδομένων που μπορεί να προκύψουν στο Business logic.

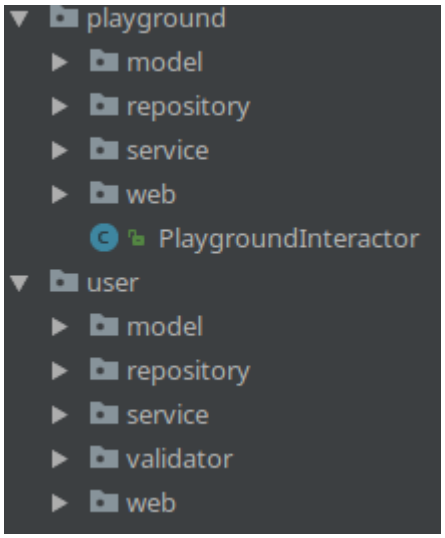
Η επικοινωνία και η εξάρτηση αυτών των τριών στρωμάτων γίνεται από έξω προς τα μέσα και είναι αυστηρά αυτή και μόνο αυτή. Δηλαδή, ο Controller καλέι το Service το οποίο με τη σειρά του καλεί το Repository. Πιο συγκεκριμένα, όταν ένας πελάτης στέλνει ένα ερώτημα στο API, συμβαίνουν τα εξείς:
Το ερώτημα αυτό το διαχειρίζεται ο Controller, εάν μπορεί να το χειριστεί θα καλέσει το Service(διαφορετικά θα ενημερώσει το πελάτη ότι το συγκεκριμένο URL δε βρέθηκε) ώστε να εφαρμόσει τις αλλαγές στο business logic το οποίο Service θα καλέσει με τη σειρά του το Repository για να ενημερώσει τη βάση δεδομένων με τυχόν αλλάγες που προκύψουν. Εφόσον ολοκληρωθούν όλα επιτυχώς, ο Controller απαντά στον πελάτη για το ερώτημα που έστειλε.

2.2.1 Συμβουλή

Μια εύκολη και απλή αρχιτεκτονική περιλαμβάνει έναν Controller ένα Service και ένα Repository. Συνήθως όταν το business logic που πρόκειται να υλοποιεί το API μας είναι μικρό τότε ο παραπάνω αριθμός είναι εντάξει. Όταν όμως θέλουμε να υλοποιήσουμε κάτι μεγάλο και περίπλοκο θα ήταν προτιμότερο και πιο ευέλικτο, για κάθε χαρακτηριστικό(feature) της εφαρμογής να υπάρχει και η ανάλογη τριάδα CSR(controller-service-repository) που θα είναι υπεύθυνη για αυτό και μόνο αυτό το χαρακτηριστικό.

Για παράδειγμα, αν η εφαρμογή μας περιέχει χρήστες και αυτοκίνητα θα μπορούσε να υπάρχει μια τριάδα CSR για την εξυπηρέτηση όλων των ερωτημάτων που θα γίνονται για τους χρήστες και μια άλλη για την εξυπηρέτηση των ερωτημάτων για τα αυτοκίνητα.

Παρακάτω ακολουθεί μια εικόνα που απεικονίζει τη δομή των φακέλων του REST API της παρούσας πτυχιακής εργασίας. Όπως θα παρατηρήσει κανείς θα δει πως και εδώ χρησιμοποιήθηκαν δύο τριάδες CSR, μία που αφορά τους παιδότοπους και μία που αφορά τους χρήστες (ο φάκελος Web αφορά τον Controller).



Εικόνα 5: Δομή φακέλων REST API

2.3 Τεχνολογίες που χρησιμοποιήθηκαν

2.3.1 Gradle

Είναι ένα ανοιχτού κώδικα εργαλείο, το οποίο έχει ως σκοπό την αυτοματοποιημένη δημιουργία (build automation) και τη διαχείρηση εξαρτήσεων ενός λογισμικού προγράμματος. Είναι βασισμένο στο Maven [1] και η σύνταξη του γίνεται σε Groovy [2].

2.3.1.1 Αυτοματοποιημένη δημιουργία

Για να χρησιμοποιηθεί αρκεί να γίνει η εγκατάσταση του [3] στον τοπικό υπολογιστή και στη συνέχεια να δημουργηθεί ένα *build.gradle* αρχείο μέσα στον ριζικό φάκελο(root folder) του προγράμματος μας στο οποίο θέλουμε να το χρησιμοποιήσουμε. Ολες οι ρυθμίσεις του προγράμματος μπορούν να γίνουν μέσα σε αυτό το αρχείο χρησιμοποιώντας τα κατάλληλα πεδία.

Για παράδειγμα, σε ένα πρόγραμμα java οπου τα πηγιαία αρχεία βρίκονται στον φάκελο ***/src/java,*** δημιουργούμε ένα αρχειο: ***build.gradle*** το οποίο περιέχει:

apply plugin: 'java'
sourceSets.main.java.srcDirs = ['src/java']

Εάν ανοίξουμε ένα τερματικό παράθυρο, μεταφερθούμε στο κατάλογο όπου βρίσκεται το build.gradle αρχείο που δημιουργήσαμε παραπάνω και δώσουμε την εντολή:

gradle build

θα δημιουργηθεί το εκτελέσιμο αρχείο του προγράμματος. Οι διαθέσιμες εντολές και εργασίες του Gradle είναι πολλές αλλά είναι εκτός των ορίων αυτής της πτυχιακής.

2.3.1.2 Διαχειριστής εξαρτήσεων

Όπως είπαμε το Gradle είναι και ένας διαχειριστής εξαρτήσεων, οπότε δεν χρειάζεται όταν θελουμε να χρησιμοποιήσουμε μια βιβλιοθήκη λογισμικού να τη κατεβάσουμε και να προσθέσουμε το jar αρχείο της στο πρόγραμμα μας. Μπορεί το Gradle να το κάνει για μας αρκεί να του ορίσουμε ποια βιβλιοθήκη επιθυμούμε και από που θα τη βρει. Για παράδειγμα, έστω ότι θέλουμε να προσθέσουμε την *commons-io* βιβλιοθήκη του Apache, αρκεί να πάμε στο build.gradle αρχείο και να προσθέσουμε τις εξής γραμμές:

```
repositories {
  // που να ψάξει για να βρει την εξάρτηση
   mavenCentral()
}

dependencies {
   compile 'commons-io:commons-io:2.5'
}
```

Προγράμματα τα οποία είναι ήδη χτισμένα με Gradle περιλαμβάνουν έναν *Gradle Wrapper* ο οποίος μπορεί να χρησιμοποιηθεί για να εκτελεστεί οποιαδήποτε λειτουργία του Gradle ακόμη και αν αυτό δεν υπάρχει εγκατεστημένο στον τοπικό υπολογιστή.
Η χρήση του στο τερματικό είναι η ίδια με αυτή που περιγράψαμε παραπάνω μόνο που αντί για την εντολή *gradle* θα πρέπει να δίνουμε την εντολή: *gradlew.*

2.3.2 JSON

Το JSON(JavaScript Object Notation) ειναι βασισμένο σε ένα υποσύνολο της γλώσσα προγραμματισμού JavaScript και είναι ένα πρότυπο κειμένου που χρησιμοποιείται για αποθήκευση και ανταλλαγή δεδομένων. Η πιο συνήθης χρήση του σήμερα είναι η μετάδοση δεδομένων ανάμεσα σε εφαρμογές και Rest API's. Είναι ελαφρύ, απλό, εύκολα κατανοητό απο τους ανθρώπους και δεν εξαρτάται απο καμιά γλώσσα προγραμματισμού, πράγματα που το έχουν κάνει ευρέως διαδεδομένο.

2.3.2.1 Μορφές JSON

Αποτελείται από δύο τιμές και η σύνταξη του είναι όπως η παρακάτω:

```
{
   "ονομα" : "τιμή"
}
```

Σαν όνομα μπορεί να είναι οποιοδήποτε αλφαριθμητικό και σαν τιμή υποστηρίζει τους παρακάτω τύπους:

- Αλφαριθμητικό(String), παράδειγμα:

  ```
  {
        "name" : "Christos"
  }
  ```

- Λογικές τιμές(true-false), παράδειγμα:

```
{
    "student" : true
}
```

- Αριθμός, παράδειγμα:

```
{
    "age" : 23
}
```

- Πίνακας(Array), παράδειγμα:

```
{
    "coordinates" : [ 20.45, 10.22 ]
}
```

- Αντικείμενο(object): ένα ή περισσότερα json από τους παραπάνω τύπους τα οποία διαχωρίζονται μεταξύ τους με το χαρακτήρα ',' (κόμα), παράδειγμα:

```
{
    "links" : {
        "rel" : "self",
        "site" : "http://google.gr"
    }
}
```

- Κενό(null), παράδειγμα:

```
{
    "color" : null
}
```

2.3.2.2 JSON Parser

Είπαμε πως το JSON χρησιμοποιείται στις ανταλλαγές μηνυμάτων και ότι είναι ανεξάρτητο απο τη γλώσσα προγραμματισμού, αυτό σημαίνει πως θα πρέπει με κάποιο τρόπο ο αποστολέας και ο παραλήπτης να καταλαβαίνουν το περιεχόμενο τους. Το πρόβλημα αυτό έρχεται να λύσει ο JSON Parser ο οποίος έχει σαν ρόλο να δημιουργεί απο ένα αντικείμενο μιας γλώσσας προγραμματισμού(πχ POJO) το αντίστοιχο JSON αντικείμενο και το αντίστροφο.
Τέτοιοι parsers είναι ο *Gson* και ο *Jackson* οι οποίοι είναι και οι πιο διαδεδομένοι και ευρέως γνωστοί.

2.3.3 MongoDB

2.3.3.1 NoSQL

Η MongoDB είναι μια δωρεάν και ανοικτού κώδικα βάση δεδομένων που ανήκει στη κατηγορία των NoSQL βάσεων δεδομένων. Αναπτύσεται από τη MongoDB Inc. και η τελευταία έκδοση της είναι η 3.4.
Με τον όρο NoSQL εννοούνται εκείνες οι βάσεις δεδομένων που διαθέτουν έναν μηχανισμό για αποθήκευση και ανάκτηση δεδομένων διαφορετικό απο εκείνο των σχεσιακών βάσεων (SQL). Οι τύποι δεδομένων που χρησιμοποιούν οι NoSQL βάσεις (κλειδί-τιμή, γράφοι, έγγραφα κ.α) είναι διαφορετικοί απο εκείνους των σχεσιακών βάσεων κάτι που τις καθιστά πιο αποδοτικές σε διάφορες λειτουργίες.

2.3.3.2 Έγγραφο(Document)

Μια εγγραφή στη MongoDB είναι ένα Έγγραφο(Document). Έγγραφο είναι μια δομή δεδομένων αποτελούμενη απο πεδία και ζεύγη τιμών. Είναι παρόμοια με ένα JSON αντικείμενο και οι τιμές των πεδίων τους μπορούν να περιέχουν: όλους τους υποστηριζόμενους JSON τύπους, άλλα ‘Εγγραφα, πίνακες ή πίνακες Εγγράφων κάθως και κάποιους επιπλέον τύπος όπως Date, Timestamps, ObjectId.

Ένα παράδειγμα ενός Εγγράφου είναι το παρακάτω:

```
{
   _id: ObjectId("5099803df3f4948bd2f98391"),  <--- πεδίο: τιμή
   name: "Chris",                               <--- πεδίο: τιμή
   age: 23,                                     <--- πεδίο: τιμή
   birth: new Date('Feb 08, 1994'),             <--- πεδίο: τιμή
   status: "A",                                 <--- πεδίο: τιμή
   groups: ["student", "admin"]                 <--- πεδίο: τιμή
}
```

2.3.3.2.1 Περιορισμοί πεδίων

Το όνομα που μπορούμε να δώσουμε σε ένα πεδίο μπορεί να είναι οποιοδήποτε αλφαριθμητικό με κάποιους ελάχιστους περιορισμούς:

- Το πεδίο *_id* είναι το πεδίο που δημιουργεί αυτόματα η MongoDB για κάθε νέο Έγγραφο και χρησιμοποιείται ως κύριο μοναδικό κλειδί. Οπότε είναι δεσμευμένο να χρησιμοποιείται για συγκεκριμένο σκοπό.
- Το όνομα των πεδίων δεν μπορούν να ξεκινάν με το χαρακτήρα *$* (δολάριο).
- Το ονόμα των πεδίων δεν μπορούν να περιέχουν το χαρακτήρα *.* (τελεία).
- Το ονόμα των πεδίων δεν μπορούν να περιέχουν το χαρακτήρα *null*.

2.3.3.2.2 _id πεδίο

Οπως αναφέραμε και παραπάνω στη MongoDB κάθε Έγγραφο απαιτεί ένα μοναδικό *_id* πεδίο που χρησιμοποιείται σαν κύριο κλειδί. Αν σε κάποιο Έγγραφο εισάγουμε εμείς το πεδίο *_id* τότε η MongoDB θα δημιουργήσει αυτόματα ένα *ObjectId* αντικείμενο γι’ αυτό. Το *_id* μπορεί να είναι οποιοσδήποτε απο τους διαθέσιμους τύπους δεδομένων εκτός απο πίνακας. Εάν δε το ορίσουμε εμείς η MongoDB σαν προεπιλεγμένο τύπο του *_id* ορίζει το αλφαριθμητικό(String).

2.3.3.2.3 Πλεονεκτήματα

Τα πλεονεκτήματα ενος εγγράφου είναι τα παρακάτω:

- Τα Έγγραφα μπορούν να αναπαρασταθούν εύκολα σε κάθε γλώσσα προγραμματισμού με τους τύπους δεδομένων που διαθέτει η κάθε μια.
- Ενσωματωμένα Έγγραφα και πίνακες μειώνουν την ανάγκη για περίπλοκες συνενώσεις(expensive joins).
- Τα Έγγραφα δεν έχουν στατική μορφή και μπορούν να αλλάξουν οποιαδήποτε στιγμή δυναμικά, υποστηρίζοντας έτσι ευέλικτα τον πολυμορφισμό και δίνωντας μια ευελιξία στο προγραμματιστή για τη δομή της βάσης δεδομένων.

2.3.3.3 Mongo κέλυφος(Shell)

Εγκαθιστώντας [5] τοπικά τη MongoDB εγκαθίστατε και ένα κέφυλος, μέσα απο το οποία μπορούμε να εκτελέσουμε διάφορα ερωτήματα. Για να ανοίξουμε ένα mongo κέλυφος αρκεί σε ένα παράθυρο γραμμής εντολών να πληκτρολογίσουμε:

mongo

Εφόσον συνδεθούμε στο κέλυφος μπορούμε να δούμε όλες τις διαθέσιμες βάσεις δεδομένων που υπάρχουν πληκτρολογώντας:

show databases

Για να δημιουργήσουμε μια νέα αρκεί να εισάγουμε το παρακάτω:

use db_name

Η εντολή *use <db>* αυτό που κάνει είναι να επιλέξει τη συγκεκριμένη βάση ώς τρέχον βάση, σε περίπτωση που αυτή δεν υπάρχει τη δημιουργεί.

2.3.3.4 Συλλογή (Collection)

Στη MongoDB ένα σύνολο εγγράφων αποτελούν μια Συλλογή(Collection) η οποία σε αντιστοίχηση με τις σχεσιακές βάσεις δεδομένων μπορούμε να πούμε ότι είναι ένας πίνακας(table).

Για να δημιουργήσουμε μια νέα Συλλογή εισάγουμε στο mongo κέλυφος(αφού πρώτα έχουμε επιλέξει μια βάση, διαφορετικά θα χρησιμοποιηθεί η test), το παρακάτω:

db.myNewCollection.insertOne({ name: "John" })

Το όνομα της Συλλογής που δημιουργήθηκε με τη παραπάνω εντολή είναι το: *myNewCollection*. Παρατηρούμε ότι πρέπει να αποθηκεύσουμε ένα Έγγραφο στη Συλλογη για να δημιουργηθεί, στη παραπάνω περίπτωση αυτό πετυχαίνεται με την μέθοδο i*nsertOne()*, διατίθεται και η *insertMany()* η οποία μας επιτρέπει να εισάγουμε παραπάνω απο ένα Έγγραφα σε μια εντολή. Μπορούμε να δούμε όλες τις διαθέσιμες Συλλογές που υπάρχουν σε μια MongoDB βάση δίνωντας τη παρακάτω εντολή στο κέλυφος:

show collections

Μπορούμε να δούμε όλα τα Έγγραφα που περιέχει μια Συλλογή δίνωντας:

db.myNewCollection.find().pretty()

Η μέθοδος *find()* αυτό που κάνει είναι να εκτελεί ένα ερώτημα και να βρίσκει Έγγραφα με συγκεκριμένα κριτήρια, τα οποία περνούνται σαν παράμετροι. Οταν δεν υπάρχουν παράμετροι επιστρέφει όλα τα Έγγραφα απο μια συλλογή.

Η μέθοδος *pretty()* παρουσιάζει το αποτέλεσμα του ερωτήματος πιο ευανάγνωστο στην οθόνη.

2.3.3.5 Ερωτήματα (Queries)

2.3.3.5.1 Τελεστές ερωτημάτων

Η MongoDB περιέχει ένα σύνολο απο τελεστές που βοηθάνε στην εκτέλεση ερωτημάτων. Κάποιοι από τους πιο γνωστούς και χρησιμοποιήσιμους είναι:

- *$eq* : Αντιστοιχεί τις τιμές που είναι ίδιες με μια καθορισμένη τιμή.
- *$gt* : Αντιστοιχεί τις τιμές που είναι μεγαλύτερες απο μια καθορισμένη τιμή.
- *$gte* : Αντιστοιχεί τις τιμές που είναι μεγαλύτερες-ίσες απο μια καθορισμένη τιμή.
- *$lt* : Αντιστοιχεί τις τιμές που είναι μικρότερες απο μια καθορισμένη τιμή.
- *$lte* : Αντιστοιχεί τις τιμές που είναι μικρότερες-ίσες απο μια καθορισμένη τιμή.
- *$ne* : Αντιστοιχεί τις τιμές που δεν είναι ίδιες με μια καθορισμένη τιμή.
- *$in* : Αντιστοιχεί τις τιμές που βρίσκονται σε έναν πίνακα.
- *$nin* : Αντιστοιχεί τις τιμές που δεν βρίσκονται σε έναν πίνακα.
- *$or* : Αυνενώνει ένα κριτήριο με ένα άλλο και επιστρει τα Έγγραφα που ταιριάζουν σε ένα απο τα 2 κριτήρια.
- *$and* :Σσυνενώνει ένα κριτήριο με ένα άλλο και επιστρέφει τα Έγγραφα που ταιριάζουν και στις 2 συνθήκες.
- *$not* : Αντιστρέφει το ερώτημα και επιστρέφει Έγγραφα που δεν ταιριάζουν στα κριτήρια.
- *$exists* : Αντιστοιχεί Έγγραφα που έχουν τα καθορισμένα πεδία.
- *$all* : Αντιστοιχεί πίνακες που περιέχουν όλα τα στοιχεία του ερωτήματος.
- *$size* : Επιλέγει τα Έγγραφα που το μέγεθος του πεδίου του πίνακα τους έχει μια συγκεκριμένη τιμή.

2.3.3.5.2 Παραδείγματα

Ακολουθούν κάποια παραδείγματα ερωτημάτων.

Έστω ότι έχουμε μια Συλλογή ‘inventory’ που περιέχει τα εξής Έγγραφα:

```
db.inventory.insertMany([
  { item: "journal", qty: 25, tags: [ "blank", "red" ], dim_cm: [ 14, 21 ] },
  { item: "notebook", qty: 50, tags: [ "red", "blank" ], dim_cm: [ 14, 21 ] },
  { item: "paper", qty: 100, tags: [ "red", "blank", "plain" ], dim_cm: [ 14, 21 ] },
  { item: "planner", qty: 75, tags: [ "blank", "red" ], dim_cm: [ 22.85, 30 ] },
  { item: "postcard", qty: 45, tags: [ "blue" ], dim_cm: [ 10, 15.25 ] }
]);
```

1. Να βρεθούν όλα τα Έγγραφα που έχουν ένα πεδίο *tags*, το οποίο είναι ένας πίνακας, με ακριβώς δύο στοιχεία “red”, “black” τοποθετημένα στο πίνακα με αυτή τη σειρά(1ο το “red”, 2ο το “black”):

   ```
   db.inventory.find( { tags: ["red", "blank"] } )
   ```

2. Να βρεθούν τα Έγγραφα που τα στοιχεία του πίνακα tags είναι τα “red”, “black” αλλά όχι τοποθετημένα σε συγκεκριμένη σειρά το καθένα:

   ```
   db.inventory.find( { tags: { $all: [ "red", "blank"] } } )
   ```

3. Να βρεθούν τα Έγγραφα που στον πίνακα *tags* περιέχουν το “red” ως ένα από τα στοιχεία τους:

   ```
   db.inventory.find( { tags: "red" } )
   ```

4. Να βρεθούν τα Έγγραφα που το πεδίο *dim_cm*(πίνακας) περιέχει τουλάχιστον ένα στοιχείο με τιμή μεγαλύτερη του 25:

   ```
   db.inventory.find( { dim_cm: { $gt: 25 } } )
   ```

5. Να βρεθούν τα Έγγραφα που το πεδίο *tags*(πίνακας) έχει ακριβώς 3 στοιχεία:
   db.inventory.find({ "tags": { $size: 3 } })

6. Να βρεθούν τα Έγγραφα που το δεύτερο στοιχείο του πίνακα *dim_cm* είναι μεγαλύτερο απο 25:
   Χρησιμοποιώντας το χαρακτήρα . (τελεία) μπορούμε να αναφερθούμε σε ένα συγκεκριμένο στοιχείο του πίνακα(θέση ή όνομα). Οι θέσεις των στοιχείων στους πίνακες ξεκινάνε απο το μηδέν. Οπότε το ερώτημα μας θα είναι:
   db.inventory.find({ "dim_cm.1": { $gt: 25 } })

2.3.3.6 Ενημέρωση (Update)

2.3.3.6.1 Τελεστές ενημέρωσης

Εκτός από τους τελεστές ερωτημάτων η MongoDB διαθέτει και τελεστές ενημερώσεων. Κάποιοι από τους πιο γνωστούς και χρησιμοποιήσιμους είναι:

- *$inc* : Αύξηση της τιμής του πεδίου κατά συγκεκριμένο ποσό.
- *$mu*l : Πολλαπλασιασμός της τιμής του πεδίου κατά συγκεκριμένο ποσό.
- *$rename* : Μετονομασία ενός πεδίου.
- *$set* : Ορισμός της τιμής ενός πεδίο σε ένα Έγγραφο.
- *$unset* : Διαγραφή πεδίου απο ένα Έγγραφο.
- *$min* : Ενημέρωση του πεδίου εάν η καθορισμένη τιμή είναι μικρότερη από την υπάρχων.
- *$max* : Ενημέρωση του πεδίου εάν η καθορισμένη τιμή είναι μεγαλύτερη από την υπάρχων.
- *$currentDate* : Ορίζει την τιμή ενός πεδίου στην τρέχον ημερομηνία.
- *$addToSet :* Προσθήκη στοιχείων σε έναν πίνακα μόνο αν δεν υπάρχουν ήδη.
- *$pop* : Διαγραφή του πρώτου ή του τελευταίου στοιχείου ενός πίνακα.
- *$pull* : Διαγραφή όλων των στοιχείων ενός πίνακα που ταιριάζουν σε ένα συγκεκριμένο ερώτημα.

- *$push* : Προσθήκη ενός στοιχείου σε έναν πίνακα.

2.3.3.6.2 Παραδείγματα

Ακολουθούν κάποια παραδείγματα ενημερώσεων.

Έστω ότι έχουμε μια Συλλογή “orders” η οποία περιέχει τα εξής Έγγραφα:

```
db.orders.insertMany( [
  { item: "canvas", qty: 100, sales: { gr: 28, fr: 35, provider: "cm" }, status: "A" },
  { item: "journal", qty: 25, sales: { gr: 14, fr: 21, provider: "cm" }, status: "A" },
  { item: "mat", qty: 85, sales: { gr: 27, fr: 35, provider: "cm" }, status: "A" },
  { item: "mousepad", qty: 25, sales: { gr: 19, fr: 22, provider: "cm" }, status: "B" },
  { item: "notebook", qty: 50, sales: { gr: 8, fr: 11, provider: "in" }, status: "B" },
  { item: "paper", qty: 100, sales: { gr: 8, fr: 11, provider: "in" }, status: "C" },
  { item: "planner", qty: 75, sales: { gr: 22, fr: 30, provider: "cm" }, status: "C" },
  { item: "postcard", qty: 45, sales: { gr: 10, fr: 15, provider: "cm" }, status: "A" },
  { item: "sketchbook", qty: 80, sales: { gr: 14, fr: 21, provider: "cm" }, status: "A" },
  { item: "sketch pad", qty: 95, sales: { gr: 22., fr: 30, provider: "cm" }, status: "A" }
]);
```

1. Αντικατάσταση του πεδίου provider σε *‘jk’* και του πεδίου *status* σε *‘B’* στο (πρώτο που θα βρεθεί) Έγγραφο με *item : "paper"* και προσθήκη ενός νέου πεδίου *lastModified* που θα έχει σαν τιμή την τρέχον ημερομηνία:

   Για να ενημερώσουμε ένα Έγγραφο μπορούμε να χρησιμοποιήσουμε την μεθοδο *updateOne()*, η οποία θα ενημέρωσει το πρώτο Έγγραφο που θα βρει να ταιριάζει στα κριτήρια που θα περιέχει.

   Η δομή της *updateOne()* αποτελείται απο 2 μέρη, τα κριτήρια στα οποία θα χρησιμοποιήσει για να βρει το πρώτο Έγγραφο στο οποίο θα εφαρμοσει τις αλλαγές και τις αλλαγές που θα εφαρμόσει στα πεδία:

```
db.inventory.updateOne(
        { item: "paper" },
        {
                $set: { "sales.provider": "jk", status: "B" },
                $currentDate: { lastModified: true }
        }
)
```

Με το *lastModified: true* λέμε στη MongoDB ότι αν δεν υπάρχει ήδη πεδίο lastModified να το δημιουργήσει.

2. Αντικατάσταση των provider σε όλα τα Έγγραφα τα οποία έχουν ‘qty’ μικρότερου του 50:
   Στη περίπτωση που θέλουμε να ενημερώσουμε παραπάνω από 1 Έγγραφα, θα χρησιμοποιήσουμε την *updateMany()*. Η δομή της και η χρήση της είναι είναι ίδια με της *updateOne()*:

```
db.inventory.updateMany(
        { "qty": { $lt: 50 } },
        {
                $set: { "sales.provider": "jk", status: "B" },
                $currentDate: { lastModified: true }
        }
)
```

3. Αντικατάσταση ολόκληρου του περιεχομένου ενός Εγγράφου εκτός απο το *_id* πεδίο:

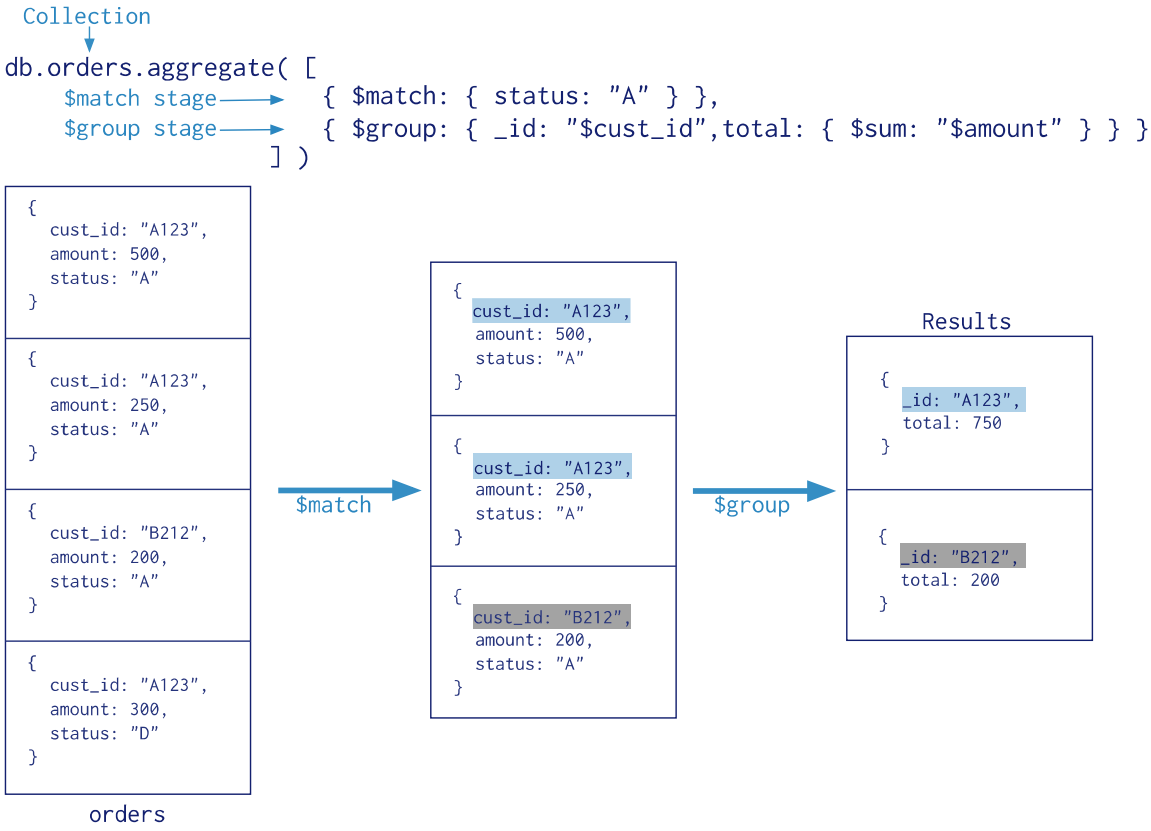
Για να το πετύχουμε αυτό μπορούμε να χρησιμοποιήσουμε τη μέθοδο *replaceOne()*. Η σύνταξη της αποτελείτα απο δύο μέρη, το πρώτο είναι τα κριτήρια που θα χρησιμοποιηθούν για την εύρεση του Έγγραφου στο οποίο θα εφαρμοστεί η αντικατάσταση και το δεύτερο είναι όλοκληρο το νέο Έγγραφο που θα αντικαταστήσει το τρέχων:

```
db.inventory.replaceOne(
        { item: "paper" },
        { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
)
```

Το παραπάνω παράδειγμα αντικαθιστά το πρώτο Έγγραφο που θα βρει να έχει *item : “paper”* με ένα νέο(τελείως διαφορετικό με πριν) Έγγραφο.

2.3.3.7 Σύνολο (Aggregation)

Η MongoDB προσφέρει ένα ειδικό σύνολο λειτουργιών συγχώνευσης το οποίο επεξεργάζεται και επιστρέφει συγχωνευμένα αποτελέσματα. Ομαδοποιεί τιμές απο πολλά Έγγραφα μαζί και εκτελεί κάποιες λειτουργίες ώστε να ομαδοποιήσει τα δεδομένα σε ένα αποτέλεσμα. Για να εφαρμοστεί απαιτεί ένα κριτήριο ταιριάσματος και τα πεδία τα οποία θα χρησιμοποιήσει στην ομαδοποίηση.
Εάν μελετήσουμε την παρακάτω εικόνα θα καταλάβουμε καλύτερα πως δουλεύει.



Εικόνα 6: Λειτουργία aggregation

Παρατηρώντας την εικόνα 4 θα δούμε ότι χρησιμοποιεί την μέθοδο aggregate() σε μια Συλλογή “orders”. Η μέθοδος *aggregate()* αποτελείται απο δύο μέρη. Το πρώτο είναι το κριτήριο ταιριάσματος που θα χρησιμοποιηθεί και το δεύτερο είναι η ομαδοποίηση που θα εφαρμοστεί.

Για το *$match* τελεστή δεν έχουμε να πούμε πολλά καθώς η λειτουργία του είναι ξεκάθαρη. Στη προκειμένη περίπτωση θα ψάξει να βρει όλα τα Έγγραφα που έχουν το πεδίο *Status:"A"*.

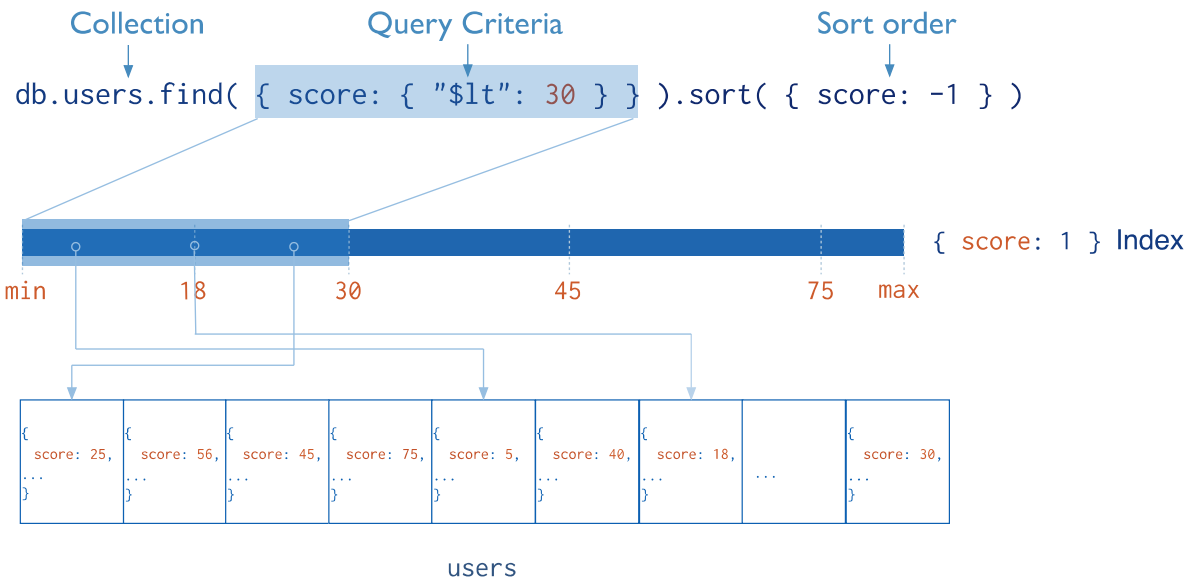
Ο τελεστής *$group* χρησιμοποιείται για την ομαδοποίηση εγγράφων. Απαιτεί να περιέχει ένα *_id* πεδίο ως προς το οποίο θα ομαδοποιήσει τα έγγραφα και οποιαδήποτε άλλα προαιρετικά πεδία που θα εμφανίσει στο αποτέλεσμα.

Στο παράδειγμα της εικόνας θα ομαδοποιήσει τα έγγραφα που έχουν *status: "A"* ως προς το *cust_id* πεδίο και στο αποτέλεσμα θα περιέχει και το σύνολο του πεδίου *amount* για κάθε *cust_id*.

Ο τελεστής *$sum* υπολογίζει και επιστρέφει ένα αριθμητικό σύνολο, στη συγκεκριμένη περίπτωση επιστρέφει το σύνολο των *amount* στις ίδιες παραγγελίες.

2.3.3.8 Ευρετήρια (Indexes)

Τα ευρετήρια υποστηρίζουν την αποτελεσματικότερη εκτέλεση ερωτημάτων(queries). Χωρίς ευρετήρια η MongoDB πρέπει να εκτελέσει σάρωση ολόκληρης της συλλογής(κάθε Έγγραφο) για να επιλέξει τα έγγραφα που ταιριάζουν στο πεδίο αναζήτησης. Σε περίπτωση όμως που υπάρχει ένα ευρετήριο για μια αναζήτηση η MongoDB περιορίζει σε σημαντικό βαθμό τον αριθμό των εγγράφων που πρέπει να σαρώσει.



Εικόνα 7: Λειτουργία ευρετηρίων

Τα ευρετήρια είναι ειδικές δομές δεδομένων που αποθηκεύουν ταξινομημένα ένα μικρό μέρος των δεδομένων μιας συλλογής ώστε να πετυχαίνονται γρηγορότερα αποτελέσματα. Στην παραπάνω είκονα 7 φένεται η λειτουργία τους.
Μπορούμε να δημιουργήσουμε ένα ευρετήριο χρησιμοποιώντας την μέθοδο:
db.collection.createIndex(<ονομα πεδίου και τύπος του index>, <προαιρετικές επιλογές>)

Η MongoDB υποστηρίζει αρκετούς τύπους ευρετηρίων, όλοι είναι διαθέσιμοι και μπορούν να βρεθούν στη επίσημη ιστοσελίδα της [6].
Ο τύπος ευρετηρίων που χρησιμοποιήθηκε σε αυτή τη πτυχιακή είναι ο *2dsphere* ο οποίος υποστηρίζει ερωτήματα γεωγραφικών συντεταγμένων χρησιμοποιώντας επίπεδη γεωμετρία. Για να μπορέσει ένα πεδίο να οριστεί ως *2dsphere* ευρετήριο θα πρέπει να ακολουθεί μια συγκεκριμένη δομή, πιο συγκεκριμένα θα πρέπει να είναι ένα GeoJSON πεδίο.
Παρακάτω φένεται η δομή ενός GeoJSON πεδίου:
field_name : { type : "Point", array_field_with_coordinates : [numberX, numberY] }

Προσοχή! Μπορεί τα ευρετήρια να κάνουν πιο αποτελεσματική μια αναζήτηση, αλλά δε θα πρέπει να το παρακάνουμε, όσο μεγαλύτερος ο αριθμός των ευρετηρίων τόσο μειώνεται η απόδοση.

2.3.3.9 Γεωγραφική αναζήτηση (Geospatial Query)

Για μια γεωγραφική αναζήτηση χρησιμοποιείται ο τελεστής *$nearSphere*. Έχει ως λειτουργία να καθορίζει ένα γεωγραφικό σημείο από το οποίο θα επιστραφούν τα έγγραφα της συλλογής ταξινομημένα βάση της απόστασης τους απο το σημείο αυτό. Για να γίνει αυτό, απαραίτητη προυπόθεση είναι να υπαρχει ένα ευρετήριο τύπου *2dsphere* και να καθοριστεί ένα GeoJSON σημείο στο ερώτημα.
Μπορούμε να δημιουργήσουμε ένα *2dsphere* ευρετήριο χρησιμοποιώντας στην *createIndex()* μέθοδο η οποία θα έχει ως κλειδί το πεδίο τοποθεσίας και ως τύπο ευρετηρίου το 2dsphere:

db.collection.createIndex({ <pedio topothesias> : "2dsphere" })

2.3.3.9.1 Παράδειγμα

Έστω ότι έχουμε μια συλλογή "places" που περιέχει Έγγραφα με ένα πεδίο *location* και ένα *name*. Θέλουμε να βρούμε ποια τοποθεσία των Εγγράφων απέχει μεταξύ 1000-5000 μέτρα απο ένα συγκεκριμένο γεωγραφικό σημείο.

Αρχικά εισάγουμε τα έγγραφα στην συλλογή:

```
db.places.insertOne(
        {
                location : { type: "Point", coordinates: [ -73.97, 40.77 ] },
                name: "Central Park"
        }
)


db.places.insertOne(
         {
                location : { type: "Point", coordinates: [ -73.88, 40.78 ] },
                name: "La Guardia Airport"
        }
)
```

Επειτα δημιουργούμε το *2dsphere* ευρετήριο εισάγωντας ως κλειδί το πεδίο location:

db.places.createIndex({ location : "2dsphere" })

Στη συνέχεια εκτελούμε το παρακάτω ερώτημα:

```
db.places.find(
  {
        location: {
              $nearSphere: {
                    $geometry: {
                          type : "Point",
                          coordinates : [ -73.9667, 40.777 ]
                    },
                    $minDistance: 1000,
                    $maxDistance: 5000
              }
        }
  }
)
```

Ο τελεστής *$geometry* ορίζει ένα GeoJSON πεδίο απο το οποίο θέλουμε να πάρουμε την απόσταση των εγγράφων της συλλογής.

2.3.3.10 GridFS

Το GridFS χρησιμοποιείται για την αποθήκευση και την ανάκτηση αρχείων. Αντί για την αποθήκευση ενός αρχείου σε ένα Έγγραφο(Document) το GridFS διαιρεί το αρχείο σε κομμάτια(chunks) και αποθηκεύει κάθε κομμάτι σαν ένα ξεχωριστό Έγγραφο. Σαν προεπιλογή το GridFS χρησιμοποιεί ένα μέγεθος κομματιού από 255 kB, δηλαδή διαιρεί ένα αρχείο σε κομμάτια των 255 kB εκτός του τελευταίου κομματιού. Το τελευταίο είναι όσο μεγάλο χρειάζεται. Αντίστοιχα, αρχεία που δεν ξεπερνούν το μέγεθος ενός κομματιού αποτελούνται μόνο απο ένα κομμάτι και καταλαμβάνουν μόνο το χώρο που χρειάζονται συν κάποια επιπλέον μεταδεδομένα(metadata). Εκτός από την αποθηκευσει αρχείων, το GridFS προσφέρει και τη δυνατότητα να πέρνουμε μόνο συγκεκριμένα κομμάτια από ένα αρχείο χωρίς να χρειάζεται να το φορτώσουμε ολόκληρο, για παράδειγμα από ένα αρχείο μουσικής μπορούμε να πάρουμε μόνο τα κομματιά που αντιστοιχούν από το πρώτο μέχρι το δεύτερο λεπτό της αναπαραγωγής του.

2.3.3.10.1 Συλλογές (Collections)

Το GridFS χρησιμοποιεί δύο συλλογές για να αποθηκεύει αρχεία. Στη μια συλλογή, **chunks**, αποθηκεύονται τα κομμάτια του αρχείου σε δυαδική μορφή και στην άλλη, **files**, τα μεταδεδομένα. Προεπιλεγμένα, το συγκεκριμένο όνομα που δίνει η MongoDB σε κάθε μια από της παραπάνω συλλογές είναι: *fs.chunks* και *fs.files* . Εάν επιθυμούμε μπορούμε να τις δώσουμε άλλο όνομα.

2.3.3.10.1.1 Chunks

Κάθε Έγγραφο στη *chunks* συλλογή αναπαριστά ένα μοναδικό κομμάτι ενός αρχείου. Τα έγγραφα σε αυτή τη συλλογή έχουν την ακόλουθη φόρμα:

{
"_id" : <ObjectId>,
"files_id" : <ObjectId>,
"n" : <num>,
"data" : <binary>
}

Ενα Έγγραφο από αυτή τη συλλογή περιέχει τα εξείς πεδία:

- **chunks._id**
  Το μοναδικό ObjectId του κομματιού.

- **chunks.files_id**
  Το _id του 'γονέα' εγγράφου, όπως καθορίζεται στη *files* συλλογή.

- **chunks.n**
  Ο αριθμός ακολουθίας του κομματιού, ξεκινώντας από το 0.

- **chunks.data**
  Το φορτίο του κομματιού, τα δυαδικά δεδομένα.

2.3.3.10.1.2 Files

Κάθε Έγγραφο στη συλλογή *files* αναπαριστά ένα αρχείο. Ένα Έγγραφο σε αυτή τη συλλογή έχει την ακόλουθη μορφή:

```
{
  "_id" : <ObjectId>,
  "length" : <num>,
  "chunkSize" : <num>,
  "uploadDate" : <timestamp>,
  "md5" : <hash>,
  "filename" : <string>,
  "contentType" : <string>,
  "aliases" : <string array>,
  "metadata" : <dataObject>,
}
```

Τα έγγραφα σε αυτή τη συλλογή περιέχουν μερικά ή όλα από τα παρακάτω πεδία:

- **files._id**
  Το μοναδικό ID του εγγράφου. Ο τύπος δεδομένων για το _id είναι αυτός που επιλέχτηκε για το αρχείο.

- **files.length**
  Το μέγεθος του εγγράφου σε bytes.

- **files.chunkSize**
  Το μέγεθος κάθε κομματιού σε bytes.

- **files.uploadDate**
  Η ημερομηνία που το Έγγραφο αποθηκεύτηκε για πρώτη φορά.

- **files.md5**
  Το MD5 hash αποτέλεσμα ολόκληρου το αρχείου.

- **files.filename**
  Προαιρετικό, ένα όνομα για το συγκεκριμένο αρχείο.

- **files.contentType**
  Προαιρετικό, ένας έγκυρος MIME τύπος για το αρχείο.

- **files.aliases**
  Προαιρετικό, ένας πίνακας από ψεύδωνυμα για το αρχείο.

- **files.metadata**
  Προαιρετικό, Επιπλέον πληροφορίες που θέλουμε να αποθηκέυσουμε.

2.3.3.10.2 Ευρετήρια (Indexes)

Το GridFS χρησιμοποιεί αυτόματα κάποια ευρετήρια και για τις δύο του συλλογές για αποδοτικότερα αποτελέσματα.
Τα ευρετήρια που δημιουργούνται για την συλλογή *chunks* αποτελούνται από τα πεδία: *files_id* και *n*.
Τα ευρετήρια που δημιουργούνται για τη συλλογή *files* αποτελούνται από τα πεδία: *filename* και *uploadDate*.
Στις παραπάνω συλλογές μπορούμε να προσθέσουμε και επιπλέον ευρετήρια που εμείς επιθυμούμε.

2.3.3.10.3 Χρήση GridFS

Για να αποθήκευσουμε και να ανακτήσουμε αρχεία χρησιμοποιώντας το GridFS μπορούμε με δύο τρόπους:

- Προγραμματιστικά, χρησιμοποιώντας έναν MongoDB οδηγό(driver).
- Από το κέλυφος, χρησιμοποιώντας την εντολή mongofiles [8].

2.3.3.11 Πότε δεν προτείνεται η MongoDB

Σε περίπλοκα συστήματα που απαιτούνται πολλές συσχετίσεις μεταξύ των πινάκων στη βάση δεδομένων, η MongoDB δε προτείνεται. Υπάρχει η δυνατότητα για συσχετίση μεταξύ των εγγράφων της αλλά η χρήση της σε τέτοια περίπτωση ίσως αποφέρει περισσότερα αρνητικά παρά θετικά.

2.3.4 Ασφάλεια

2.3.4.1 Basic Authentication

Το Basic Authentication είναι μια μέθοδος η οποία δίνει τη δυνατότητα σε κάποιον πελάτη(client) να παρέχει ένα όνομα χρήστη(username) και ένα κωδικό χρήστη(password) όταν στέλνει ένα HTTP ερώτημα.
Είναι η πιο απλή μέθοδος ασφάλειας για προστασία των πόρων από μη αυθεντικοποιημένους χρήστες σε κάποιο REST API καθώς το μόνο που απαιτεί είναι το όνομα χρήστη και ο κωδικός χρήστη συγχωνευμένα σε **Base64* μορφή.
Η εισαγωγή του στο ερώτημα γίνεται στη κεφαλίδα(Header) εισάγοντας ένα πεδίο:
Authorization με τιμή: *Basic username:PasswordInBase64Encode*

Για παράδειγμα αν το όνομα χρήστη είναι: Aladdin και ο κωδικός: OpenSesame τότε η κωδικοποίηση σε base64 του Aladdin:OpenSesame είναι:
QWxhZGRpbjpPcGVuU2VzYW1l
Αρα το πεδίο Authorization στη κεφαλίδα θα έχει τιμή:
Basic QwxhZGRpbjpPcGVuU2VzYW1l

* Base64 είναι η κωδικοποίηση δυαδικών δεδομένων σε ASCII κείμενο. Αντιστοιχεί 4 χαρακτήρες για κάθε 3 bytes δεδομένων συν ένα πιθανώς επιπλέον bit γεμίσματος στο τέλος 

2.3.5 Google Maps API (Geocode)

Πρόκειται για ένα Rest API σχεδιασμένο απο τη Google που προσφέρει λειτουργίες σχετικά με τα Google Maps.

Με τον όρο Geocode εννοείται η διαδικασία μετατροπής μιας διεύθυνσης ("Τσιμισκή 147, Θεσσαλονίκη") σε γεωγραφικές συντεταγμένες(γεωγραφικό μήκος, γεωγραφικό πλάτος). Δηλαδή ένας πελάτης(client) στέλνει ένα ερώτημα με το όνομα μιας διεύθυνσης και το API του απαντά ποιές είναι οι γεωγραφικές συντεταγμένες αυτής της διεύθυνσης.

Επίσης δίνεται και η δυνατότητα για την αντίστροφη διαδικασία(Reverse geocode) όπου γεωγραφικές συντεταγμένες μετατρέπονται στο αντίστοιχο όνομα διεύθυνσης. Δηλαδή ένας πελάτης στέλνει ένα ερώτημα με γεωγραφικές συντεταγμένες και το API του απαντά ποιά διεύθυνση βρίσκεται εκεί.

2.3.5.1 Λειτουργία

Αρκεί να στείλουμε ένα HTTP GET ερώτημα στο παρακάτω URL:

http://maps.googleapis.com/maps/api/geocode/outputFormat?parameters

Σαν *outputFormat* στο URL μπορούμε να δώσουμε: json ή xml και αφορά την μορφή των δεδομένων που θα έχει η απάντηση του API. Σαν *parameters* στο URL πρέπει να δώσουμε το όνομα της διεύθυνσης.

Για παράδειγμα το παρακάτω HTTP GET ερώτημα:

https://maps.googleapis.com/maps/api/geocode/json?address=Τσιμισκή77,Θεσσαλονίκη

Θα επιστρέψει το παρακάτω:

```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "77",
               "short_name" : "77",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Tsimiski",
               "short_name" : "Tsimiski",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Thessaloniki",
               "short_name" : "Thessaloniki",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Thessaloniki",
               "short_name" : "Thessaloniki",
               "types" : [ "administrative_area_level_3", "political" ]
            },
            {
               "long_name" : "Greece",
               "short_name" : "GR",
               "types" : [ "country", "political" ]
            },
            {
               "long_name" : "546 22",
               "short_name" : "546 22",
               "types" : [ "postal_code" ]
            }
         ],
```

```
      "formatted_address" : "Tsimiski 77, Thessaloniki 546 22, Greece",
      "geometry" : {
        "bounds" : {
          "northeast" : {
            "lat" : 40.6312668,
            "lng" : 22.9452693
          },
          "southwest" : {
            "lat" : 40.631256,
            "lng" : 22.945257
          }
        },
        "location" : {
          "lat" : 40.631256,
          "lng" : 22.945257
        },
        "viewport" : {
          "northeast" : {
            "lat" : 40.6326103802915,
            "lng" : 22.9466121302915
          },
          "southwest" : {
            "lat" : 40.6299124197085,
            "lng" : 22.9439141697085
          }
        }
      },
      "place_id":"EkDOpMDzrnOvM65z4POus6uIDc3LCDOmM61z4PPg86xzrvOv869zq_Ous63IDU0NiAyMiwgzpXOu867zq
zOtM6x",
      "types" : [ "street_address" ]
    }
  ],
  "status" : "OK"
}
```

Εάν παρατηρήσουμε την απάντηση θα δούμε ότι πρόκειται για ένα JSON το οποίο περιέχει ένα *result* που είναι μια λίστα απο JSON αντικείμενα και ένα *status* το οποίο μας ενημερώνει για το αν το ερώτημα ολοκληρώθηκε επιτυχώς.

Το *result* πιο συγκεκριμένα περιέχει τέσσερα JSON αντικείμενα, το *address_components* το *formatted_address*, το *geometry*, το *place_id* και το *types*.

Το *address_components* περιέχει μια λίστα με JSON αντικείμενα που κάθε αντικείμενο είναι τη μορφής: *long_name, short_name, types*. Αυτό αφορά τη διεύθυνση που στείλαμε και βλέπουμε οτι περιέχει τον ταχυδρομικό κώδικα, τη χώρα, τη περιοχή, το όνομα και τον αριθμό της διεύθυνσης.

Το *formatted_address* αναγράφει το όνομα της διεύθυνσης που στείλαμε.

Το *geometry* περιέχει ένα *bound* JSON αντικείμενο, ένα *location* και ένα *viewport.*

Το *bounds* αφορά τα γεωγραφικά όρια που ανήκει αυτή η διεύθυνση.

Το *location* αφορά την συγκεκριμενη γεωγραφική τοποθεσία αυτής της διεύθυνσης.

Το *place_id* αφορά το μοναδικό id αυτής της διεύθυνσης στο google και το *types* τι τύπος είναι αυτή η διεύθυνση.

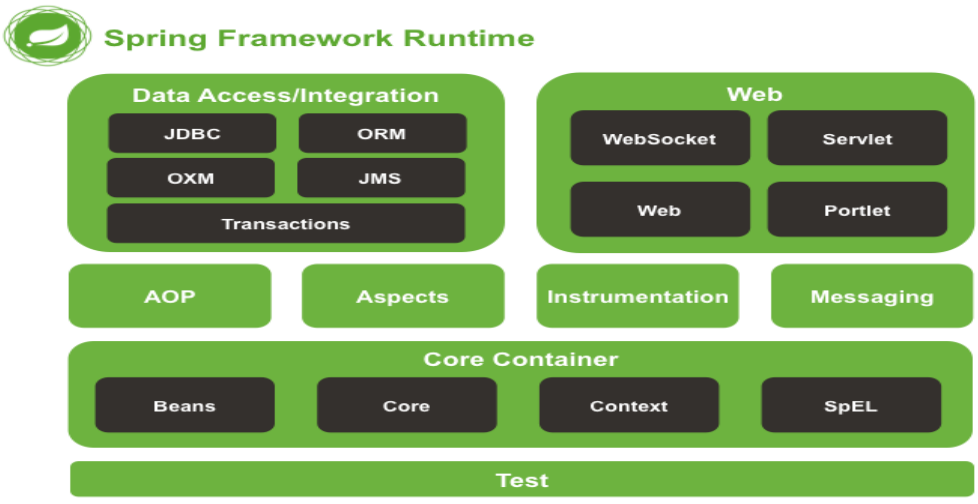
Με παρόμοιο τρόπο πραγματοποιείται και ένα ερώτημα για το Reverse Geocode αλλά αυτή τη φορά αντί για διεύθυνση δίνουμε στο ερώτημα τις γεωγραφικές συντεταγμένες. Παράδειγμα, το παρακάτω ερώτημα που ζητάει πληροφορίες για το γεωγραφικό σημείο: 40.629933, 22.947175:

https://maps.googleapis.com/maps/api/geocode/json?latlng=40.629933,22.947175

2.3.6 Spring framework

Είναι ένα δημοφιλής java framework για την ανάπτυξη διαδικτυακών και με μεγάλο σκέλος εφαρμογών. Στη πρώτη του έκδοση(το 2002) ξεκίνησε ως ένα Inversion of control container(Dependency Injection) αλλά στη συνέχεια εξελίχθηκε σε πολύ περισσότερα. Αποτελείται απο 20 ενότητες(Modules) οι οποίες όπως φένονται και στη εικόνα 8 ομαδοποιούνται σε:

- Core container
- Data Access/Integration
- Web
- AOP (Aspect Oriented Programming)
- Instrumentation
- Messaging,
- Test



Εικόνα 8: Δομή Spring framework

2.3.6.1 Inversion of control container(Dependency Injection)

Παρέχει τα μέσα για τη διαμόρφωση και τη διαχείριση Java αντικειμένων χρησιμοποιώντας αντανάκλαση(reflection, η δυνατότητα ενός προγράμματος να εκτελεί, εξετάζει και να αλλάζει τη δομή και τη συμπεριφορά του κατά το χρόνο εκτέλεσης(runtime)). Ο container είναι υπεύθυνος να διαχειρίζεται συγκεκριμένα αντικείμενα(τα δημιουργεί, καλεί τις μεθόδους αρχικοποίησης τους και τα διαμορφώνει έτσι ώστε να είναι συνδεδεμένα). Αντικείμενα που δημιουργούνται από τον container ονομάζονται *Beans*. Ο container μπορεί να ρυθμιστεί είτε απο XML αρχεία είτε μέσω Java annotations σε αρχεία κλάσεων, αυτές οι ρυθμίσεις περιέχουν τη πληροφορία που απαιτείται για να δημιουργηθηκούν τα beans. Τα αντικείμενα μπορούν να χρησιμοποιηθούν μεσω του *Dependency injection* το οποίο είναι πρότυπο όπου ο container περνά αντικείμενα κατ'ονομα σε άλλα αντικείμενα μέσω δομητών(constructors) ή πεδίων(fields) ή εργοστασιακών μεθόδων(factory methods). Ετσι, όταν ο προγραμματιστής σε μια κλάση χρειάζεται να χρησιμοποιήσει κάποιο αντικείμενο μιας άλλης κλάσης, δεν είναι ανάγκη να το δημιουργησεί (μέσω του new τελεστή), μπορεί να το λάβει αρχικοποιημένο απο τον container.

2.3.6.2 Ανάλυση ενοτήτων

2.3.6.2.1 Core container:

Ο core container αποτελείτε απο τις ενότητες: *Beans*, *Core*, *Context*, *SpE*L.

Τα **Beans**, **Cores** παρέχουν τα θεμελιώδη μέρη του framework, συμπεριλαμβανομένου του IoC(inversion of control container) και του Dependency Injection χαρακτηριστικών.

Το **Context** βασίζεται στο Core και στα Beans, είναι ένα μέσο για τη πρόσβαση σε αντικείμενα. Υποστηρίζει επίσης χαρακτηριστικά της Java EE και παρέχει υποστήριξη για ενσωμάτωση βιβλιοθηκών από τρίτους, προσωρινή αποθήκευση(caching), mailing, χρονοδρομολόγηση. Σαν επίκεντρο έχει τη διεπαφή *ApplicationContext* .

Το **SpEL**(spring-expression) παρέχει μια ισχυρή γλώσσα έκφρασης(Expression Language) για ερωτήματα και χειρισμούς ενός αντικειμένου κατά τη διάρκεια εκτέλεσης(runtime). Υποστηρίζει set και get μεθόδους, επίκληση μεθόδων, πρόσβαση στα περιεχόμενα ενός πίνακα ή συλλογών(collection), αριθμητικούς και λογικούς τελεστές, ονομασία μεταβλητών και ανάκτηση αντικειμένων βάση ονόματος από τον IoC container.

2.3.6.2.2 AOP και Instrumentation

Παρέχει μια υλοποίηση για το AOP(Alliance-compliant aspect oriented programming) [9]. Το AOP είναι ένα τρόπος για να τροποποιήσουμε υπάρχουσες κλάσεις και να τις αλλάξουμε συμπεριφορά χρησιμοποιώντας κανόνες που ορίζονται ξεχωριστά. Αυτοί οι κανόνες ονομάζονται Aspects. Για παράδειγμα, όταν μια μέθοδος εκτελείται μπορούμε να προσθέσουμε επιπλέον λειτουργικότητα πριν ή μετά την εκτέλεση της, χωρίς να τροποποιήσουμε καθόλου τον κώδικα της αλλά χρησιμοποιώντας ένα Aspect το οποίο βρίσκεται σε ξεχωριστό αρχείο. Το Spring AOP προσφέρει έναν απλό και ισχυρό τρόπο για να δημιουργήσουμε τέτοιους κανόνες.

2.3.6.2.3 Messaging

Παρέχει βασικές αφαιρέσεις(Abstractions) όπως: Message, MessageChannel, MessageHandler και άλλες για να χρησιμοποιηθούν ως θεμέλιο σε εφαρμογές μηνυμάτων. Παρέχει επίσης ένα σύνολο απο annotations για να αντιστοιχεί μηνύματα σε μεθόδους.

2.3.6.2.4 Data Access/Integration

Αποτελείται απο τα: *JDBC*, *ORM*, *OXM*, *JMS* και *Transaction*

JDBC: πρόκειται για ένα αφηρημένο επίπεδο για την χρήση του JDBC.

ORM: παρέχει επίπεδα ενσωμάτωσης των δημοφιλών Object Relation Mapping. APIs, συμπεριλαμβανομένου JPA, JDO και HIBERNATE.

OXM: ένα αφηρημένο επίπεδο που υποστηρίζει Object/XML mapping υλοποιήσεις όπως JAXB, Casto, XMLBeans, JiBX και Xstream.

JMS: περιέχει χαρακτηριστικά για αποστολή και λήψη μηνυμάτων.

TX: Παρέχει λειτουργίες για συνναλλαγές για ολα τα POJOs και για κλάσεις που υλοποιούν συγκεκριμένες διεπαφές.

2.3.6.2.5 Web

Αποτελείται απο το *web*, *webmvc*, *websocket*, *portlet.*

Το **web** παρέχει χαρακτηριστικά όπως το ανέβασμα πολλαπλών αρχείων και την αρχικοποίηση του IoC container. Επίσης διαθέτει έναν HTTP πελάτη και τα διαδικτυακά μέρη απο την απομακρυσμένη υποστήριξη.

Το **webmvc** περιέχει το Model-View-Controller(MVC της Spring) και την υλοποίηση REST υπηρεσιών για διαδικτυακές εφαρμογές. Το Spring MVC παρέχει έναν διαχωρισμό μεταξύ του business logic και του UI της εφαρμογής και ενσωματώνεται με όλα τα άλλα χαρακτηριστικά του framework.

Το **portlet** παρέχει την MVC υλοποίηση για να χρησιμοποιηθεί σε ένα portlet περιβάλλον [10].

2.3.6.2.6 Test

Υποστηρίζει τα **Unit tests** [11] και τα **Integration tests** [12].

Επίσης διαθέτει τα **Spring tests** τα οποία αρχικοποιούν όλο το framework κατά τη διάρκεια του test για να μπορέσουμε να χρησιμοποιήσουμε όλα αυτά που μας προσφέρει(IoT, dependency injection, context κλπ).

2.3.6.3 Spring projects

Εκτός του βασικού framework έχουν αναπτυχθεί και άλλα Spring projects που βοηθάνε στην ανάπτυξη εφαρμογών, κάθε ένα από αυτά επικεντρώνεται σε συγκεκριμένες λειτουργίες. Αυτά που χρησιμοποιήθηκαν στη συγκεκριμένη πτυχιακή εργασία είναι τα παρακάτω:

2.3.6.3.1 Spring boot

Οπως είδαμε και παραπάνω για να δημιουργηθεί ένα bean χρειάζεται να υπάρχει και το κατάλληλο αρχείο ρυθμίσεων. Τα περισσότερα χαρακτηριστικά του framework απαιτούν κάποιο αρχείο ρυθμίσεων για να λειτουργήσουν, όσο αυξάνονται τα χαρακτηριστικά τόσο αυξάνονται και τα αρχεία ρυθμίσεων. Ετσι η ομάδα του Spring για να τα περιορίσει, δημιουργήσε το *Spring boot*. Ενα νέο project το οποίο δημιουργεί αυτόματα, τα περισσότερα, απαραίτητα αρχεία ρυθμίσεων και αφήνει το προγραμματιστή να επικεντρωθεί στα πρόβλημα του προγράμμτος του και όχι σε αυτά του framework.

2.3.6.3.2 Spring data

Επικεντρώνεται στη διαχείριση των δεδομένων. Απλοποιεί την πρόσβαση και το χειρισμό των δεδομένων σε σχεσιακές και μη σχεσιακές βάσεις δεδομένων παρέχωντας έτοιμες αφαιρέσεις και διεπαφές.

2.3.6.3.3 Spring security

Επικεντρώνεται στην ασφάλεια. Παρέχει κλασεις και διεπαφές που βοηθάνε στην ευκολότερη υλοποίηση της αυθεντικοποίησης και εξουσιοδότησης χρηστών.

2.3.7 Προγραμματιστικά περιβάλλοντα

2.3.7.1 Intellij IDEA

Πρόκειται για ένα από τα δημοφιλέστερα και πιο ολοκληρωμένα προγραμματιστικά περιβάλλοντα(IDE) το οποίο αναπτύσεται από την JetBrain.

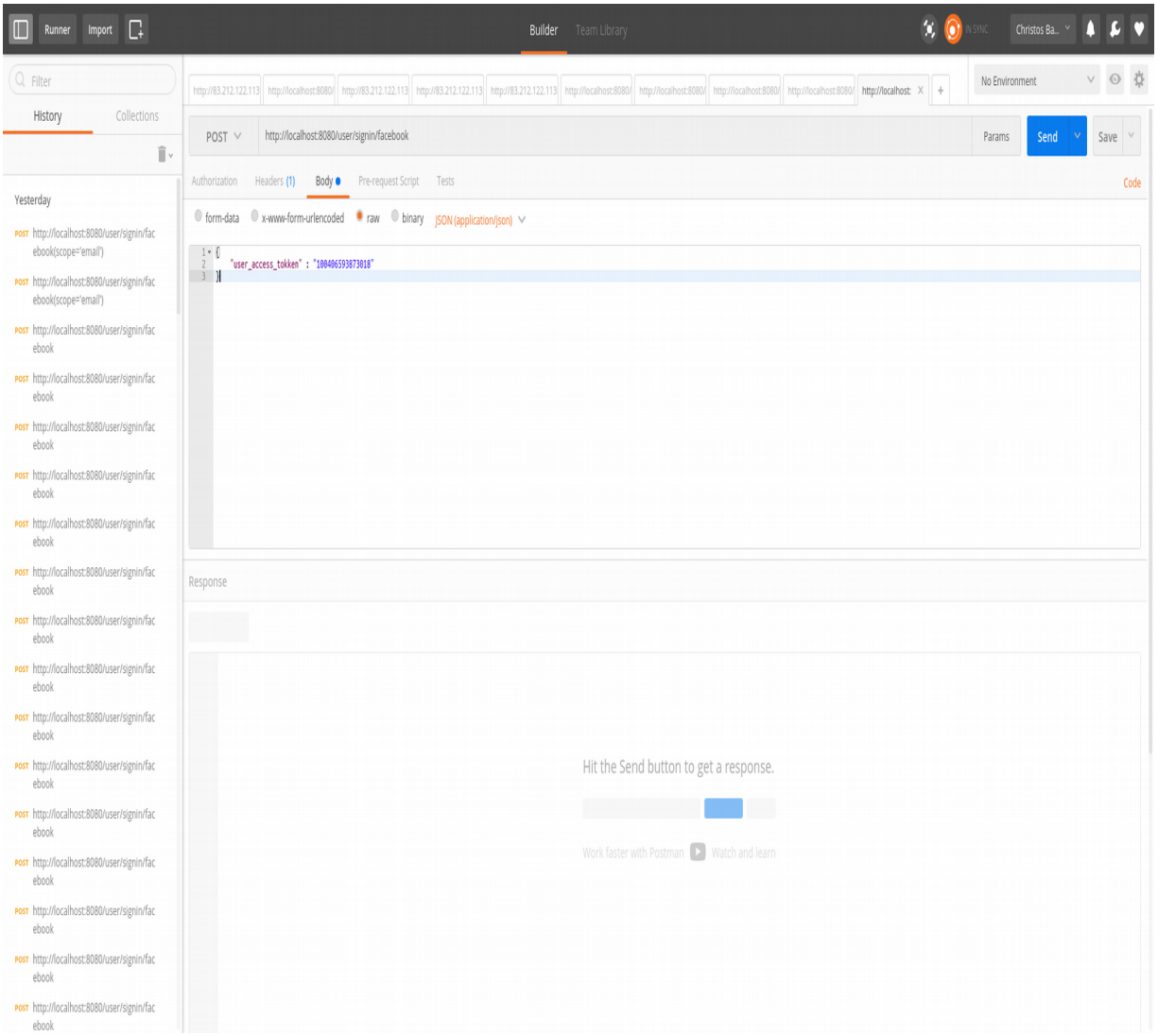
Αποτελείται απο 2 εκδόσεις, μια δωρεάν η οποία υποστηρίζει οποιαδήποτε γλώσσα χρησιμοποιεί το JVM (java, Groovy, Scala, Android κ.α) και μια επι πληρωμής η οποία υποστηρίζει επιπλέον γλώσσες διαδικτύου(javascript, HTML, Anjular κ.α) καθώς και διάφορα frameworks(Spring, Java EE, Grails, Junit κ.α). Έχει γίνει ευρέως γνώστο χάρεις στα χαρακτηριστικά που προσφέρει(και στις 2 εκδόσεις) και διευκολύνει τον προγραμματιστή. Χαρακτηριστικά όπως: Decompiler, Version Control και βάσεων δεδομένων εργαλεία καθώς και έναν αναπτυγμένο κειμενογράφο, ο οποίος παρέχει υπηρεσίες οπως αυτόματη παραγωγή κώδικα, αναλύση κώδικα κατά την πληκτρολόγηση και ορθογραφικό έλεγχο.

Σχεδιασμένο να λειτουργεί και στα τρία γνωστά λειτουργικά συστήματα(Linux, Windows, MacOS).

Για δοκιμή και περισσότερες πληροφορίες [13].

2.3.7.2 Postman

Ενα εργαλείο που χρησιμοποιείται στην ανάπτυξη ενός Rest API. Προσφέρει έναν εύκολο τρόπο να δοκιμαστεί ένα Rest API στέλνωντας HTTP ερωτήματα από το γραφικό του περιβάλλον. Προσφέρεται για όλα τα γνωστά λειτουργικά συστήματα(Linux, Windows, MacOS) [14] καθώς και ώς πρόσθετο στο φυλομετρητή Chrome. Το περιεχόμενο του Postman φένεται στη παρακάτω εικόνα.

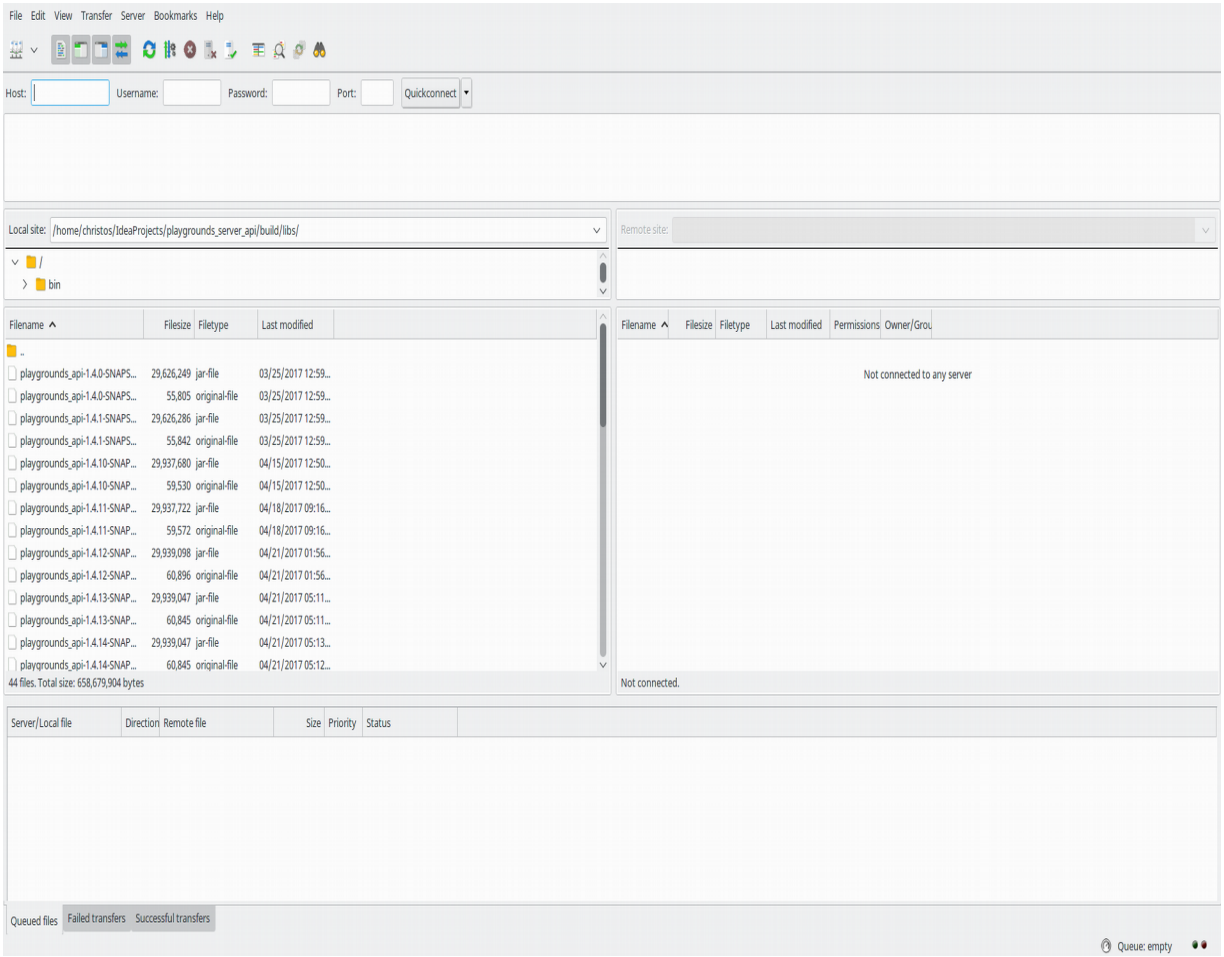


Εικόνα 9: Postman περιβάλλον

2.3.7.3 Filezilla

Πρόκειται για έναν FTP πελάτη τον οποίο μπορούμε να χρησιμοποιήσουμε για να ανεβάσουμε αρχεία σε κάποιον εξυπηρετητή. Υπάρχουν διαθέσιμες εκδόσεις για Linux, Windows και MacOS. Είναι ένα ελεύθερο λογισμικό και διαθέσιμο στο παρακάτω σύνδεσμο [15].

Ακολουθεί μια εικόνα που παρουσιάζει την κύρια οθόνη του.



Εικόνα 10: Filezilla περιβάλλον

2.4 Επίλογος

Συνοψίζοντας, σε αυτό το κεφάλαιο έγινε μια προσπάθεια να περιγραφούν αναλυτικά η αρχιτεκτονική, οι τεχνολογίες που χρησιμοποιήθηκαν καθώς και τα περιβάλλοντα ανάπτυξης προκειμένου να ολοκληρωθεί επιτυχώς το REST API.

Είμαστε έτοιμοι λοιπόν να περιγράψουμε όλα τα διαθέσιμα URL’s που δημιουργήθηκαν και αποτελούν το REST API της πτυχιακής εργασίας.

3 Χρήση και λειτουργία του Rest API

3.1 Εισαγωγή

Στο τρίτο και τελευταίο κεφάλαιο που αφορά το Rest API και την μεριά του εξυπηρετητή(server) θα γίνει εκτενή αναφορά στα διαθέσιμα URL's που αναπτύχθηκαν και πως μπορεί ένα πελάτης(client) να τα χρησιμοποιήσει.

3.2 Επεξηγήσεις

Σε αυτό το σημείο, πριν ξεκινήσουμε να αναλύουμε τα διαθέσιμα URL's , αξίζει να αναφερθούν κάποιες επεξηγήσεις για την καλύτερη κατανόηση του αναγνώστη. Αρχικά, για λόγους απλότητας αναφέρεται μόνο το σχετικό τμήμα από κάθε URL και όχι το απόλυτο. Δηλαδή, από ένα απόλυτο URL όπως το παρακάτω:
http://mydomain.com/playgrounds/search?id=1
κρατάμε και αναφέρουμε μόνο το κομμάτι που ακολουθεί μετά το domain name(mydomain.com), το σχετικό URL του παραπάνω συνδέσμου είναι το: */playgrounds/search?id=1*
Επίσης, τα διαθέσιμα URL's ενός Rest API ονομάζονται endpoints(τελικά σημεία). Οπότε από τώρα όποτε θέλουμε να αναφερθούμε σε ένα διαθέσιμο URL αυτό θα αναφέρεται ως endpoint.
Τέλος, σε κάποια endpoints που ακολουθούν γίνεται χρήση του παρακάτω προτύπου:
{ name }
Αυτό σημαίνει πως η τιμή *name* στο URL είναι δυναμική και θα προέρχεται από τον πελάτη(client).

3.3 Endpoints παιδότοπου

Παρακάτω ακολουθούν τα διαθέσιμα endpoints που αφορούν λειτουργίες του παιδότοπου.

/playgrounds

HTTP ερώτημα: POST, PUT

Χρήση: Εισαγωγή ενός νέου παιδότοπου(POST) ή ανανέωση κάποιου ήδη υπάρχων(PUT)

Μορφή πεδίων σώματος: JSON

Απαραίτητα πεδία:

- Αυθεντικοποιημένος χρήστης: εισαγωγή στη κεφαλίδα ενός "Authorization" πεδίου με τιμή το base64 αποτέλεσμα του ID του χρήστη μαζι με τη λέξη *:password*
- Παιδότοπος: Στο σώμα(Body) του HTTP ερωτήματος θα πρέπει να περιέχονται οι εξείς τιμές:
  - name: Όνομα παιδότοπου.
  - address: Διεύθυνση παιδότοπου.
  - location: Συντεταγμένες παιδότοπου. Ένας double πίνακας που περιέχει το γεωγραφικό μήκος/πλάτος του παίδοτοπου.
  - added_by: ID του χρήστη που προσθέτει τον παιδότοπο.

Προαιρετικά πεδία:

Στο σώμα του ερωτήματος μπορεί να περιέχονται προαιρετικά και τα παρακάτω πεδία

- phone: Τηλέφωνο παιδότοπου.
- website: Ιστοσελίδα παιδότοπου.
- base64Image: Η φωτογραφία προφίλ για το παιδότοπο σε base64 μορφή.

/playgrounds/{ playground_id }

HTTP ερώτημα: GET

Μορφή απάντησης: JSON

*Χρήση***:** Επιστρέφονται οι πληροφορίες για έναν συγκεκριμένο παιδότοπο.

Απαραίτητα πεδία: Το μοναδικό ID του παιδότοπου, το οποίο εισάγεται στο URL (playground_id).

Προαιρετικά πεδία: Στο URL μπορεί να προστεθεί η παράμετρος user με τιμή το ID ενός χρήστη έτσι ώστε εάν ο χρήστης έχει βαθμολογήσει το συγκεκριμένο παιδότοπο η βαθμολογία του στην απάντηση του ερωτήματος να μετακινηθεί στη κορυφή των βαθμολογιών(ώστε να είναι η πρώτη που θα βλέπει).

/playgrounds/coordinates

*HTTP ερώτημα**: GET***

*Μορφή απάντησης**: JSON***

Χρήση: Επιστρέφονται όλες οι συνταταγμένες των παιδότοπων, μαζί με τα όνοματά τους.

Απαραίτητα πεδία: Κανένα.

/playgrounds/rate/{ playground_id }

HTTP ερωτήμα: POST, PUT

Μορφή πεδίων σώματος: JSON

Χρήση: Νέα(POST) βαθμολόγη ενός παιδότοπου από έναν χρήστη ή ανανέωση(PUT) τρέχων βαθμολογίας.

Απαραίτητα πεδία:

- Αυθεντικοποιημένος χρήστης στη κεφαλίδα (όπως στο πρώτο URL)
- Το μοναδικό ID του παιδότοπου, το οποίο εισάγεται στο URL (playground_id)

- Η βαθμολογία η οποία τοποθετήται στο σώμα(Body) του ερωτήματος και θα πρέπει να περιέχει τα εξής πεδία:
  - user: Το id του χρήστη που βαθμολογεί.
  - general_rate: Η γενική βαθμολογία του χρήστη (1-5).
  - environmnet: Η βαθμολογία του χρήστη σχετικά με το περιβάλλον (1-5).
  - equipment: Η βαθμολογία του χρήστη σχετικά με τον εξοπλισμό (1-5).
  - kids_supervision: Η βαθμολογία του χρήστη σχετικά με την επίβλεψη των παιδιών.
  - prices: Η βαθμολογία του χρήστη σχετικά με τις τιμές του παιδότοπου.
  - comment: Ένα σχόλιο του χρήστη σχετικά με το παιδότοπο.

/playgrounds/location

HTTP ερώτημα: GET

Μορφή απάντησης: JSON

Χρήση: Επιστρέφονται οι διαθέσιμοι παιδότοποι σε μια τοποθεσία(γεωγραφική) σε ταξινομημένη σειρά βάση της γενικής βαθμολογίας τους.

Απαραίτητα πεδία:

- x : Γεωγραφικό πλάτος, ως παράμετρος στο URL.
- y : Γεωγραφικό μήκος, ως παράμετρος στο URL.

Προαιρετικά πεδία:

- distance: Boolean τιμή για τον αν θέλουμε να μας επιστραφεί και η απόσταση του κάθε παιδότοπου από τη συγκεκριμένη τοποθεσία, εισάγεται ως παράμετρος στο URL. Σαν προκαθορισμένη τιμή έχει false.

/playgrounds/search

HTTP ερώτημα: GET

Μορφή απάντησης: JSON

Χρήση: Αναζήτηση ενός παίδοτοπου(με το όνομα) σε μια συγκεκριμένη τοποθεσία.

Απαραίτητα πεδία:

- x : Γεωγραφικό πλάτος τοποθεσίας, ως παράμετρος στο URL.
- y : Γεωγραφικό μήκος τοποθεσίας, ως παράμετρος στο URL.
- playground: Όνομα παιδότοπου, ως παράμετρος στο URL.

/playgrounds/near_me

HTTP ερώτημα: GET

Μορφή απάντησης: JSON

Χρήση: Επιστρέφονται όλοι οι παιδότοποι που απέχουν απο μια τοποθεσία το πολύ κάποια συγκεκριμένα χιλίομετρα(τα οποία εισάγωνται στο URL).

Απαραίτητα πεδία:

- x : Γεωγραφικό πλάτος τοποθεσίας, ως παράμετρος στο URL.
- y : Γεωγραφικό μήκος τοποθεσίας, ως παράμετρος στο URL.
- maxDistance : Μέγιστη απόσταση(σε χιλιόμετρα) απο τις παραπάνω συντεταγμένες, ως παράμετρος στο URL.

/playgrounds/optionalFields

HTTP ερώτημα: PUT

Μορφή πεδίων σώματος: JSON

Χρήση: Εισαγωγή μη προαιρετικών πεδίων που μπορεί να λείπουν από έναν παιδότοπο, τοποθετούνται στο σώμα του ερωτήματος.

Απαραίτητα πεδία:

- Αυθεντικοποιημένος χρήστης στη κεφαλίδα (όπως στο πρώτο URL).
- id : ID παιδότοπου για τον οποίο θα εισαχθούν τα παρακάτω πεδία.

- phone: Τηλέφωνο παιδότοπου.
- website: Ιστοσελίδα παιδότοπου.

/playgrounds/upload/{playground_id}

HTTP ερώτημα: POST

Μορφή πεδίων σώματος: JSON

Χρήση: Εισαγωγή φωτογραφίας σε έναν παιδότοπο.

Απαραίτητα πεδία:

- Αυθεντικοποιημένος χρήστης στη κεφαλίδα (όπως στο πρώτο URL).
- playground_id: Το ID του παιδότοπου στον οποίο θέλουμε να προσθέσουμε την φωτογραφία, εισάγεται στο URL.
- images: Η φωτογραφία η οποία θα προστεθεί, τοποθετείται στο σώμα του ερωτήμας με τα εξής πεδία:
  - userId: ID του χρήστη ο οποίος εισάγει την φωτογραφία.
  - multipartFile: Η φωτογραφία σε base64 μορφή.

/playgrounds/images/{image_id}

HTTP ερώτημα: GET

Μορφή απάντησης: Image_JPEG

Χρήση: Επιστροφή μιας συγκεκριμένης φωτογραφίας.

Απαραίτητα πεδία:

- image_id : Το ID της φωτογραφίας που θέλουμε, εισάγεται στο URL.

3.4 Endpoints χρήστη

Παρακάτω ακολουθούν τα διαθέσιμα endpoints που αφορούν λειτουργίες του χρήστη.

/users

HTTP ερώτημα: POST

Μορφή πεδίων σώματος: JSON

Χρήση: Δημιουργία νέου χρήστη.

Απαραίτητα πεδία:

- user: Τα στοιχεία του χρήστη που θα δημιουργηθεί, τοποθετείται στο σώμα του ερωτήματος με τα εξής πεδία:
  - username: Το όνομα χρήστη.
  - email: Το ηλεκτρονικό ταχυδρομείο του χρήστη.
  - imageUrl: Ένα URL που περιέχει τη φωτογραφία προφίλ του χρήστη.

/users/{ id }

HTTP ερώτημα: GET

Μορφή απάντησης: JSON

Χρήση: Επιστρέφονται οι πληροφορίες για έναν συγκεκριμένο χρήστη.

Απαραίτητα πεδία:

- id: Το ID του χρήστη για τον οποίον θέλουμε να λάβουμε τις πληροφορίες του, εισάγεται στο URL.

/users/{ user_id }/favorites

HTTP ερωτήματα: POST, GET, DELETE

Μορφή απάντησης: JSON

Μορφή πεδίων σώματος: JSON

Χρήση: Εισαγωγή(POST) ενός νέου παιδότοπου στα αγαπημένα ενός χρήστη, επιστροφή(GET) όλων των αγαπημένων παιδότοπων ενός χρήστη, διαγραφή(DELETE) κάποιου παιδότοπου από τα αγαπημένα ενός χρήστη.

Απαραίτητα πεδία για την εισαγώγη(POST):

- user_id: Το ID του χρήστη για τον οποίο θα εισαχθεί ο παιδότοπος στα αγαπήμενα του, εισάγεται στο URL.
- favorite: Ο παιδότοπος που θα προστεθεί στα αγαπήμενα του χρήστη, εισάγεται στο σώμα του ερωτήμας και περιέχει το παρακάτω πεδίο:
  - playground: Το ID του παιδότοπου.

Απαραίτητα πεδία για την επιστροφή(GET):

- user_id: Το ID του χρήστη για τον οποίο θα επιστραφούν οι αγαπήμενοι παιδότοποι, εισάγεται στο URL.

Απαραιτητα πεδία για την διαγραφή(DELETE):

- user_id: Το ID του χρήστη για τον οποίο θα αφαιρεθεί ο παιδότοπος από αγαπήμενα του, εισάγεται στο URL.
- favorite: Ο παιδότοπος που θα αφαιρεθεί από τα αγαπήμενα του χρήστη, εισάγεται στο σώμα του ερωτήμας και περιέχει το παρακάτω πεδίο:
  - playground: Το ID του παιδότοπου.

3.5 Επίλογος

Σε αυτό το κεφάλαιο έγινε προσπάθεια να περιγραφεί ξεχωριστά η λειτουργία και η χρήση του κάθε διαθέσιμου endpoint του REST API. Με το τέλος αυτού του κεφαλαίου ολοκληρώνεται η αναφορά και ανάλυση του ενός(εξυπηρετητή) από τα δύο μέρη αυτής της πτυχιακής εργασίας.

Τα επόμενα κεφάλαια που ακολουθούν αναφέρονται στο άλλο κομμάτι της πτυχιακής, τον πελάτη, και πιο συγκεκριμένα στο Android.

4 Εισαγωγή στο λειτουργικό σύστημα Android

4.1 Εισαγωγή

Σε αυτό το εισαγωγικό κεφάλαιο επεξηγείται το τι ακριβώς είναι το λειτουργικό σύστημα Android, πάνω στο οποίο αναπτύχθηκε η εφαρμογή της πτυχιακής εργασίας. Επιπρόσθετα θα αναλυθεί η αρχιτεκτονική του αλλά και εκείνα τα χαρακτηριστικά τα οποία το έχουν οδηγήσει να βρίσκεται στη κορυφή των πιο δημοφιλών λειτουργικών συστημάτων.

4.2 Τι είναι το Android

Το Android είναι ένα λειτουργικό σύστημα ανοικτού κώδικα για κινητές συσκευές, βασισμένο στο Linux πυρήνα και σχεδιασμένο αρχικά για οθόνες αφής σε συσκευές, όπως έξυπνα τηλέφωνα(Smartphones) και ταμπλέτες(Tablets). Στη συνέχεια, με τη δημοτικότητα που απέκτησε στην αγορά προσαρμόστηκε να χρησιμοποιείται και σε ρολόγια(Android Wear), τηλεοράσεις(Android TV), αυτοκίνητα(Android Auto) καθώς και σε συσκευές διαδικτύου των πραγμάτων(Internet of Things). Αυτή τη στιγμή πρόκειται για το πιο ευρέως διαδεδομένο λογισμικό στο κόσμο. Η ανάπτυξη του γίνεται απο την Open Handset Alliance (ένας οργανισμός μιας κοινοπραξίας 48 τηλεπικοινωνιακών εταιριών της οποίας ηγείται η Google).

Επιτρέπει στους κατασκευαστές λογισμικού να συνθέσουν κώδικα με τη χρήση της γλώσσας προγραμματισμού Java και C++ μέσω βιβλιοθηκών λογισμικού ανεπτυγμένων απο την Google. Η τελευταία έκδοση η οποία είναι διαθέσιμη τη παρούσα στιγμή ειναι η 7.1.2 με κωδική ονομασία Nougat.

4.3 Κύρια χαρακτηριστικά

Τα κυριότερα χαρακτηριστικά που οδήγησαν το Android να γίνει τόσο διαδεδομένο και νούμερο ένα στις πωλήσεις κινητών συσκευών ακολουθούν παρακάτω:

4.3.1 Δωρεάν και ανοιχτή πηγή

Το Android είναι μια πλατφόρμα ανοικτού κώδικα και δωρεάν για χρήση. Πιο συγκεκριμένα το λειτουργικό σύστημα έχει κατοχυρωθεί με την άδεια χρήσης GNU General Public Licence Version 2 (GPLv2) η οποία υποχρεώνει βελτιώσεις τρίτων να εξακολουθούν να εμπίπτουν στους όρους των προτύπων ανοικτού κώδικα. Αυτό σημαίνει ότι μπορεί οποιοσδήποτε να κλωνοποιήσει τον πρωτότυπο κώδικα του λειτουργικού συστήματος, να τον τροποποιήσει και να διαθέσει μια δικιά του έκδοση με ότι επιπλέον χαρακτηριστικά έχει προσθέσει αρκεί να εξακολουθεί να παραμένει υπό τις ανοικτού κώδικα άδειες. Το περιβάλλον δημιουργίας του Android διανέμεται μέσω της άδειας Apache Software Licence η οποία προβλέπει τη διανομή παρεκκλίσεων ανοιχτής και κλειστής πηγής του πηγαίου κώδικα. Αυτό σημαίνει ότι μπορεί μια εταιρεία να χρησιμοποιήσει τη πλατφόρμα και να της προσθέσει χαρακτηριστικά χωρίς να υποχρεωούται να παρέχει το τελικό προιόν με άδεια ανοικτού κώδικα, κάτι που συμβαίνει αρκετά συχνά θέλωντας έτσι η κάθε εταιρεία να διαφοροποιηθεί από τους υπόλοιπους ανταγωνιστές.

4.3.2 Δωρεάν διαθέσιμα εργαλεία ανάπτυξης λογισμικού

Όλα τα εργαλεία καθώς και το SDK-NDK(οι βιβλιοθήκες προγραμματισμού για την ανάπτυξη εφαρμογών σε Android) που θα χρειαστεί κάποιος για να αναπτύξει μια Android εφαρμογή παρέχονται δωρεάν για όλα τα γνωστά λειτουργικά συστήματα(Linux, Windows, MacOS).

4.3.3 Οικίες γλώσσες προγραμματισμού

Η ανάπτυξη μιας Android εφαρμογής γίνεται είτε μέσω του SDK(software development kit) το οποίο χρησιμοποιεί τη γλώσσα προγραμματισμού Java είτε μέσω του NDK(native development kit) το οποίο χρησιμοποιεί τη γλώσσα προγραμμτισμού C++. Και οι δύο είναι πολύ γνωστές και δημοφιλείς έτσι ώστε να μην απαιτείται από τον προγραματιστή να μάθει μια τελείως καινούργια γλώσσα η οποία θα χρησιμοποιείται αποκλειστικά και μόνο για το Android.

4.3.4 Google Play store

Πρόκειται για την επίσημη ψηφιακή πλατφόρμα διανομής εφαρμογών για το Android. Λειτουργεί απο τη Google, επιτρέπωντας στους χρήστες να περιηγηθούν και να κατεβάσουν εφαρμογές και στους προγραμματιστές να ανεβάσουν τις εφαρμογές τους, τις οποίες μπορούν να διαθέσουν είτε επί πληρωμής είτε δωρεάν. Για να ανεβάσει κάποιος μια εφαρμογή στο Google Play store θα πρέπει αρχικά να εγγραφεί σε αυτό, κάτι το οποίο κοστίζει 20$. Όταν μια καινούργια εφαρμογή ανέβει για πρώτη φορά, το Google Play store πραγματοποιεί κάποιους ελέγχους για κακόβουλο λογισμικό, έτσι οι χρήστες μπορούν να είναι σίγουροι ότι χρησιμοποιούν ασφαλείς εφαρμογές.

4.3.5 Material design

Τον Νοέμβριου του 2014 η Google ανακοίνωσε την έκδοση 5 του Android με κωδική ονομασία Lollipop.

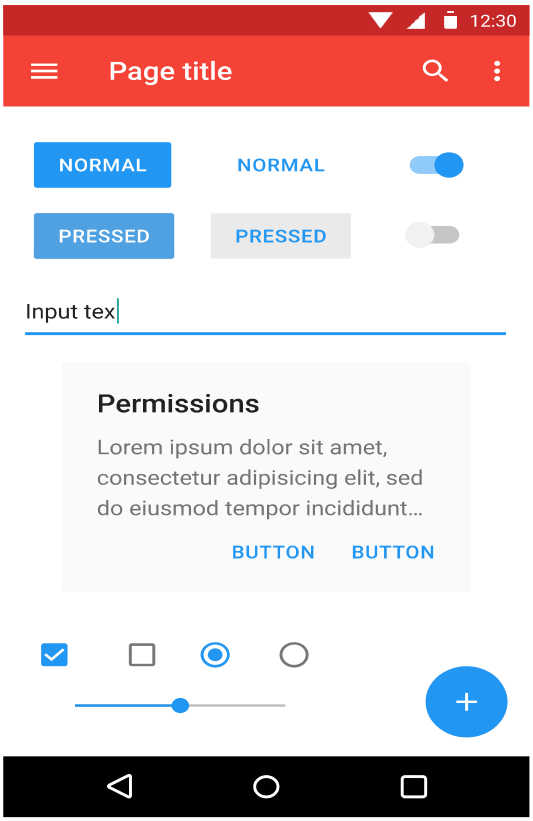
Μεταξύ άλλων χαρακτηριστών προσέφεραι και το Material design, το οποίο απέφερε δραστικές αλλαγές στον όλο σχεδιασμό της διεπαφής χρήστη(UI) στο Android. Η επιτυχία του Material design ήταν τόσο μεγάλη έχωντας σαν αποτέλεσμα να υιοθετηθεί απο πολλές εφαρμογές ανεξαρτήτου πλατφόρμας και λειτουργικού συστήματος.

Πρόκειται για ένα σύνολο σχεδιαστικών κανόνων επεκτείνωντας το μοντέλο των "καρτών", που είναι σχεδιασμός βασισμένος σε διάταξη πίνακα με κινούμενα σχέδια και μεταβάσεις που ανταποκρίνονται στις κινήσεις των χρηστών, καθώς και εφέ βάθους με κατάλληλο φωτισμό και σκίαση αντικειμένων. Ο σχεδιαστής του tias Duarte είχε δηλώσει ' *το Material έχει φυσικές επιφάνειες και άκρα. Οι σκιές δίνουν νόημα σε αυτό που αγγίζει ο χρήστης*'.

Το σύνολο των κανόνων και τεχνικές για έναν γραφιστικό σχεδιασμό μιας εφαρμογής στο Material design μπορούν να βρεθούν στην επίσημη ιστοσελίδα του [16].

Η εφαρμογή των κανόνων του Material Design για διαδικτυακές(Web) εφαρμογές ονομάζεται polymer paper elements και αποτελείται απο τη βιβλιοθήκη Polymer [17].

Παρακάτω ακολουθεί μια εικόνα που παρουσιάζει ένα κομμάτι μιας εφαρμογής η οποια χρησιμοποιεί material design.

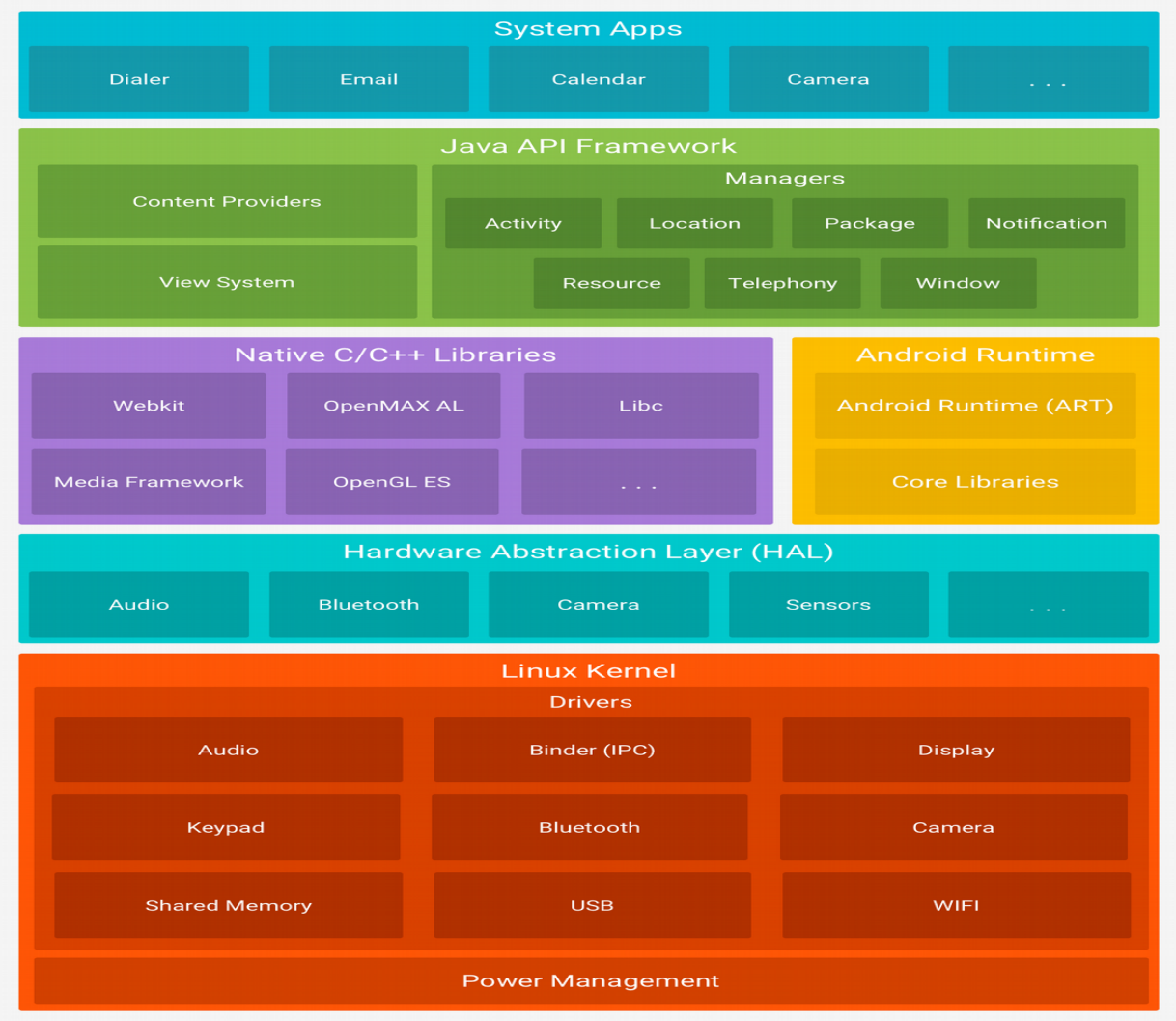


Εικόνα 11: Material design

4.4 Αρχιτεκτονική του Android

Για να καταλάβει κάποιος τον τρόπο με τον οποίο δουλεύει το Android αρκεί να μελετήσει την εικόνα 11 στην οποία φαίνονται τα διαφορετικά στρώματα απο τα οποία αποτελείται το λειτουργικό σύστημα του Android.

Η εικόνα προβάλεται παρακάτω και στην συνέχεια αναλύονται οι βασικοί τομείς στους οποίους χωρίζεται το λειτουργικό σύστημα.



Εικόνα 12: Αρχιτεκτονική του λειτουργικού Android

Οπως φένεται το λειτουργικό σύστημα χωρίζεται σε 6 τομείς και 5 στρώματα. Αναφέρονται συνοπτικά οι 6 βασικοί τομείς και η λειτουργία που επιτελεί ο καθένας.

- **Linux Kernel**

  Είναι ο πυρήνας στον οποίο βασίζεται το λειτουργικό σύστημα Android. Περιέχει όλους τους χαμηλού επιπέδου οδηγούς συσκευών (low level device drivers) που χρειάζονται για τα διαφορα υλικά μέρη της συσκευής καθώς και υποκείμενες λειτουργίες όπως διαχείριση νημάτων(threads) ή μνήμης χαμηλού επιπέδου.

- **Hardware Abstraction Layer**

  Περιέχει συγκεκριμένες διεπαφές(interfaces) που παρέχουν αλληλεπίδραση των υλικών μερών της συσκευής με τα πιο υψηλά επίπεδα.

- **Android Runtime**

  Παρέχει ένα σύνολο από βασικές βιβλιοθήκες που δίνουν τη δυνατοτήτα στους προγραμματιστές να γραψουν εφαρμογές για το Android σε γλώσσα προγραμματισμού Java. Ακόμη περιέχει την εικονική μηχανή (virtual machine) ART(για εκδόσεις του android >= 5) ή Dalvik (για εκδόσεις του android < 5) οι οποίες επιτρέπουν κάθε εφαρμογή να τρέξει σε δική της διαδικασία, με δική της ξεχωριστή μνήμη. Και οι 2 είναι ειδικές εκδόσεις του JVM(Java Virtual Machine) τροποποιημένες για κινητές συσκευές. Η ART περιέχει παραπάνω χαρακτηριστικά από τη Dalvik και πετυχαίνει καλύτερες αποδόσεις.

- **Native C/C++ libraries**

  Εδώ βρίσκεται ο κώδικας όλων των χαρακτηριστών του Android. Πολλά βασικά χαρακτηριστικά και υπηρεσίες του είναι γραμμένες σε C/C++. Το Android παρέχει ένα Java API(Application Programming Interface) για να προσφέρει τη λειτουργία αυτών των χαρακτηριστικών στις εφαρμογές.

- **Java API Framework**

  Όλα τα χαρακτηριστικά του Android(οι προηγούμενες βιβλιοθήκες) γραμμένα στη γλώσσα προγραμματισμού Java.

  Ό,τι χρειάζεται ένας προγραμματιστής για να αναπτύξει μια εφαρμογή βρίσκεται εδώ.

- **System Apps**

  Εδώ βρίσκονται όλες οι εφαρμογές που βρίσκονται στη κινητή συσκευή ή πρόκειται να εγκαταστηθούν σε αυτή.

  Οι εφαρμογές μπορούν να αλληλεπιδρούν μεταξύ τους, για παράδειγμα εάν μια εφαρμογή χρειάζεται να στείλει κάποιο SMS μπορεί να καλέσει την ήδη εγκατεστημένη SMS εφαρμογή για να το στείλει.

4.5 Επίλογος

Συνοψίζοντας, σε αυτό το κεφάλαιο έγινε μια εισαγωγή στο λειτουργικό σύστημα Android προκειμένου ο αναγνώστης να κατανοήσει τι είναι το Android με πολλαπλές αναφορές σε ορισμούς, στα χαρακτηριστικά του αλλά και στην αρχιτεκρονική του από την οποία απαρτίζεται.

5 Προγραμματισμός στο Android

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει μια περιγραφή για το τι κατηγορίες εφαρμογών μπορούν να υπάρξουν στο Android καθώς και μια ανάλυση των συστατικών στοιχείων που μπορούν να χρησιμοποιηθούν σε μια Android εφαρμογή. Πρόκειται για θεμελιώδης έννοιες με τις οποίες εάν εξοικειωθεί ο αναγνώστης, θα είναι σε θέση να ξεκινήσει το προγραμματιστικό του ταξίδι στο κόσμο του Android.

5.2 Κατηγορίες εφαρμογών

Οι τύποι εφαρμογών που μπορούν να αναπτυχθούν στο Android είναι τέσσερεις και αναλύονται παρακάτω.

5.2.1 Εφαρμογές προσκηνίου 

Πρόκειται για το συνηθέστερο είδος εφαρμογών. Ξεκινάνε τη λειτουργία τους όποτε ο χρήστης τις ανοίξει (όπου και ξεκινάνε να εκτελούνται στο προσκήνιο) και η λειτουργία τους αναστέλλεται όταν δεν είναι πλέον ορατές, όταν δηλαδή ο χρήστης μεταφερθεί σε μια άλλη εφαρμογή ή στη κύρια οθόνη(home screen). Όταν μια τέτοια εφαρμογή αναστέλλει τη λειτουργία της σταματάει και ο κύκλος ζωής της, κάτι που σημαίνει πως όταν ο χρήστης ξανά ανοίξει την εφαμοργή αυτή είναι πολύ πιθανό να μην βρίσκεται στην ίδια κατάσταση με αυτή που βρισκόταν πρωτού ανασταλλεί. Οπότε, ο προγραμματιστής είναι υπεύθυνος να αποθηκεύει την κατάσταση της πριν η εφαρμογεί απομακρυνθεί απο το προσκήνιο ώστε αν χρειαστεί, να επανέλθει στην ίδια κατάσταση με πριν. Τέλος, εφόσον πρόκειται για εφαρμογές προσκηνίου θα πρέπει να διαθέτουν μια καλή διεπαφή χρήστη ώστε η χρήση τους να είναι ευχάριστη-εύκολη.

5.2.2 Εφαρμογές παρασκηνίου

Οι εφαρμογές αυτής τη κατηγορίας εκτελούνται στο παρασκήνιο και δεν αλληλεπιδρούν σχεδόν καθόλου με το χρήστη. Η χρήση τους αφορά τον έλεγχο για διάφορα μηνύματα απο ενέργειες που εκτελούνται από το υλικό, το σύστημα ή άλλες εφαρμογές. Για παράδειγμα, υπάρχει μια εφαρμογή παρασκηνίου η οποία ελέγχει/περιμένει για εισερχόμενες κλήσεις προς τη κινητή συσκευή.

5.2.3 Διακοπτόμενες εφαρμογές

Εδώ κατατάσονται οι εφαρμογές οι οποίες χρειάζεται και να αλληλεπιδρούν με το χρήστη μέσω μιας διεπαφής αλλά και να αντιδρούν σε γεγονότα όταν δεν βρίσκονται στο προσκήνιο. Οπότε πρόκειται για μια ένωση εφαρμογών προσκηνίου-παρασκηνίου.

Τέτοιες εφαρμογές θα πρέπει να ενημέρωνουν πάντα το χρήστη, σε όποια κατάσταση και αν βρίσκονται. Είτε με ενημέρωση του UI εάν βρίσκονται στο προσκήνιου είτε με κάποια ειδοποίηση εάν βρίσκονται στο παρασκήνιο. Παράδειγμα τέτοιων εφαρμογών είναι αυτή του Email.

5.2.4 Widgets

Τέτοιες εφαρμογές παρέχουν διαδραστικά γραφικά στοιχεία τα οποία μπορούν να εισαχθούν στην αρχική οθόνη(home screen) ώστε να ενημερώνουν τον χρήστη χωρίς να χρειάζεται να εισέλθει στην εφαρμογή. Για παράδειγμα η στάθμη της μπαταρίας ή η πρόβλεψη καιρού είναι τέτοιες εφαρμογές.

5.3 Συστατικά στοιχεία μιας εφαρμογής

Μετά τους τύπους των εφαρμογών στο Android, θα ήταν σκόπικο να αναλυθούν τα κυριότερα στοιχεία που χρησιμοποιούνται σε μια Android εφαμοργή καθώς και η διατύπωση των όρων.

5.3.1 Context

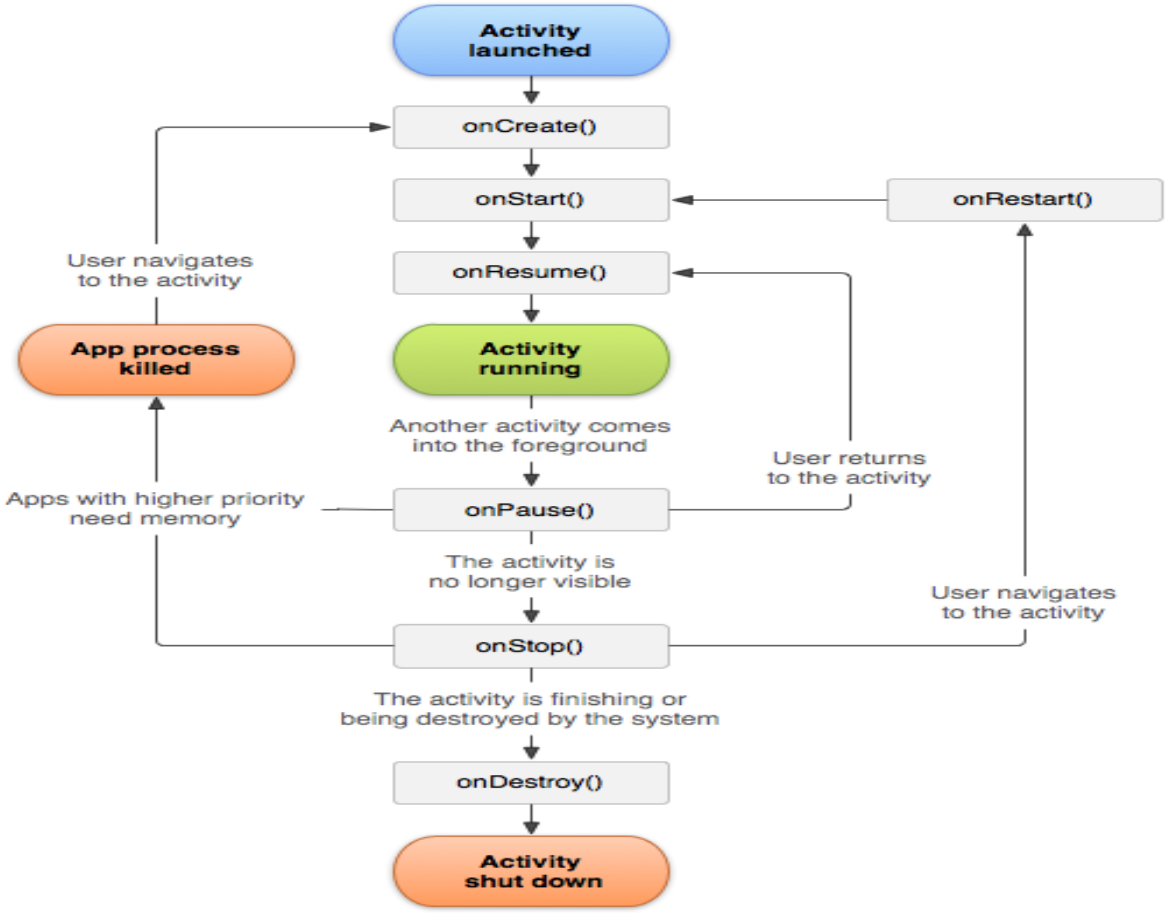
Πρόκειται για μια διεπαφή(interface) που παρέχει πληροφορίες για το περιβάλλον μιας εφαρμογής.

Κάθε εφαρμογή διατηρεί τις κλάσεις και τους πόρους της σε ένα συγκεκριμένο περιβάλλον, οπότε χρησιμοποιώντας το Context μπορούμε να έχουμε πρόσβαση σε αυτά.

5.3.2 Activity

Ένα Activity αναπαρηστά μια οθόνη σε μια διεπαφή χρήστη(UI). Εν συντομία, ένα Activity εκτελεί ενέργειες σε μια οθόνη. Για παράδειγμα, μια εφαρμογή ηλεκτρονικού ταχυδρομείου μπορεί να έχει ένα Activity που παρουσιάζει τα νέα μηνύματα, ένα άλλο που παρουσιάζει τα απεσταλμένα μηνύματα και ένα άλλο που παρουσιάζει τα διαβασμένα μηνύματα. Ένα Activity θα πρέπει να οριστεί ως το πρώτο που θα ξεκινά όταν η εφαρμογή θα εκτελείται. Επίσης, ένα Activity μπορεί να καλέσει κάποιο άλλο, κάθε φορά που ένα νέο ξεκινά το προηγούμενο σταματά και το σύστημα το αποθηκεύει σε μια στοίβα, το λεγόμενο backstack. Έτσι, δίνεται η δυνατότητα στο χρήστη πατώντας το back(πίσω) πλήκτρο να μεταφερθεί στο προηγούμενο Activity.

Κάθε Activity έχει το δικό του κύκλο ζωής ο οποίος παρουσιάζεται στη εικόνα 12 που ακολουθεί. Μελετώντας τη μπορεί να γίνουν κατανοητά όλα τα στάδια στα οποία μπορεί να βρεθεί ένα Activity καθώς και πότε συμβαίνει το καθένα.



Εικόνα 13: Κύκλος ζωής ενός Activity

Για να δημιουργηθεί ένα Activity αρκεί μια κλάση να επεκτείνει την Activity και να ορίσει όπωσδήποτε την onCreate() μέθοδο η οποία αναπαριστά το στάδιο onCreate στο κύκλο ζωής. Η *onCreate()* θα πρέπει να οριστεί αναγκαστικά καθώς το σύστημα την καλεί όταν δημιουργεί το Activity και είναι η μέθοδος στην οποία αρχικοποιούνται τα απαραίτητα στοιχεία του συγκεκριμένου Activity. Η πιο σημαντική κλήση που θα πρέπει να οριστεί στην *onCreate()* είναι η μέθοδος *setContentView()* με την οποία ορίζεται η διάταξη(layout) της διεπιφάνειας χρήστη που θα χρησιμοποιηθεί για το συγκεκριμένο Activity. Εννοείται πως μπορούμε να ορίσουμε και τις υπόλοιπες

μεθόδους που αναπαριστούν τα υπόλοια στάδια ζωής ενός Activity εάν θέλουμε να συμβεί κάτι εκείνη τη στιγμή.

5.3.3 Fragment

Ένα Fragment αναπαριστά μια συμπεριφορά ή ένα τμήμα μιας διεπαφής χρήστη σε ένα Activity. Μπορούν να συνδυαστούν πολλά Fragments σε ένα Activity ώστε να δημιουργήθούν πολλά ανεξάρτητα σημεία τα οποία θα μπορούν να επαναχρησιμοποιηθούν και σε άλλα Activities. Ενα Fragment μπορεί να θεωρηθεί ως ένα τμήμα ενός Activity το οποίο έχει το δικό του κύκλο ζωής, αλληλεπιδρά με το χρήστη και μπορεί να προστεθεί ή να αφαιρεθεί ενώ το Activity εκτελείται.

Ενα Fragment πρέπει να τοποθετείται σε ένα Activity και ο κύκλος ζωής του εποιρεάζεται απο το κύκλο ζωής του Activity. Για παράδειγμα, όταν ένα Activity σταματά να είναι είναι ενεργό (paused) σταματάνε και τα Fragments σε αυτό ή αν ένα Activity καταστραφεί(destroyed) καταστρέφονται και τα αντίστοιχα Fragments.

Στη παρακάτω εικόνα παρουσιάζεται ο κύκλος ζωής ενός Fragment σε σχέση με αυτόν των Acitivities.

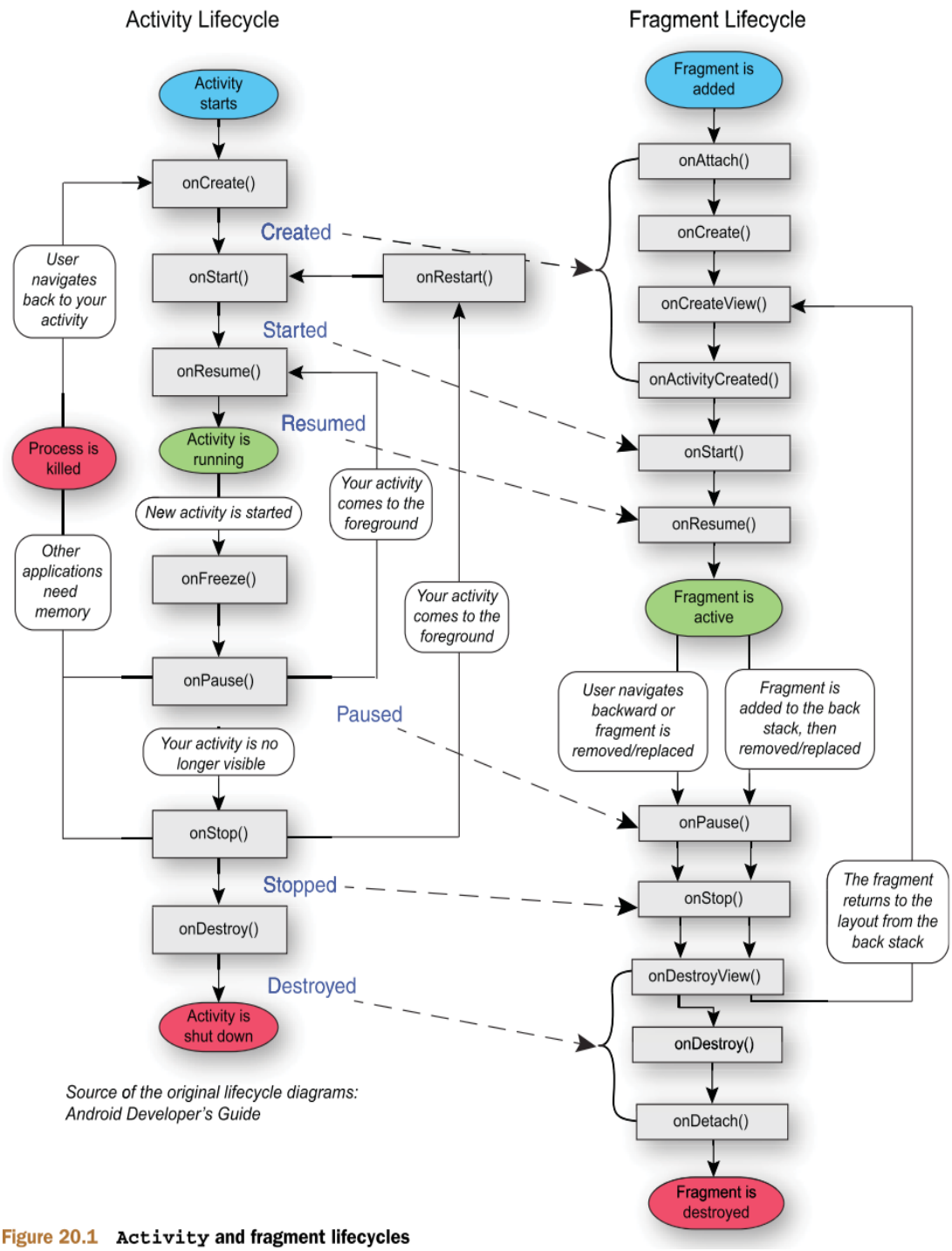


Figure 20.1 **`Activity` and fragment lifecycles**

Εικόνα 14: Κυκλός ζωής ενός Fragment σε σχέση με αυτον του Activity

5.3.4 Manifest

Πρόκειται για ένα XML αρχείο το οποίο είναι απαραίτητο να υπάρχει σε κάθε εφαρμογή. Οταν ο χρήστης εγκαθίστα μια καινούργια εφαρμογή, το πρώτο πράγμα που συμβαίνει στο λειτουργικό σύστημα είναι να διαβάσει το αρχείο Manifest. Σε αυτό περιέχονται διαφορες πληροφορίες της εφαρμογής, όπως όλα τα δικαιώματα(permissions) που απαιτούνται, όλα τα Activities και ποιο θα είναι το πρώτο που θα ξεκινά όταν εκτελείται η εφαρμογή, το όνομα της εφαρμογής κ.α

5.3.5 Intent

Το Android χρησιμοποιεί ένα σύγχρονο μηχανισμό αποστολής/παραλαβής μηνυμάτων ώστε να ανταπεξέλθει σε αιτήσεις εργασιών με το κατάλληλο Activity. Κάθε αίτηση συσκευάζεται ως ένα Intent. Αυτό σημαίνει πως κάθε αίτηση δηλώνει μια πρόθεση να γίνει κάτι, για παράδειγμα έαν απο ένα Activity θέλουμε να ξεκινήσουμε κάποιο άλλο αυτό προγραμματιστικά θα γίνει με την χρήση Intent.

5.3.6 Service

Ενα Service μπορεί να εκτελέσει μακροχρόνιες λειτουργίες στο παρασκήνιο χωρίς να παρέχει τη δυνατότητα για αλληλεπίδραση με τη διεπαφή χρήση(UI). Συστατικά μέρη μιας εφαρμογής μπορούν να ξεκινήσουν ένα service και αυτό να εξακολουθεί να εκτελείται ακόμη και αν ο χρήστης εγκαταληψει τη συγκεκριμένη εφαρμογή(που ξεκίνησε το service). Ενα service για παράδειγμα μπορεί να παίζει μουσική ή να εκτελεί διαδικτυακές συναλλαγές.

5.3.7 Broadcast

Το Android μπορεί να στέλνει ή να δέχεται μηνύματα από το λειτουργικό σύστημα ή από εφαρμογές. Αυτά τα μηνύματα στέλνονται όταν προκύψουν συγκεκριμένα γεγονότα. Για παράδειγμα το λειτουργικό σύστημα στέλνει ένα τέτοιο μήνυμα όταν η συσκευή ξεκινά να φορτίζεται. Οι εφαρμογές μπορούν να στείλουν δικά τους τέτοια μηνύματα για να ενημερώσουν για παράδειγμα άλλες εφαρμογές που ενδιαφέρονται γι’ αυτά.

5.3.8 Threads

Όταν μια εφαρμογή ξεκινά να εκτελείται, το σύστημα δημιουργεί ένα thread(νήμα) για αυτή την εφαμοργή το οποίο ονομάζεται "Main". Αυτό το thread είναι πολύ σημαντικό καθώς όλα τα συστατικά της εφαρμογής δημιουργούνται εδώ όπως και οποιοδήποτε σύστημα κλήσεων προς τα συστατικά αυτά. Για παράδειγμα, όταν ο χρήστης πατήσει ένα κουμπί στην οθόνη, η ενέργεια που θα ενεργοποιήσει το κουμπι θα εκτελεστεί στο "Main" thread. Αυτό δε παρουσιάζει κάποιο πρόβλημα εάν αυτή η ενέργεια δεν είναι χρονοβόρα και γίνεται γρήγορα. Σε αντίθετη περίπτωση όμως, για παράδειγμα μια διαδικτυακή εργασία ή ένα ερώτημα σε μια βάση δεδομένων που μπορεί να διαρκέσει αρκετά, θα μπλοκάρει το Main thread και ο χρήστης δε θα μπορεί να αλληλεπιδρά με το σύστημα μέχρις ότου ολοκληρωθεί η προηγούμενη ενέργεια. Για το λόγο αυτό υπάρχουν και άλλα threads ("background" ή "worker") τα οποία μπορούν να χρησιμοποιηθούν για χρονοβόρες ενέργειες. Προσοχή, μια ενέργεια σε ένα thread διαφορετικό του Main δε μπορει να ενημερώσει τη διεπαφή χρήστη. Οπότε το αποτέλεσμα της ενέργειας θα πρέπει να μεταφέρεται στο Main thread.
Μια λύση σε παραπάνω ζήτημα προσφέρει η AsyncTask που αναλύεται στη συνέχεια.

5.3.9 AsyncTask

Η AsyncTask είναι μια ειδική κλάση η οποία παρέχει επεξεργασία στο παρασκήνιο και συμβάλλει στη διευκόλυνση της επικοινωνίας του Main thread ενώ γινεται διαχείριση του κύκλου ζωής της εργασίας του παρασκηνίου μέσα στο πλαίσιο του κύκλου ζωής του Activity.

Δύο από τις πιο σημαντικές μεθόδους της κλασης AsyncTask είναι η μέθοδος *doInBackground()* η οποία χειρίζεται την επεξεργασία παρασκηνίου ενώ η *publishProgress()* ενημερώνει το Main thread κατά διαστήματα σχετικά με την πρόοδο της επεξεργασίας παρασκηνίου. Όταν η επεξεργασία παρασκηνίου ολοκληρωθεί, η μέθοδος *onPostExecute()* εκτελείται στο Main thread, ώστε να προσφέρει μια τελική ενημέρωση.

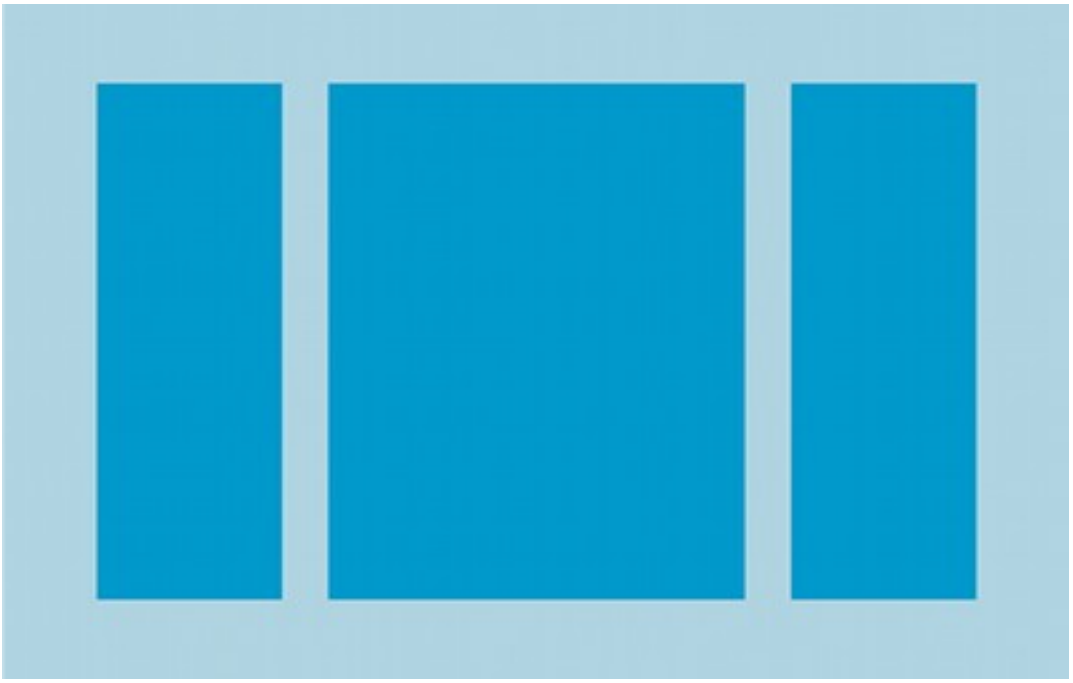
5.3.10 Layouts

Ένα layout ορίζει την οπτική δομή για μια διεπαφή χρήστη σε μια εφαρμογή. Ένα layout μπορεί να οριστεί με δύο τρόπους, είτε μέσω XML αρχείων είτε προγραμματιστικά(κατά τη δημιουργία ενός Activity). Το πλεονέκτημα να οριστούν τα layout απο αρχεία XML είναι ότι κερδίζουμε την αποσύνδεση του layout από τον κώδικα της εφαμοργής. Παρακάτω αναφέρονται μερικά από τα κύρια σχεδιαστικά πρότυπα που χρησιμοποιούνται στην ανάπτυξη μιας Android εφαμοργής.

5.3.10.1 LinearLayout

Μια προβολή LinearLayout στοιχίζει όλα τα στοιχεία της προς μια κατευθυνση, οριζόντια ή κάθετα.

Για παράδειγμα η εικόνα 14 παρουσιάσει ένα LinearLayout που εχει στοιχισμένα τα στοιχεία του οριζόντια το ένα μετά το άλλο.



Εικόνα 15: Παράδειγμα διάταξης LinearLayout

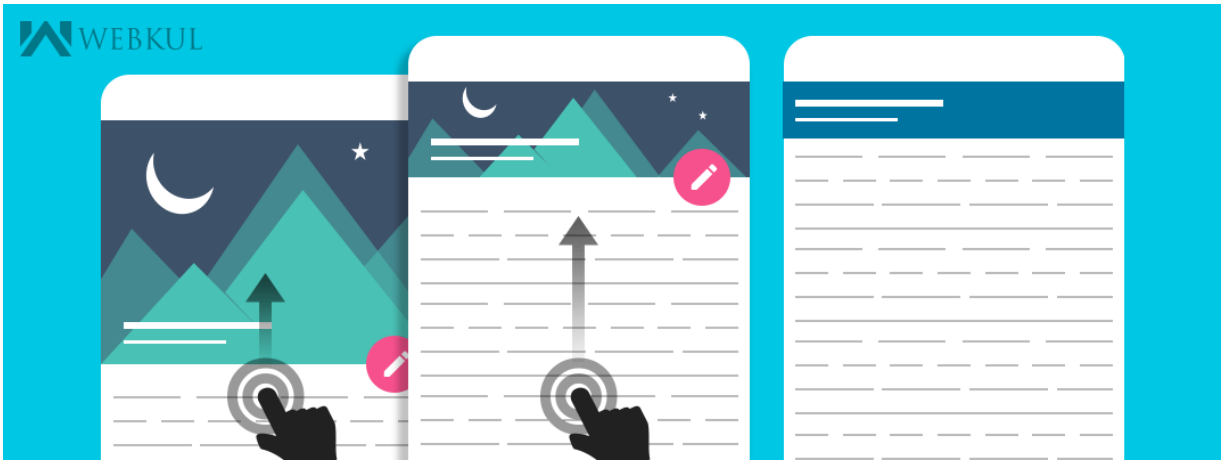
5.3.10.2 RelativeLayout

Μια προβολή RelativeLayout στοιχίζει τα στοιχεία της σε σχετικές θέσεις. Η θέση του κάθε στοιχείου μπορεί να προσδιοριστεί σε σχέση με άλλα στοιχεία της προβολής. Για παράδειγμα "στα αριστερά"(left-of) ή "κάτω"(below) από κάποιο συγκεκριμένο στοιχείο. Επίσης οι θέσεις των στοιχείων μπορούν να σχετίζονται και με όλη τη προβολή, για παράδειγμα ευθυγράμμιση προς τα κάτω(aligned to the bottom). Στη εικόνα 15 που ακολουθεί φένεται ένα RelativeLayout παράδειγμα.

Εικόνα 16: Παράδειγμα διάταξης RelativeLayout

5.3.10.3 CoordinatorLayout

Πρόκειται για μια προβολή η οποία επιτρέπει να αλληλεπιδράνε τα γραφικά στοιχεία της μεταξύ τους. Κάθε στοιχείο μπορεί να έχει μια συμπεριφορά(Behavior), για παράδειγμα ίσως θέλουμε σε μια εφαρμογή όποτε ο χρήστης κατευθύνεται(scroll) προς τα κάτω στην οθόνη το toolbar(αναλύεται παρακάτω) να εξαφανιζέται. Επίσης κάποιο γραφικό στοιχείο μπορεί να ‘προσδεθεί’ με κάποιο άλλο έτσι ώστε όταν συμβεί κάτι στο ένα να συμβεί και στο άλλο. Ένα τέτοιο παράδειγμά παρουσιάζεται στην παρακάτω εικόνα 16. Τα στοιχεία που είναι συνδεδεμένα μεταξύ τους είναι η εικόνα και το στρόγγυλο ροζ κουμπί(FAB). Εάν ο χρήστης κατευθυνθεί προς τα κάτω, η εικόνα αρχίζει και εξαφανίζεται μαζί με το κουμπί, ώσπου κάποια στιγμή εξαφανιζονται τελείως. Εάν ο χρήστης στη συνέχεια κατευθυνθεί προς τα πάνω θα αρχίσουν να εμφανίζονται και πάλι τα δύο στοιχεία.



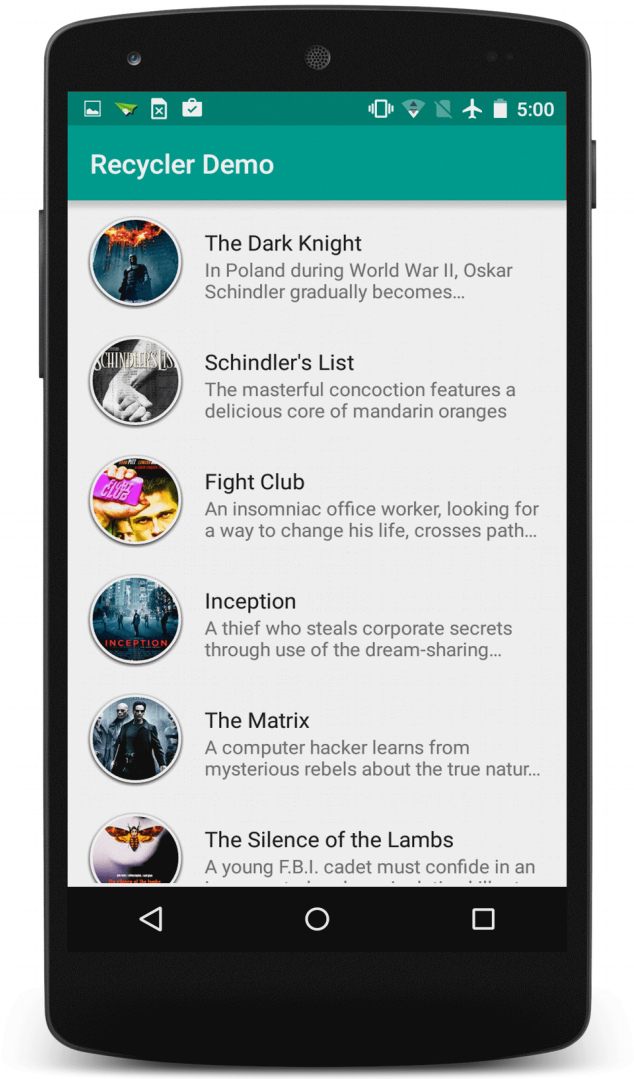
Εικόνα 17: Παράδειγμα CoordinatorLayout

5.3.10.4 RecyclerView

Πολλές εφαρμογές χρειάζεται να εμφανίζουν γραφικά στοιχεία βασισμένα σε μεγάλο όγκο δεδομένων τα οποία αλλάζουν συχνά. Για παράδειγμα, μια εφαρμογή μουσικής μπορεί να παρουσιάζει πληροφορίες σχετικά με χιλιάδες albums αλλά μόνο κάποια απο αυτά κάθε φορά στην οθόνη. Εάν η εφαρμογή δημιουργούσε γραφικά στοιχεία για κάθενα από αυτά θα χρειαζόταν πολύ μνήμη και αποθηκευτικό χώρο και συνεπώς θα ήταν επιρρεπής στο να σταματήσει η λειτουργία της κάποια στιγμή. Από την άλλη, εάν η εφαρμογή δημιουργούσε ένα γραφικό στοιχείο κάθε φορά που ένα νέο album μετακινούνταν(scrolled) στην οθόνη και κατέστρεφε το προηγούμενο, αυτό θα είχε πολύ καλύτερα αποτελέσματα στην επίδοση και στην μνήμη.

Για την αντιμετώπιση αυτής της κατάστασης το Android παρέχει(μέσω του Android Support Library) το RecyclerView το οποίο χρησιμοποιειται για δημιουργία λιστών χρησιμοποιώντας τον αποτελεσματικό τρόπο που περιγράψαμε παραπάνω.

Η εικόνα 17 που ακολουθεί παρουσιάζει ένα παράδειγμα χρήσης του RecyclerView.



Εικόνα 18: Παράδειγμα RecyclerView

5.3.10.5 Dialogs

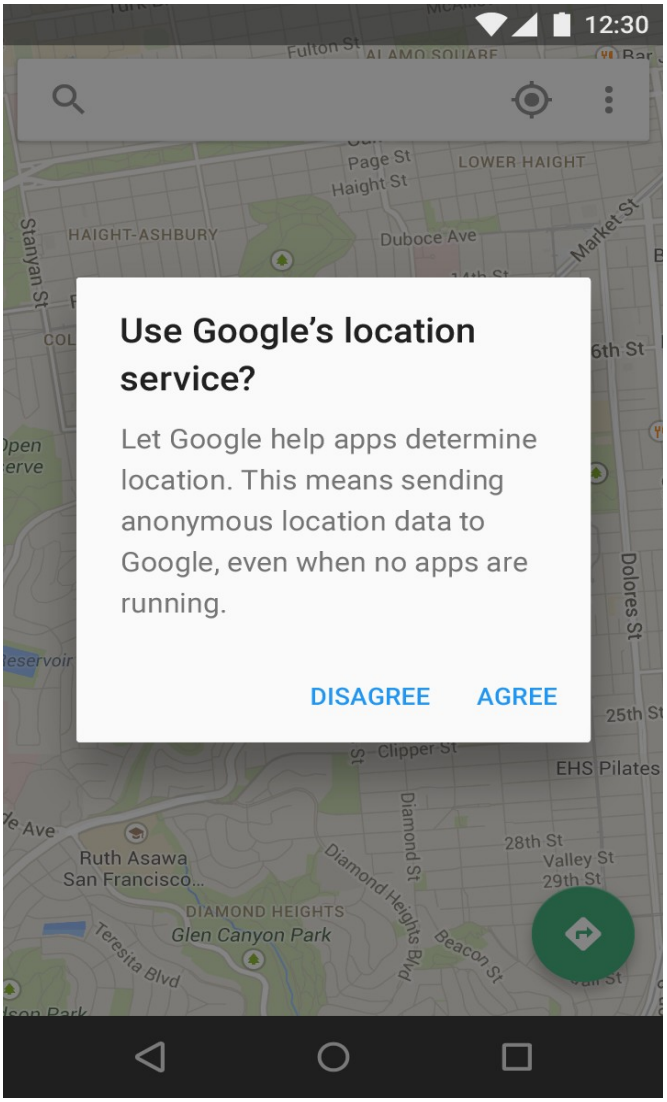
Ένα Dialog(διάλογος) είναι ένα μικρό παράθυρο το οποίο προτρέπει τον χρήστη να πάρει μια απόφαση ή να εισάγει επιπρόσθετες πληροφορίες. Ένα Dialog δεν γεμίζει όλη την οθόνη και συνήθως χρησιμοποιείται όταν ο χρήστης πρέπει να κάνει κάποια ενέργεια για να μπορέσει να συνεχίσει. Τα πιο βασικά είδη διαλόγου είναι:

- **AlertDialog**: ένα dialog που περιέχει έναν τίτλο, μέχρι τρία κουμπιά, μια λίστα με κάποια επιλεγμένα στοιχεία ή μια τελείως προσαρμοσμένη διάταξη(custom layout).

- **DatePickerDialog**: Ένα dialog με μια προκαθορισμένη διεπαφή χρήστη που επιτρέπει στον χρήστη να επιλέξει ημερομηνία ή ώρα.

- **Progress Dialog**

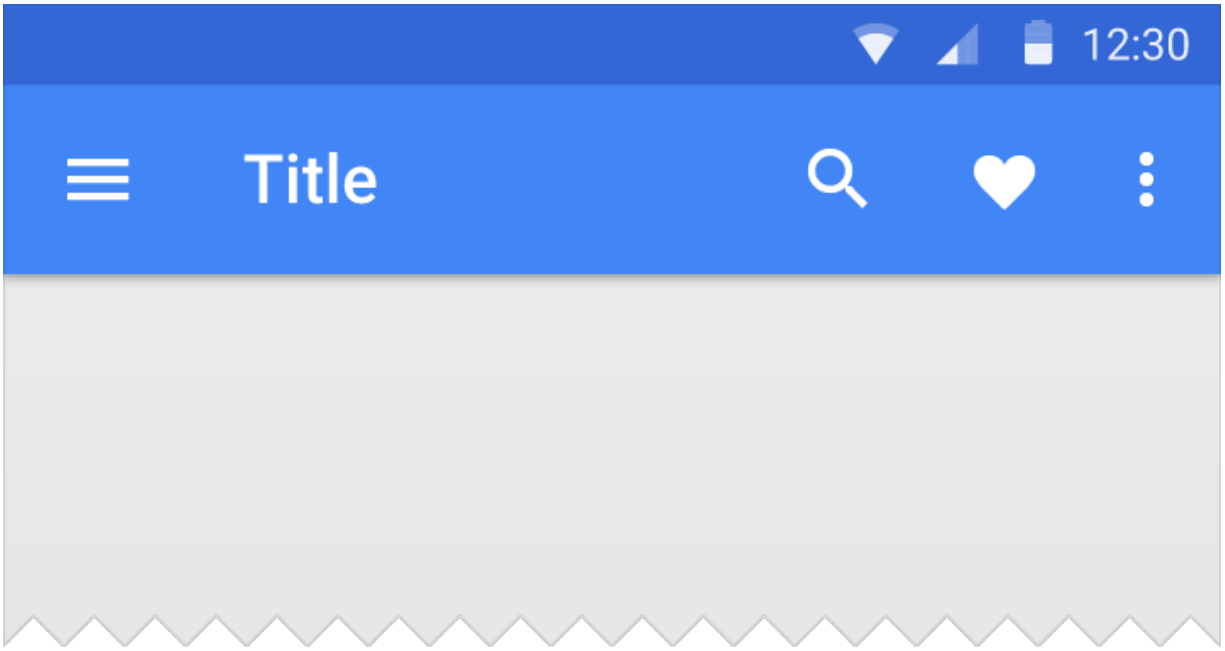
  Ένα Progress Dialog χρησιμοποιείται για να δείξει στον χρηστη την πρόοδο μιας διαδικάσίας. Όταν εκτελούνται χρονοβόρες διαδικάσίες για να μη βλέπει ο χρήστης μια παγωμένη οθόνη, χρησιμοποιείται το Progress Dialog που ενημερώνει τον χρήστη για την πρόοδο της διαδικάσίας. Στην είκονα 18 που ακολουθει παρουσιάζεται ένα AlertDialog.



Εικόνα 19: Παράδειγμα Alert Dialog

5.3.10.6 Toolbar

Το toolbar είναι ένα στοιχείο που προσδιορίζει τη θέση του χρήστη και περιέχει διαθέσιμες ενέργειες και τρόπους πλοήγησης. Χρησιμοποιώντας ένα toolbar προσφέρεται στους χρήστες ένα οικείο περιβάλλον σε όλες τις εφαρμογές στο οποίο το σύστημα προσαρμόζεται χάρη στις διαφορετικές διαμορφώσεις οθόνης. Κάνει σημαντικές δράσεις εμφανέστατες και προσβάσιμες με έναν προβλέψιμο τρόπο. Στην εικόνα 19 που ακολουθεί παρουσιάζεται ένα παράδειγμα ενός toolbar.



Εικόνα 20: Παράδειγμα Toolbar

5.4 Επίλογος

Συνοψίζοντας, έγινε μια προσπάθεια ο αναγνώστης να κατανοήσει τα βασικά συστατικά στοιχεία μιας Android εφαρμογής τα οποία αποτελούν το θεμέλιο για να ξεκινήσει κάποιος να προγραμματίζει στο Android.

Παρακάτω θα δούμε την αρχιτεκτονική και τις τεχνολογίες που χρησιμοποιήθηκαν για τη εφαρμογή της πτυχιακής εργασίας όπως και την αναλυτική περιγραφή της.

6 Αρχιτεκτονική και τεχνολογίες που χρησιμοποιήθηκαν

6.1 Εισαγωγή

Στο παρόν κεφάλαιο θα περιγραφεί η αρχιτεκτονική, η δομή των φακέλων μαζί με τα αρχεία που περιέχουν καθώς και οι τεχνολογίες(γλώσσες προγραμματισμού, βιβλιοθήκες λογισμικού, προγραμματιστικά περιβάλλοντα) που χρησιμοποιήθηκαν για να υλοποιηθεί η Android εφαρμογή.

6.2 Αρχιτεκτονική

Για την αρχιτεκτονική της εφαρμογής ώστε αυτή να γίνει εύκολα διαχειρίσιμη, ευέλικτη και διατηρήσιμη χρησιμοποιήθει το MVP, το οποίο περιγράφεται παρακάτω.

6.2.1 Τι είναι το MVP

Το MVP(Model View Presenter) είναι ένα παράγωγο του γνωστού MVC(Model View Controller) το οποίο επιτρέπει τον διαχωρισμό του επιπέδου παρουσίασης(UI) από αυτό της λογικής(business logic). Οπότε, το πως θα υλοποιήσουμε τη λειτουργικότητα ενός συστατικού της εφαρμογής μας είναι ξεχωριστό απο το πως θα το παρουσιάσουμε στην οθόνη.

6.2.2 Γιατί MVP

Ένα πρόβλημα που παρατηρήθηκε στην ανάπτυξη Android εφαρμογών είναι το γεγονός ότι τις περισσότερες φορές τα Activities είναι στενά συνδεδεμένα με το γραφικό περιβάλλον αλλά και με τους μηχανισμούς για τα δεδομένα, κάτι που έχει ως αποτέλεσμα ένα Activity να καταλήγει να έχει το ρόλο ενός God Object [18].
Σίγουρα τα XML αρχεία που χρησιμοποιούνται για τη γραφική αναπαράσταση είναι ένα θετικό αλλά σε μεγάλα projects δεν αρκούν μόνο αυτά.

Τι θα γίνει για παράδειγμα στην εφαρμογή μας εάν κάποια στιγμή αντί να πέρνουμε τα δεδομένα απο τη βάση δεδομένων χρειαστεί να τα πέρνουμε από ένα REST API? Θα έπρεπε να τροποποιήσουμε πολλά κομμάτια του ήδη υπάρχων κώδικα.
Το MVP ανεξαρτητοποιεί τη διεπαφή χρήστη απο τα δεδομένα χωρίζοντας την εφαρμογή στα τρία παρακάτω στρώματα: *Model*, *View*, *Presenter.*

6.2.3 Ανάλυση MVP

- **Model**
  Πρόκειται για απλές Java κλάσεις (POJO's) οι οποίες αναπαριστούν τα μοντέλα του business logic της εφαρμογής.

- **View**
  Συνήθως πρόκειται για μια διεπαφή(interface) η οποία περιέχει όλες τις μεθόδους που εφαρμόζουν μια αλλαγή στη διεπαφή χρήστη.
  Για παράδειγμα, μπορεί να περιέχει τις εξής μεθόδους:
  showDialog(), *hideToolbar()* οι οποίες θα δείχνουν έναν Dialog και θα κρύβουν το Toolbar αντίστοιχα.

- **Presenter**
  Είναι ο διαμεσολαβητής μεταξύ του Model και του View. Πέρνει δεδομένα απο το Model και τα επιστρέφει μορφοποιημένα στο View για να τα παρουσιάσει. Αντίθετα όμως με τον Controller του MVC είναι αυτός που αποφασίζει ποια ενεργεια θα συμβεί όταν ο χρήστης αλληλεπιδρά με τα γραφικά στοιχεία.

6.2.4 Σύνδεση

Είδαμε για τι είναι υπεύθυνο κάθε επίπεδο του MVP ξεχωριστά και πως λειτουργεί. Η απορροία που δημιουργήται είναι πως συνδέονται αυτά τα τρία επίπεδα μεταξύ τους. Όπως αναφέραμε, ένα View είναι μια διεπαφή που περιέχει μεθόδους οι οποίες τροποποιούν γραφικά στοιχεία. Αυτή η διεπαφή λοιπόν μπορεί να υλοποιείται από ένα Activity ή Fragment. Οπότε, ένα Activity ή ένα Fragment ο μόνος ρόλος που έχουν μέχρι στιγμής αφορά την υλοποίηση των μεθόδων του View.

Αναφέρθηκε επίσης ότι ο Presenter πέρνει δεδομένα απο το Model, άρα μπορούμε στην κλάση του Present να εισάγουμε αναφορές στα αντικείμενα του Model ώστε να μπορεί να αλληλεπιδρά μαζί τους.

Είδαμε όμως ότι ο Present έχει και το ρόλο του να αποφασίζει ποιά ενέργεια θα συμβεί όταν ο χρήστης αλληλεπιδρά με γραφικά στοιχεία. Άρα μπορούμε να εισάγουμε στο Activity ή Fragment έναν Presenter τον οποίο το Activity/Fragment θα καλεί για κάθε ενέργεια που μπορεί να κάνει ο χρήστης.

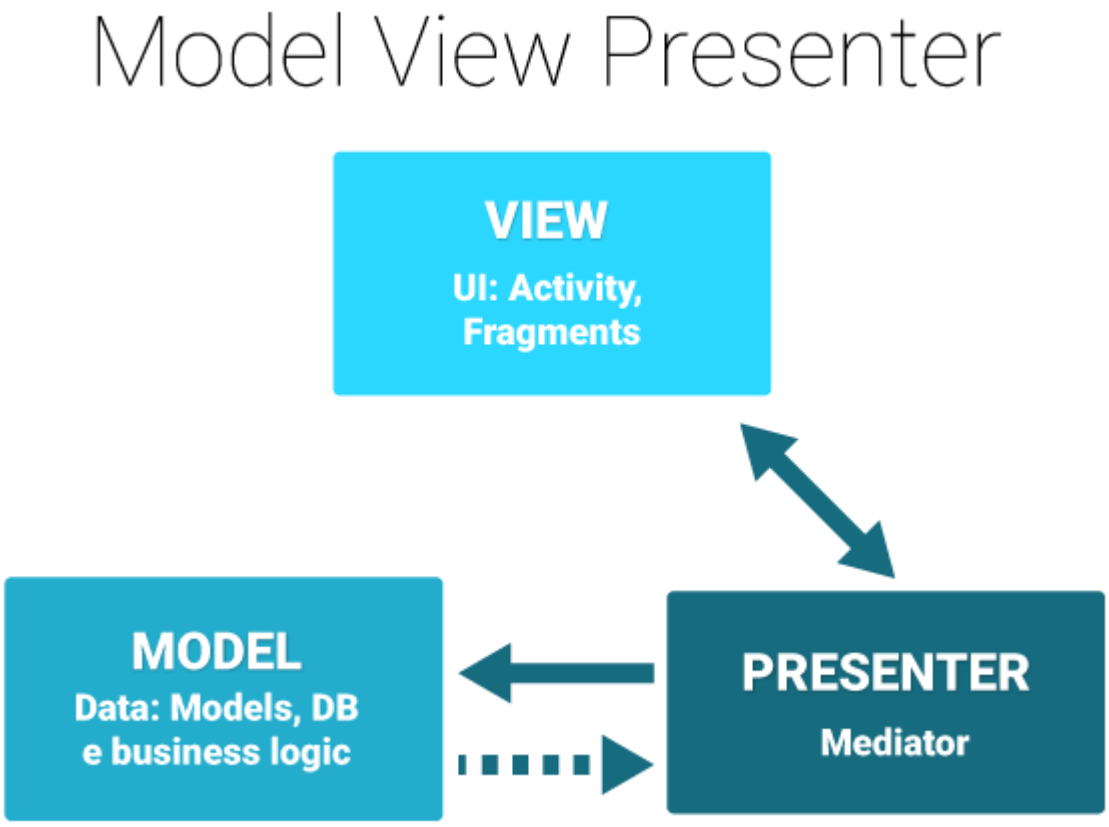
Για να γίνει πιο κατανοητό ας δούμε ένα παράδειγμα, έστω η εφαρμογή μας αποτελείται από ένα Activity η οποία στην διεπαφή χρήστη περιλαμβάνει ένα κουμπί, το οποίο όταν πατηθεί εμφανίζεται ένα Dialog με κάποιες πληροφορίες.

Χρησιμοποιώντας MVP για το παραπάνω η υλοποιηση θα είναι η εξής:

Μια διεπαφή(View) η οποία θα περιέχει τη μέθοδο ~~για~~ για να εμφανίζεται το Dialog. Μια κλάση(Model) η οποία θα περιέχει τις πληροφορίες που θελουμε να εμφανίσουμε στο dialog και μια κλάση(Presenter) η οποία θα έχει μια αναφορά στη κλάση του model και μια στη διεπαφή του View.

Το Activity υλοποιεί την διεπαφή του View και έχει μια αναφορά στη κλάση του Presenter. Όταν ο χρήστης πατήσει το κουμπί, το Activity θα καλέσει τον Presenter ώστε να πάρει τα δεδομένα απο το Model και αυτό με τη σειρά του να καλέσει το View ώστε να εμφανιστεί το dialog με τις πληροφορίες.

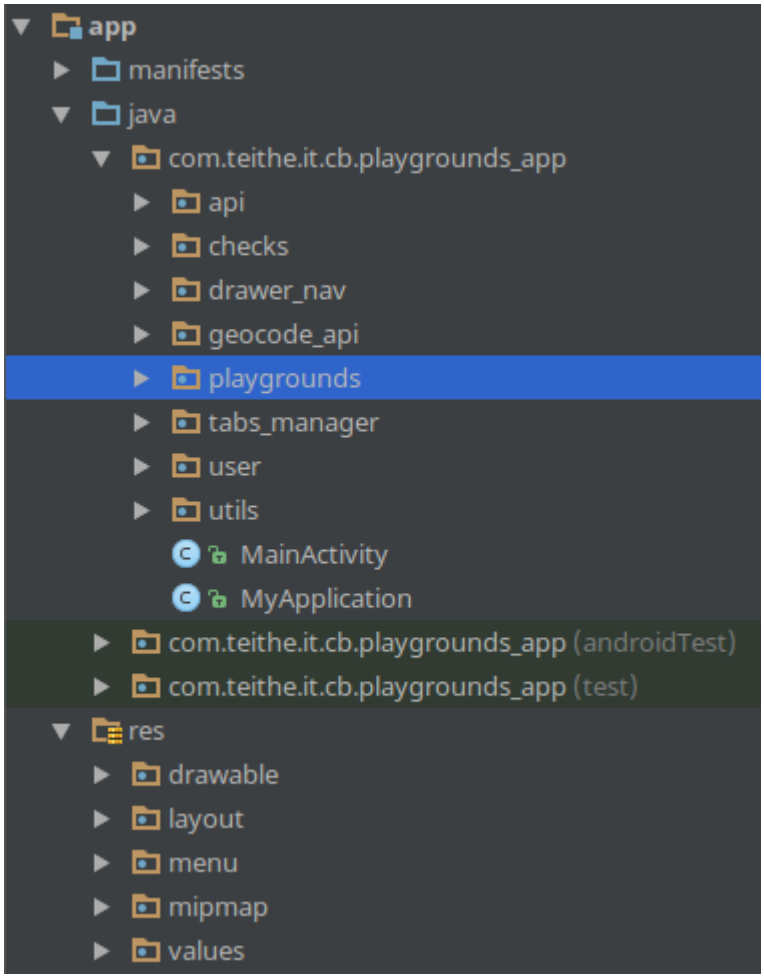
Ακολουθεί και μια εικόνα που παρουσιάζει το MVP.



Εικόνα 21: Λειτουργία MVP

6.3 Δομή του project στο Android Studio

Η δομή της εφαρμογής αποτελείται από φακέλους και διάφορα αρχεία σε μορφή δέντρου, όπως και κάθε Android εφαρμογή. Πιο συγκεκριμένα έγινε μια προσπάθεια να εφαρμοστεί η ταξινόμηση των αρχείων σε φακέλους βάση χαρακτηριστικών(package by feature). Η συγκεκριμένη εφαρμογή αναπτύχθηκε στο περιβάλλον Android Studio και η δομή της απεικονίζεται παρακάτω:



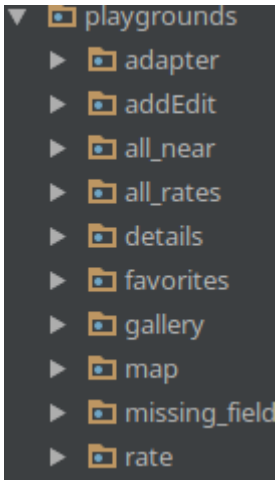
Εικόνα 22: Δομή της εφαρμογής στο Android Studio

Συγκεκριμένα βλέπουμε τους εξείς φακέλους(πακέτα):

- ***/app:***
  Ριζικός φάκελος που περιλαμβάνει όλους τους υποφακέλους του project.

- ***/app/manifest:***
  Περιλαμβάνει το απαραίτητο manifest.xml αρχείο.

- **/app/java:**
  Περιέχει τους φακέλους με τις κλάσεις της εφαρμογής καθώς και τις κλάσεις για junit και integration tests. Ο πρώτος *com.teithe.it.cb.playgrounds_app* φάκελος περιλαμβάνει σχεδόν όλη τη λειτουργικότητα της εφαρμογής. Βλέπουμε ότι αποτελείται απο επιμέρους φακέλους οι οποίοι κάθε ένας αφορά και μια λειτουργία της εφαρμογής. Εάν επεκτείνουμε το *playgrounds* φάκελο για παράδειγμα, θα δούμε άλλους φακέλους με λειτουργίες-χαρακτηριστικά που αφορούν τους παιδότοπους.



Εικόνα 23: Υποφάκελοι-χαρακτηριστικά του Playgrounds

- ***/app/res***

  Σε αυτόν το φάκελο αποθηκεύονται όλοι οι πόροι(resources) της εφαρμογής.

- **/app/res/drawable**

  Σε αυτό το φάκελο αποθηκεύονται όλα τα γραφικά της εφαρμογής. Με τον όρο γραφικά αναφερόμαστε σε κάθε τύπου εικόνας αλλά και σε αρχεία XML που αφορούν τα γραφικά συστατικά της εφαρμογής.

- ***/app/res/layout***

  Σε αυτόν τον φάκελο αποθηκεύονται αρχεία XML τα οποία σχετίζονται με το γραφικό περιβάλλον της εφαρμογής.

- ***/app/res/menu***

  Σε αυτό το φάκελο περιέχονται τα menu που χρησιμοποιεί η εφαρμογή μορφοποιημένα σε XML.

- **/app/res/values**

  Στον φάκελο values αποθηκεύονται κατάλληλα αρχεία που περιέχουν ειδικές μεταβλητές που χρησιμοποιεί η εφαρμογή, όπως τιμές χρωμάτων και αλφαριθμητικών με συγκεκριμένα ονόματα ή αρχεία με σύνολα κανόνων(styles) για την εμφάνιση συγκεκριμένων γραφικών στοιχείων.

6.4 Τεχνολογίες που χρησιμοποιήθηκαν

Σε αυτό το μέρος του κεφαλαίου θα αναπτυχθούν αναλυτικά οι γλώσσες προγραμματισμού, τα προγραμματιστικά περιβάλλοντα καθώς και διάφορες τεχνολογίες που χρησιμοποιήθηκαν για τη δημιουργία της εφαρμογής.

6.4.1 Java

Η Java είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού που σχεδιάστηκε από την εταιρεία πληροφορικής Sun Microsystems. Ένα από τα βασικά πλεονεκτήματα της Java έναντι των περισσότερων άλλων γλωσσών είναι η ανεξαρτησία του λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε Java τρέχουνε ακριβώς το ίδιο σε Windows, Linux, Unix και MacOS χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) αλλά και λειτουργικού συστήματος (Windows, Unix, Linux, MacOS). Ο λόγος είναι ότι κάθε κεντρική μονάδα επεξεργασίας κατανοεί διαφορετικό κώδικα μηχανής. Ο συμβολικός κώδικας (assembly) που μεταφράζεται και εκτελείται σε Windows είναι διαφορετικός από αυτόν που μεταφράζεται και εκτελείται σε έναν υπολογιστή Macintosh. Η λύση δόθηκε με την ανάπτυξη της Εικονικής Μηχανής (Virtual Machine ή VM). Αφού γραφεί κάποιο πρόγραμμα σε Java, στη συνέχεια μεταγλωτίζετται μέσω του μεταγλωττιστή javac, ο οποίος παράγει έναν αριθμό από αρχεία .class (κώδικας byte ή bytecode). Ο κώδικας byte είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττιστεί. Όταν πρόκειται να εκτελεστεί η εφαρμογή σε ένα μηχάνημα, το Java Virtual Machine που πρέπει να είναι εγκατεστημένο σε αυτό θα αναλάβει να διαβάσει τα αρχεία .class. Στη συνέχεια τα μεταφράζει σε γλώσσα μηχανής που να υποστηρίζεται από το

λειτουργικό σύστημα και τον επεξεργαστή, έτσι ώστε να εκτελεστεί (αυτό συμβαίνει με την παραδοσιακή Εικονική Μηχανή (Virtual Machine). Πιο σύγχρονες εφαρμογές της εικονικής Μηχανής μπορούν και μεταγλωττίζουν εκ των προτέρων τμήματα bytecode απευθείας σε κώδικα μηχανής (εγγενή κώδικα ή native code) με αποτέλεσμα να βελτιώνεται η ταχύτητα). Χωρίς αυτό δε θα ήταν δυνατή η εκτέλεση λογισμικού γραμμένου σε Java. Η JVM είναι λογισμικό που εξαρτάται από την πλατφόρμα, δηλαδή για κάθε είδος λειτουργικού συστήματος και αρχιτεκτονικής επεξεργαστή υπάρχει διαφορετική έκδοση του. Έτσι υπάρχουν διαφορετικές JVM για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδομηχανές κλπ.

Για να να γράψει κάποιος κώδικα Java δε χρειάζεται τίποτα άλλο παρά έναν επεξεργαστή κειμένου. Παρ' όλ' αυτά, ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) βοηθάει πολύ, ιδιαίτερα στον εντοπισμό σφαλμάτων (debugging). Υπάρχουν αρκετά διαθέσιμα, ενώ πολλά από αυτά έρχονται δωρεάν.

6.4.2 XML

Η XML (eXtensible Markup Language) είναι μία γλώσσα σήμανσης, που περιέχει ένα σύνολο κανόνων για την ηλεκτρονική κωδικοποίηση κειμένων. Ορίζεται, κυρίως, στην προδιαγραφή XML 1.0 (XML 1.0 Specification), που δημιούργησε ο διεθνής οργανισμός προτύπων W3C (World Wide Web Consortium), αλλά και σε διάφορες άλλες σχετικές προδιαγραφές ανοιχτών προτύπων.

Η XML σχεδιάστηκε δίνοντας έμφαση στην απλότητα, τη γενικότητα και τη χρησιμότητα στο Διαδίκτυο. Είναι μία μορφοποίηση δεδομένων κειμένου, με ισχυρή υποστήριξη Unicode για όλες τις γλώσσες του κόσμου. Αν και η σχεδίαση της XML εστιάζει στα κείμενα, χρησιμοποιείται ευρέως για την αναπαράσταση αυθαίρετων δομών δεδομένων, που προκύπτουν για παράδειγμα στις υπηρεσίες ιστού.

Υπάρχει μία ποικιλία διεπαφών προγραμματισμού εφαρμογών, που μπορούν να χρησιμοποιούν οι προγραμματιστές, για να προσπελαύνουν δεδομένα XML, αλλά και διάφορα συστήματα σχημάτων XML, τα οποία είναι σχεδιασμένα για να βοηθούν στον ορισμό γλωσσών, που προκύπτουν από την XML.

6.4.2.1 Βασική ορολογία

Τα βασικά στοιχεία που θα συναντήσει κάποιος στην χρήση της XML είναι τα παρακάτω:

- **Χαρακτήρας Unicode**

  Εξ ορισμού, ένα κείμενο XML είναι μία ακολουθία χαρακτήρων. Σχεδόν κάθε χαρακτήρας Unicode μπορεί να εμφανίζεται σε ένα κείμενο XML.

- **Επεξεργαστής και εφαρμογή**

  Είναι το λογισμικό που επεξεργάζεται ένα κείμενο XML. Είναι αναμενόμενο, ότι ένας επεξεργαστής δουλεύει για μία εφαρμογή. Υπάρχουν μερικές πολύ συγκεκριμένες απαιτήσεις, σχετικά με το τι μπορεί και τι δεν μπορεί να κάνει ένας επεξεργαστής XML, αλλά καμία, όσον αφορά στη συμπεριφορά της εφαρμογής. Ο επεξεργαστής (όπως ονοματίζεται από την προδιαγραφή), αναφέρεται συχνά, με τον αγγλικό όρο XML parser.

- **Σήμανση και περιεχόμενο**

  Οι χαρακτήρες που απαρτίζουν ένα κείμενο XML, αποτελούν είτε τη σήμανση είτε το περιεχόμενό του. Η σήμανση και το περιεχόμενο, μπορούν να επισημανθούν και να διακριθούν, ύστερα από την εφαρμογή κάποιων απλών συντακτικών κανόνων. Όλα τα αλφαριθμητικά που συνιστούν τη σήμανση, είτε ξεκινούν με το χαρακτήρα "<" και καταλήγουν στο χαρακτήρα ">", είτε ξεκινούν με το χαρακτήρα "&" και καταλήγουν στο χαρακτήρα ";". Ακολουθίες χαρακτήρων που δε συνιστούν τη σήμανση, αποτελούν το περιεχόμενο ενός κειμένου XML.

- **Ετικέτα**

  Ένα στοιχείο σήμανσης που ξεκινά με το χαρακτήρα "<" και καταλήγει στο χαρακτήρα ">". Υπάρχουν τρία είδη ετικέτας: ετικέτες-αρχής, για παράδειγμα *<section>*, ετικέτες-τέλους, για παράδειγμα *</section>*, και ετικέτες-χωρίς-περιεχόμενο, για παράδειγμα *<line-break/>*.

- **Στοιχείο**

  Ένα λογικό απόσπασμα ενός κειμένου, που είτε ξεκινά με μία ετικέτα-αρχής και καταλήγει σε μία ετικέτα-τέλους, είτε αποτελείται μόνο από μία ετικέτα-χωρίς-περιεχόμενο. Οι χαρακτήρες που υπάρχουν, αν υπάρχουν, μεταξύ μιας ετικέτας-αρχής και μιας ετικέτας-τέλους, συνιστούν το περιεχόμενο του στοιχείου, το οποίο μπορεί να περιέχει σήμανση, συμπεριλαμβανομένων και άλλων στοιχείων, που ονομάζονται στοιχεία-παιδιά. Ένα παράδειγμα ενός στοιχείου είναι το *<Greeting>Hello, world.</Greeting>*.

- **Χαρακτηριστικό**

  Ένα στοιχείο σήμανσης που αποτελείται από ένα ζευγάρι όνομα/τιμή, το οποίο υπάρχει μέσα σε μία ετικέτα-αρχής ή σε μία ετικέτα-χωρίς-περιεχόμενο. Στο παράδειγμα παρακάτω, το στοιχείο img έχει δύο χαρακτηριστικά, τα *src* και *alt*: **. Ένα άλλο παράδειγμα θα ήταν το *<step number="3">Connect A to B.</step>*, όπου το όνομα του χαρακτηριστικού είναι "number" και η τιμή του είναι "3".

- **Δήλωση XML**

  Τα κείμενα XML μπορούν να αρχίζουν, με τη δήλωση κάποιων πληροφοριών σχετικών με αυτά, όπως στο ακόλουθο παράδειγμα:
  <?xml version="1.0" encoding="UTF-8"?>

Παράδειγμα:

```
<?xml version="1.0" encoding='UTF-8'?>
<painting>
  <img src="myImage.jpg" alt='My fantastic portrait, by Chris'/>
  <caption>This is a fantastic portrait, painted in
  <date>2015</date>-<date>2016</date>.</caption>
</painting>
```

6.4.3 Facebook SDK

Το Facebook SDK για Android προσφέρει τη δυνατότητα στους προγραμματιστές να επιτρέπουν στους χρήστες να συνδέονται στις εφαρμογές τους μέσω του Facebook. Παρακάτω αναλύονται τα βήματα που πρέπει να ακολουθήσουμε για να χρησιμοποιήσουμε το Facebook SDK στην εφαρμογή μας.

6.4.3.1 Εγγραφή σαν προγραμματιστής στο Facebook

Κάνωντας σύνδεση με τον Facebook λογαριασμό μας στην ακόλουθη ιστοσελίδα <https://developers.facebook.com> δηλωνώμαστε ως προγραμματιστές στο facebook και μπορούμε να χρησιμοποιήσουμε τα διαθέσιμα εργαλεία του.

6.4.3.2 Προσθήκη νέας εφαρμογής

Μέσω της παραπάνω ιστοσελίδας, θα πρέπει να δημιουργήσουμε μια νέα εφαρμογή η οποία θα περιέχει στοιχεία της Android εφαρμογής ώστε το Facebook να γνωρίζει με ποια συγκεκριμένη εφαρμογή θα αλληλεπιδρά. Πιο συγκεκριμένα, εφόσον δημιουργηθεί η εφαρμογή θα πρέπει να προστεθεί σε ποια πλατφόρμα θα τη χρησιμοποιήσουμε. Αυτή είναι το Android, για το οποίο τα απαραίτητα στοιχεία που ζητά να γνωρίζει το Facebook είναι το όνομα του κύριου πακέτου της Android εφαρμογής, το όνομα της κλάσης του αρχικου Activity(το πρώτο που ξεκινά όταν εκτελείται η εφαρμογή) καθώς και το key hash.

Το key hash πρόκειται για ένα μοναδικό 'κλειδι' που περιέχει κάθε Android εφαρμογή. Εισάγωντας τη παρακάτω γραμμή σε ένα παράθυρο γραμμής εντολών θα πάραχθει ένα key hash το οποίο μπορεί να χρησιμοποιηθεί κατά την ανάπτυξη της εφαρμογής, πρόκειται δηλαδή για ένα Development key hash.

keytool -exportcert -alias androiddebugkey -keystore %HOMEPATH %\.android\debug.keystore | openssl sha1 -binary | openssl base64

Όταν η εφαρμογή είναι έτοιμη για διάθεση στο κοινό θα πρέπει να παραχθεί ένα νέο key hash, το release key hash, με την παρακάτω εντολή:

keytool -exportcert -alias YOUR_RELEASE_KEY_ALIAS -keystore YOUR_RELEASE_KEY_PATH | openssl sha1 -binary | openssl base64

Τέλος θα πρέπει να ενεργοποιήσουμε την επιλογή *Single Sign On* που υπάρχει στην ιστοσελίδα.

6.4.3.3 Εισαγωγή του Facebook SDK στην Android εφαρμογή

Αυτό μπορεί να γίνει εύκολα μέσω του Gradle. Κάθε νέα Android εφαρμογή που δημιουργήται στο Android Studio περιλαμβάνει εξ αρχής δύο *build.gradle* αρχεία. Το ένα αφορά ολόκληρο την εφαρμογή(Project) και το άλλο αφορά κάθε ενότητα(Module) ξεχωριστά. Οπότε, στο ήδη υπάρχων build.gradle(module app) της εφαρμογής μας εισάγουμε το παρακάτω κώδικα, εάν δεν υπάρχει:

```
repositories {
    mavenCentral()
}
```

Και στο πεδίο dependencies προσθέτουμε τη παρακάτω γραμμή:

```
compile 'com.facebook.android:facebook-android-sdk:4.0.0'
```

6.4.3.4 Εισαγωγή στοιχείων στο Manifest

Η εφαρμογή που έχει δημιουργηθεί στην ιστοσελίδα του Facebook developers περιέχει ένα μοναδικό ID. Θα πρέπει να το εισάγουμε στο manifest αρχείο της εφαρμογής χρησιμοποιώντας meta-data μέσα στο application στοιχείο:

```
<application android:label="@string/app_name" ...>;
  ...
  <meta-data android:name="com.facebook.sdk.ApplicationId"
     android:value="facebook_app_id_here"/>
  ...
</application>
```

Επίσης θα πρέπει να εισάγουμε και ένα Internet permission (η εφαρμογή θα ζητά το δικαίωμα να συνδέεται στο διαδίκτυο):

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Καθώς και ένα FacebookActivity:

```
<activity android:name="com.facebook.FacebookActivity"
        android:configChanges=
            "keyboard|keyboardHidden|screenLayout|screenSize|orientation"
        android:label="@string/app_name" />
```

6.4.3.5 Εισαγωγή ενός Facebook-login κουμπιού

Το facebook SDK περιλαμβάνει ένα έτοιμο LoginButton το οποίο μπορεί να χρησιμοποιηθεί σε ένα layout XML αρχείο ώστε να προστεθεί στη διεπαφή χρήστη(UI).

Εισάγουμε στο layout αρχείο που θα χρησιμοποιεί η εφαρμογή μας για να απεικονίσει το Facebook κουμπι, το παρακάτω:

```
<com.facebook.login.widget.LoginButton
   android:id="@+id/login_button"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="center_horizontal"
   android:layout_marginTop="30dp"
   android:layout_marginBottom="30dp" />
```

Στην συνέχεια στο Activity στο οποίο θα υπάρχει το παραπάνω Facebook κουμπί θα πρέπει να υπάρχει ένα *LoginButton* πεδίο, το οποίο θα το διαχειρίζεται σε επίπεδο κώδικα και ένα *CallbackManager* πεδίο που θα διαχειρίζεται τις απαντήσεις του Facebook όποτε ο χρήστης ζητά να συνδεθεί με αυτό. Οπότε, στη κλάση του Activity θα έχουμε τα εξής πεδία:

```
private LoginButton loginButton;
private CallbackManager callbackManager;
```

Στην onCreate μέθοδο του Activity θα πρεπει να αρχικοποιήσουμε το Facebook SDK καθώς και τα παραπάνω πεδία που δημιουργήσαμε:

```
FacebookSdk.sdkInitialize(getApplicationContext());
callbackManager = CallbackManager.Factory.create();
loginButton = (LoginButton) findViewById(R.id.login_button);
```

Θα πρέπει να δώσουμε μια λειτουργία στο *loginButton*, να μας επιστρέφει μια απάντηση όταν επιλέγεται και ο χρήστης ζητά να συνδεθεί μέσω του Facebook λογαριασμού του. Αυτό μπορούμε να το πετυχουμε με την *registerCallback*, η οποία αν η σύνδεση είναι επιτυχής θα καλέσει την *onSuccess*, αν ο χρήστης ακυρώσει την σύνδεση θα καλέσει την *onCancel* και αν η σύνδεση αποτύχει θα καλέσει την *onError*. Ο παρακάτω κώδικας εισάγεται και αυτός στην *onCreate* μέθοδο.

```
loginButton.registerCallback(callbackManager,new FacebookCallback<LoginResult>() {
    @Override
    public void onSuccess(LoginResult loginResult) {
        // do something on Success login
    }

    @Override
    public void onCancel() {
        // do something on cancel request
    }

    @Override
    public void onError(FacebookException e) {
        // do something on error result
    }
});
```

Τέλος θα πρέπει να χρησιμοποιήσουμε την *onActivityResult* για να δεχόμαστε την απάντηση του Facebook καθώς αυτό χρησιμοποιεί την Facebook Activity που δηλώσαμε στο Manifest. Γενικώς ένα Activity δέχεται και χειρίζεται αποτελέσματα που προέρχονται από άλλα Activities στην *onActivityResult* μέθοδο. Οπότε χρησιμοποιώντας αυτή τη μέθοδο έχουμε το παρακάτω:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    callbackManager.onActivityResult(requestCode, resultCode, data);
}
```

6.4.4 Google Sign-In

Οι προγραμματιστές έχουν τη δυνατότητα να προσφέρουν στους χρήστες των εφαρμογών τους τη σύνδεση σε αυτές μέσω του Google λογαριασμού τους χρησιμοποιώντας το Google Sign-In. Για να είναι δυνατή η χρήση του στις Android εφαρμογές απαιτήτει κάποιες προυποθέσεις:

- Η εφαρμογή να στοχεύει σε εκδόσεις του Android λειτουργικού από 2.3 (Gingerbread) και πάνω.
- Να υπάρχουν στην συσκευή εγκατεστημένα τα Google Play Services.

6.4.4.1 Αρχείο ρυθμίσεων

Αρχικά, χρειαζόμαστε ένα αρχείο ρυθμίσεων(configure file) το οποίο όπως και στη περίπτωση του Facebook login χρειάζεται για να ταυτοποιεί η Google την εφαρμογή με την οποία πρόκειται να αλληλεπιδρά. Για να παραχθεί αυτό το αρχείο ρυθμίσεων, απαιτείται να δημιουργηθεί μια νέα εφαρμογή στο λογαριασμό του χρήστη στο google developer console η οποία θα περιλαμβάνει:
το όνομα του κύριου πακέτου της Android εφαρμογής καθώς και ένα key hash. Αφού συμπληρωθούν τα απαραίτητα αυτά στοιχεία θα κατέβει αυτόματα ένα json αρχείο με το όνομα *google-service.json*. Αυτο θα πρέπει να τοποθετηθεί στον ριζικό φάκελο(/app) του project της Android εφαρμογής.
Το αρχείο ρυθμίσεων μπορεί να δημιουργηθεί γρήγορα και εύκολα στη παρακάτω ιστοσελίδα <https://developers.google.com/identity/sign-in/android/start> στο βήμα 2 *Get a configuration file*.

6.4.4.2 Εισαγωγή Google Sign-In στην Android εφαρμογή

Μέσω του Gradle(project build.gradle αρχείο) θα πρέπει να προσθέσουμε ένα πρόσθετο (plugin) για να μπορεί να αναγνωρίζεται το google-service.json αρχείο. Οπότε στο dependencies πεδίο προσθέτουμε την παρακάτω γραμμή:
classpath 'com.google.gms:google-services:3.0.0'

Το ίδιο κάνουμε και στο build.gradle(app) αρχείο, στο τμήμα dependencies όπου προσθέτουμε τη παρακάτω γραμμή για να χρησιμοποιήσουμε την αυθεντικοποίηση μέσω των Google Play Services:
compile 'com.google.android.gms:play-services-auth:9.8.0'

6.4.4.3 Εισαγωγή ενός Google Sign-In κουμπιού

Είμαστε έτοιμοι να εισάγουμε τον απαραίτητο κώδικα στην Android εφαμοργή μας. Διατίθεται ένα ετοιμο Google Sign-In κουμπι το οποίο μπορούμε να το εισάγουμε στο layout xml αρχείο που θέλουμε αυτό να εμφανίζεται με το παρακάτω τρόπο:

```
<com.google.android.gms.common.SignInButton
 android:id="@+id/sign_in_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

Στο Activity, χρειαζόμαστε ένα αντικείμενο *GoogleSignInOptions* για να δηλώσουμε ποιά πεδία θέλουμε να πάρουμε από το χρήστη(email, id, basic profile), ένα *GoogleApiClient* το οποίο θα μας δώσει πρόσβαση στο Google για τις πληροφορίες που ζητάμε και ένα *SignInButton* που είναι το Sign-In κουμπί που δηλώσαμε παραπάνω. Οπότε στη κλάση του Activity έχουμε τα παρακάτω πεδία:

```
private SignInButton signInButton;
private GoogleApiClient googleApiClient;
```

Και στην *onCreate* μέθοδο έχουμε τα παρακάτω:

```
signInButton = (SignInButton) findViewById (R.id.sign_in_button);
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
   .requestEmail()
   .build();

googleApiClient = new GoogleApiClient.Builder(this)
   .enableAutoManage(this /* activity */, this /* OnConnectionFailedListener */)
   .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
   .build();
```

Το Google Sign-In λειτουργεί με τον ίδιο τρόπο οπως και το facebook login παραπάνω, στέλνουμε το ερώτημα μας πατώντας το κουμπί και η απάντηση έρχεται από ένα άλλο Activity.
Για να στείλουμε αρχικά το ερώτημα θα πρέπει όταν επιλέγεται το signIButton να συμβαίνει το παρακάτω:

```
Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(googleApiClient);
startActivityForResult(signInIntent, 1000);
```

Και για να λάβουμε την απάντηση θα πρέπει στην *onActivityResult* μέθοδο να έχουμε το παρακάτω:

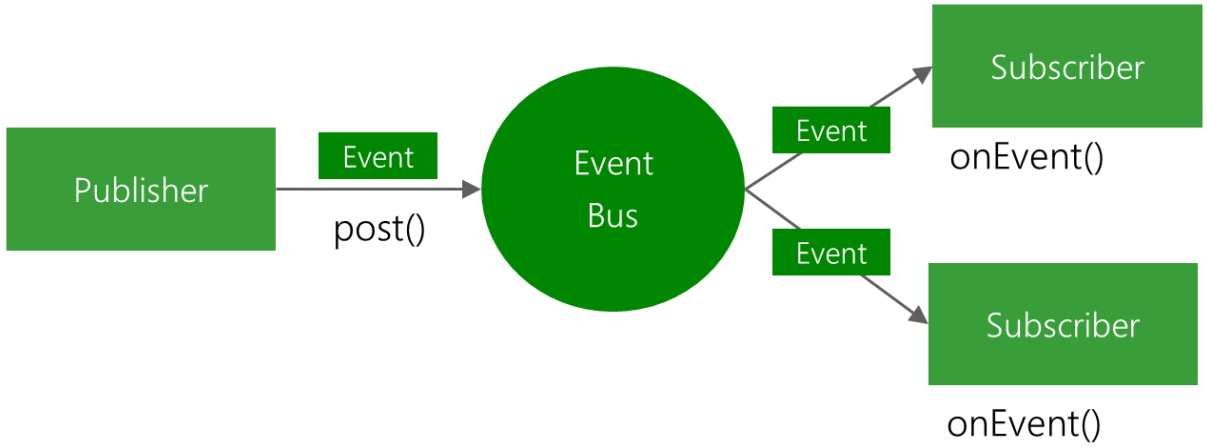
```
@Override public void onActivityResult(int requestCode, int resultCode, Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   // Result returned from launching the Intent from
   //   GoogleSignInApi.getSignInIntent(...);
   if (requestCode == 1000) {
      GoogleSignInResult result =
Auth.GoogleSignInApi.getSignInResultFromIntent(data);
      if (result.isSuccess()) {
         GoogleSignInAccount acct = result.getSignInAccount();
         // Get account information
         mFullName = acct.getDisplayName();
         mEmail = acct.getEmail();
      }
   }
}
```

6.4.5 EventBus

Το EventBus είναι μια ανοικτού κώδικα βιβλιοθήκη λογισμικού η οποία χρησιμοποιεί το publish/subscriber σχεδιαστικό πρότυπο(design pattern) [20] για να πετύχει λίγοτερη σύζευξη μεταξύ των κλάσεων.

Το EventBus επιτρέπει μια κεντρική επικοινωνία μεταξύ των κλάσεων με λίγες γραμμές κώδικα, απλοποιεί τον κώδικα, αφαιρεί εξαρτήσεις και επιταχύνει την ανάπτυξη της εφαρμογής.

Στην παρακάτω εικόνα 24 παρουσιάζεται σε ένα γράφημα για το πως λειτουργεί.



Εικόνα 24: Λειτουργία EventBus

6.4.5.1 Πλεονεκτήματα EventBus

- Απλοποιεί την επικοινωνία μεταξύ των συστατικών της εφαρμογής.
- Διαχωρίζει τον αποστολέα του event από τον παραλήπτη.
- Εχει καλή επίδοση στην χρήση με Activities, Fragments και threads του παρασκηνίου(background).
- Αποφεύγει περίπλοκες και επιρρεπείς σε λάθη εξαρτήσεις και ζητήματα με το κύκλο ζωής των Activities/Fragments.
- Μικρή σε μέγεθος βιβλιοθήκη (<50k jar).
- Διαθέτει προχωρημένα χαρακτηριστικά όπως διανομή του event σε κάποιο thread, προτεραιότητες σε παραλήπτες κ.α.

6.4.5.2 Χρήση EventBus

Για να χρησιμοποιήσουμε τη βιβλιοθήκη του EventBus σε ένα project, αρκεί στο build.gradle(Module: app) αρχείο να προσθέσουμε την παρακάτω γραμμή στο πεδίο dependencies:

```
compile 'org.greenrobot:eventbus:3.0.0'
```

6.4.5.3 Ορισμός Event

Ένα Event πρόκειται για ένα POJO(plain old java object) χωρίς κάποια συγκεκριμένα απαίτηση. Για παράδειγμα, ένα Event μπορεί να είναι η ακόλουθη κλάση:

```
public class MessageEvent {
    public final String message;
    public MessageEvent(String message) {
        this.message = message;
    }
}
```

Οι παραλήπτες(Subscribers) υλοποιούν τις μεθόδους που θα λάβουν/χειριστούν ένα Event όταν αυτό αποσταλεί. Αυτές οι μέθοδοι θα πρέπει να οριστούν με ένα @Subscribe annotation. Ακολουθεί ένα παράδειγμα με δύο τέτοιες μεθόδους:

```
// Αυτή η μέθοδο θα κληθεί όταν ένα MessageEvent(το ορίσαμε παραπάνω) σταλθεί (στο 'Main' thread)
@Subscribe(threadMode = ThreadMode.MAIN)
public void onMessageEvent(MessageEvent event) {
    Toast.makeText(getActivity(), event.message, Toast.LENGTH_SHORT).show();
}


// Αυτή η μέθοδος θα κληθεί όταν ένα SomeOtherEvent σταλεί.
```

```
@Subscribe
public void handleSomethingElse(SomeOtherEvent event) {
    doSomethingWith(event);
}
```

6.4.5.4 Προετοιμασία παραλήπτη (subscriber)

Οι παραλήπτες για να λαμβάνουν Events θα πρέπει να 'εγγραφούν'(register) και όταν πλέον δε χρειάζονται να καταργήσουν την εγγραφή τους. Οι εγγραφές και οι καταργήσεις εγγραφών θα πρέπει να γίνονται στις κατάλληλες μεθόδους των Activities/Fragments σε σχέση με τον κύκλο ζωής τους. Για τις περισσότερες περιπτώσεις οι *onStart/onStop* μέθοδοι είναι οι κατάλληλες, παράδειγμα:

```
@Override
public void onStart() {
    super.onStart();
    // εγγραφή
    EventBus.getDefault().register(this);
}

@Override
public void onStop() {
    //κατάργηση εγγραφης
    EventBus.getDefault().unregister(this);
    super.onStop();
}
```

6.4.5.5 Αποστολή Event

Μπορούμε να στείλουμε ένα Event από οποιοδήποτε μέρος του κώδικά μας. Όλοι οι παραλήπτες που έχουν κάνει εγγραφή(register) και ο τύπος Event που δέχονται ταιριάζει με αυτό που στάλθηκε θα το λάβουν.
Παράδειγμα αποστολής ενός Event:
EventBus.getDefault().post(new MessageEvent("Hello everyone!"));

6.4.6 Retrofit

Μια βιβλιοθήκη λογισμικού η οποία μετατρέπει ένα HTTP API σε μια Java διεπαφή(interface) επιτρέποντας στο προγραμματιστή να διαχειρίζεται πιο εύκολα HTTP ερωτήματα.

6.4.6.1 Δήλωση API

Annotations στις μεθόδους της διεπαφής και στις παραμέτρους των μεθόδων υποδεικνύουν πως θα χειριστεί ένα ερώτημα. Κάθε μέθοδο πρέπει να περιέχει ένα HTTP annotation που θα παρέχει τον τύπο του HTTP ερωτήματος και το αντίστοιχο σχετικό(relative) URL στο οποίο θα συμβεί το ερώτημα. Υποστηρίζονται πέντε τέτοιου είδους annotations: *GET, POST, PUT, DELETE, HEAD*. Το σχετικό URL εισάγεται μέσα στο annotation. Για παράδειγμα:
@GET("users/list")

Μπορούν επίσης να οριστούν και παράμετροι ερωτημάτων στο URL
@GET("users/list?sort=desc")

Κάθε μέθοδο της διεπαφής θα πρέπει να είναι τύπου *Call<>* η οποία μέσα στο Generic θα περιλαμβάνει το τύπο που επιστρέφει το συγκεκριμένο ερώτημα.
Για παράδειγμα, εάν θέλουμε να στείλουμε ένα ερώτημα στο REST API του Github ώστε να πάρουμε τα αποθετήρια ενός χρήστη θα έχουμε:

Τη διεπαφή για το HTTP ερώτημα:

```
public interface GitHubService {
  @GET("users/{user}/repos")
  Call<List<Repo>> listRepos(@Path("user") String user);
}
```

Ένα Retrofit αντικείμενο που θα δημιουργεί την υλοποίηση της GitHubService διεπαφής και θα περιέχει το κύριο URL του HTTP ερωτήματος:

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .build();
GitHubService service = retrofit.create(GitHubService.class);
```

6.4.6.2 Χειρισμός URL

Ένα ερώτημα σε ένα URL μπορεί να ενημερωθεί δυναμικά χρησιμοποιώντας μπλοκ αντικατάστασης(replacement blocks) και παραμέτρους στη μέθοδο. Ένα μπλόκ αντικατάστασης είναι ένα αλφαρηθμιτικό που περικλύεται απο *{* και *}* . Η αντίστοιχη παράμετρος θα πρέπει να έχει ένα @Path annotation και να χρησιμοποιεί το ίδιο αλφαριθμιτικό όνομα. Για παράδειγμα, η διεπαφή που ορίσαμε παραπάνω χρησιμοποιεί ένα μπλόκ αντικατάστασης:

```
@GET("users/{user}/repos")
Call<List<Repo>> listRepos(@Path("user") String user);
```

Παράμετροι στο URL μπορούν να εισαχθούν επίσης δυναμικά χρησιμοποιώντας το @Query annotation στη παράμετρο της μεθόδου της διεπαφής:

```
@GET("group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId, @Query("sort") String sort);
```

6.4.6.3 Σώμα ερωτήματος (Body Request)

Ένα αντικείμενο μπορεί να καθοριστεί για χρήση στο σώμα(Body) ενός HTTP ερωτήματος με το @Body annotation. Για παράδειγμα, έστω ότι έχουμε ένα Java αντικείμενο User και θέλουμε να το προσθέσουμε στο σώμα ενός POST ερωτήματος:

```
@POST("users/new")
Call<User> createUser(@Body User user);
```

Όπως έχουμε δει όμως σε προηγούμενα κεφάλαιο θα πρέπει με κάποιο τρόπο αυτό το Java αντικείμενο να μετατραπεί στη μορφή που δέχεται ερωτήματα το Rest API, συνήθως JSON, οπότε χρειαζόμαστε έναν JSON Parser. Το Retrofit υποστηρίζει αρκετούς JSON Parsers(Gson, Jackson, Moshi κ.α). Οπότε, για να χρησιμοποιήσουμε στο ερώτημά μας το Gson parser, στο αντικείμενο Retrofit εισάγουμε την μέθοδο *addConverterFactory(GsonConverterFactory.create()):*

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com")
    .addConverterFactory(GsonConverterFactory.create())
    .build();
```

6.4.6.4 Χειρισμός κεφαλίδας (header)

Μπορούμε να προσθέσουμε μια στατική κεφαλίδα σε ένα ερώτημα, χρησιμοποιώντας το @Headers annotation. Για παράδειγμα, για να δηλώσουμε τον τύπο που δέχεται το ερώτημα και το όνομα του πελάτη(User-agent):

```
@Headers({
    "Accept: application/json",
    "User-Agent: Retrofit-Sample-App"
})
@GET("users/{username}")
Call<User> getUser(@Path("username") String username);
```

Επίσης μπορούμε να προσθέσουμε δυναμικά μια κεφαλιδα σε ένα ερώτημα χρησιμοποιώντας το @Header annotation στην παράμετρο της μεθόδου της διεπαφής. Για παράδειγμα, εάν θέλουμε σε ένα ερώτημα να προσθέτουμε ένα πεδίο Authorization(όπως στην τρέχουσα πτυχιακή):

```
@GET("/user")
Call<User> getUser(@Header("Authorization") String authorization)
```

6.4.6.5 Εκτέλεση ερωτημάτων

Τα ερώτημα μπορούν να εκτελεστούν σύγχρονα και ασύγχρονα. Όπως είδαμε παραπάνω ένα Retrofit αντικείμενο δημιουργεί την υλοποίηση της διεπαφής του HTTP ερωτήματος. Επιστρέφωντας στο αρχικό παράδειγμα όπου πέρναμε τα αποθετήρια ενός χρήστη απο το Github είχαμε την υλοποίηση της διεπαφής στο παρακάτω πεδίο:

```
GitHubService service = retrofit.create(GitHubService.class);
```

Εάν θέλουμε να εκτελέσουμε ασύγχρονα το HTTP ερώτημα θα καλέσουμε την *enqueue()* μέθοδο του Retrofit η οποία περιλαμβάνει ένα Callback με μια *onResponse()* μέθοδο και μια *onFailure()* μέθοδο όπου διαχειριζόμαστε την απάντηση του ερωτήματος:

```
service.listRepos("gitHubUserName").enqueue(new Callback<List<Repo>>() {

    @Override
    public void onResponse(Call<List<Repo>> call, Response<List<Repo>> response) {
        // do something on success response
    }

    @Override
    public void onFailure(Call<List<Repo>> call, Throwable t) {
        // do something on failure response
    }

});
```

Σε αντίθετη περίπτωση, εάν θέλουμε να εκτελέσουμε το ερώτημα σύγχρονα θα χρησιμοποιήσουμε την *execute()* μέθοδο του Retrofit η οποία επιστρέφει ένα Response<> αντικείμενο το οποίο περιλαμβάνει την απάντηση του ερωτήματος. Η *execute()* μέθοδος θα 'πετάξει' ένα IOException εάν υπάρξει κάποιο προβλημα επικοινωνίας με τον εξυπηρετητή(server) ή ένα RuntimeException σε περίπτωση που δε μπορέσει να δημιουργήσει το ερώτημα:

```
Response<List<Repo>> response = service.listRepos("gitHubUsername").execute();
if (response.isSuccessful()) {
    // do something on success response
}
```

6.4.7 RxJava

Το ReactiveX είναι ένα API σχετικά με την ασύγχρονη σύνθεση και τον χειρισμό των παρατεταμένων ροών από δεδομένα ή γεγονότα χρησιμοποιώντας ένα συνδυασμό του Observer προτύπου [21], του Iterator προτύπου [22] και χαρακτηριστικά του συναρτησιακού προγραμματισμού(Functional Programming) [23].

Η διαχείριση δεδομένων σε πραγματικό χρόνο είναι μια συχνή περίπτωση, οπότε μια αποτελεσματική και επεκτάσιμη προσέγγιση είναι σημαντική.

Το ReactiveX προσφέρει ένα σύνθετο και ευέλικτο API για τη δημιουργία και τον χειρισμό παρατεταμένων δεδομένων ενώ απλοποιεί τον ασύγχρονο προγραμματισμό, όπως τη δημιουργία thread και προβληματα συγχρονισμού.

Η RxJava είναι μια ανοιχτού κώδικα υλοποίηση του ReactiveX σε Java. Αποτελείται από δύο εκδόσεις, την RxJava 1 και RxJava 2. Έχουν αρκετές διαφορές μεταξύ τους μιας και η δεύτερη δεν στηρίχτηκε στην πρώτη άλλα γράφτηκε από την αρχή. Στη συγκεκριμένη πτυχιακή χρησιμοποιήθηκε η RxJava 1.

Οι δύο κύριες κλάσεις της είναι η *Observable* και η *Subscriber*. Η Observable είναι η κλάση που εκπέμπει παρατεταμένα δεδομένα ή γεγονότα και η Subscriber είναι μια κλαση που ενεργεί πάνω στα εκπεμπόμενα αντικείμενα. Η τυπική ροή από ένα Observable είναι να εκπέμπει ένα η περισσότερα αντικείμενα και στη συνέχεια να ολοκληρώνεται επιτυχώς ή με κάποιο σφάλμα αυτή η εκπομπή. Ένα Observable μπορεί να έχει πολλούς Subscribers, και για κάθε αντικείμενο που εκπέμπεται από το Observable, το αντικείμενο θα σταλεί στην *Subscriber.onNext()* μέθοδο για να το διαχειριστεί. Μόλις ένα Observable τελειώσει επιτυχώς να εκπέμπει αντικείμενα θα καλέσει την *Subscriber.onCompleted()* μέθοδο, σε αντίθετη περίπτωση θα καλέσει την *Subscriber.onError()* μέθοδο.

Τώρα που γνωρίζουμε για τις Observable και Subscriber κλάσεις μπορούμε να δούμε πως τις δημιουργούμε και πως τις χειριζόμαστε.

6.4.7.1 Δημιουργία/διαχείριση Observable/Subscriber

Χρησιμοποιώντας την μέθοδο *Observable.create()* μπορούμε να δημιουργήσουμε ένα Observable.

```
Observable integerObservable = Observable.create(new Observable.OnSubscribe() {
  @Override
  public void call(Subscriber subscriber) {
     subscriber.onNext(1);
     subscriber.onNext(2);
     subscriber.onNext(3);
     subscriber.onCompleted();
  }
});
```

Το παραπάνω Observable θα εκπέμψει τους ακέραιους αριθμούς 1,2,3 και συνεχεία θα ολοκληρωθεί. Αυτό που μένει είναι να δημιουργήσουμε και έναν Subscriber ώστε να μπορούμε να ενεργήσουμε στα δεδομένα που μόλις στάλθηκαν.

```
Subscriber integerSubscriber = new Subscriber() {
  @Override
  public void onCompleted() {
     System.out.println("Complete!");
  }
  @Override
  public void onError(Throwable e) {

  }
```

```
  @Override
  public void onNext(Integer value) {
     System.out.println("onNext: " + value);
  }
};
```

Ο Subscriber θα εμφανίσει όσα στοιχεία στάλθηκαν και θα μας ενημερώσει κατά την ολοκλήρωση.
Εφόσον διαθέτουμε ένα Observable και ένα Subscriber μπορούμε να τα συνδέσουμε μεταξύ τους με την *Observable.subscribe()* μέθοδο:
integerObservable.subscribe(integerSubscriber);

Το οποίο θα έχει το παρακάτω αποτέλεσμα:
onNext: 1
onNext: 2
onNext: 3
Complete!

Όλος ο παραπάνω κώδικας μπορεί να απλοποιηθεί χρησιμοποιώντας την *Observable.just()* μέθοδο για να δημιουργήσουμε ένα Observable που θα εκπέμπει μόνο τις τιμές που θα οριστούν στην just() μέθοδο, και να αλλάξουμε το Subscriber σε μια εσωτερική ανώνυμη κλάση:

```
Observable.just(1, 2 ,3).subscribe(new Subscriber() {
  @Override
  public void onCompleted() {
     System.out.println("Complete!");
  }

  @Override
  public void onError(Throwable e) {}

  @Override
  public void onNext(Integer value) {
     System.out.println("onNext: " + value);
  }
});
```

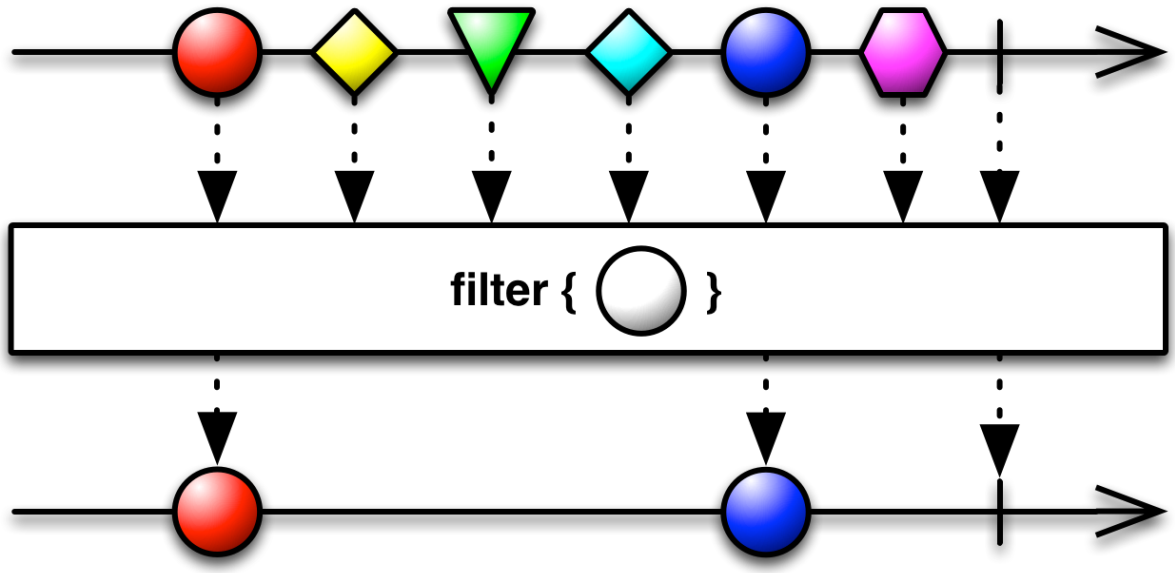
6.4.7.2 Τελεστές

Ένα Observable μπορεί να δεχτεί αλλαγές στο αποτέλεσμά του από έναν τελεστή(Operator) ή από πολλούς τελεστές οι οποίοι μπορούν να προστεθούν σε ένα Observable. Υπάρχουν πάρα πολλοί διαθέσιμοι τελεστές στην RxJava. Παρακάτω αναλύονται αυτοί που χρησιμοποιήθηκαν στη παρούσα πτυχιακή εργασία.
Το σύνολο τους μπορούν να βρεθούν στο παρακάτω σύνδεσμο [24].

6.4.7.2.1 Filter

Εκπέμπει μόνο εκείνα τα αντικείμενα από ένα Observable που ικανοποιούν κάποια συνθήκη.

Στην παρακάτω εικόνα Χ φένεται η λειτουργία του:



Εικόνα 25: Λειτουργία Filter τελεστή

Ακολουθεί ένα παράδειγμα το οποίο από τους πέντε ακέραιους αριθμούς που εκπέμπονται αρχικά από ένα Observable, μόνο αυτοί που είναι μικρότεροι του τέσσερα (< 4) τελικά θα σταλούν.

```
Observable.just(1, 2, 3, 4, 5)
        .filter(new Func1<Integer, Boolean>() {
            @Override
            public Boolean call(Integer item) {
              return( item < 4 );
            }
         }).subscribe(new Subscriber<Integer>() {
       @Override
       public void onNext(Integer item) {
          System.out.println("Next: " + item);
       }

       @Override
       public void onError(Throwable error) {
          System.err.println("Error: " + error.getMessage());
       }

       @Override
       public void onCompleted() {
          System.out.println("Sequence complete.");
       }
   });
```

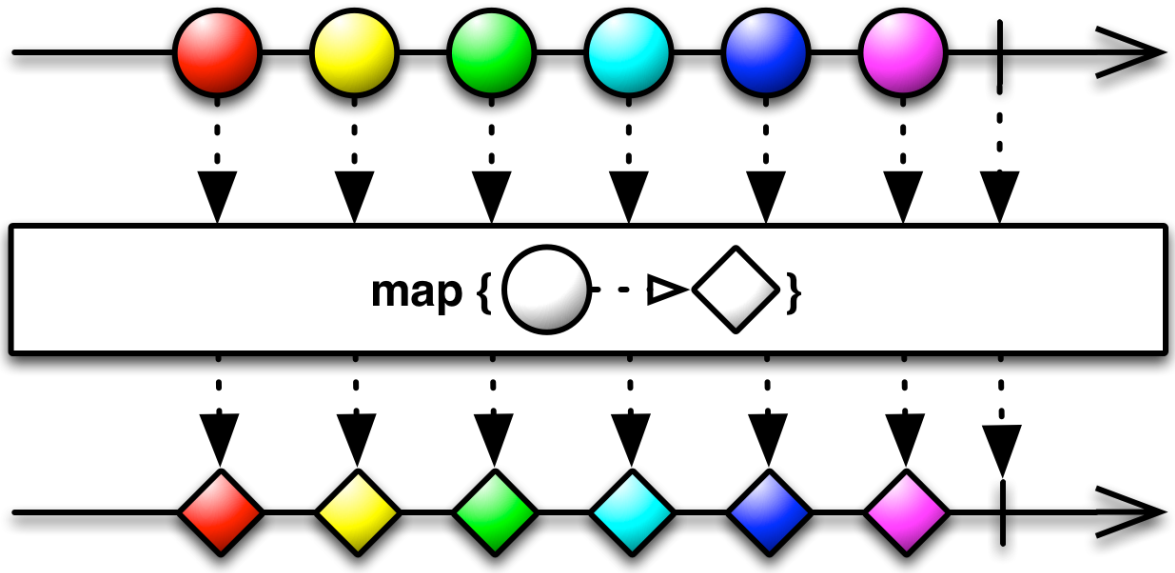
Αποτέλεσμα του παραπάνω κώδικα:

```
Next: 1
Next: 2
Next: 3
Sequence complete.
```

6.4.7.2.2 Map

Μετατρέπει τα εκπεμπόμενα στοιχεία από ένα Observable εφαρμώζοντας μια λειτουργία σε κάθε ένα στοιχείο.

Η λειτουργία της φένεται και στη παρακάτω εικόνα 26.



Εικόνα 26: Λειτουργία Map τελεστή

Για παράδειγμα, στο κώδικα που ακολουθεί οι αριθμοί που στέλνονται αρχικά, μετατρέπονται σε ένα αλφαριθμητικό(String) με τη βοήθεια του τελεστή Map.

```
Observable.just(1, 2, 3, 4, 5)
        .map(new Func1<Integer, String>() {
            @Override
            public String call(Integer number) {
                return number + " is a number";
            }
        }).subscribe(new Subscriber<String>() {
        @Override
        public void onNext(String item) {
            System.out.println("Next: " + item);
        }

        @Override
        public void onError(Throwable error) {
            System.err.println("Error: " + error.getMessage());
        }

        @Override
        public void onCompleted() {
            System.out.println("Sequence complete.");
        }
    });
```

Αποτέλεσμα του παραπάνω κώδικα:

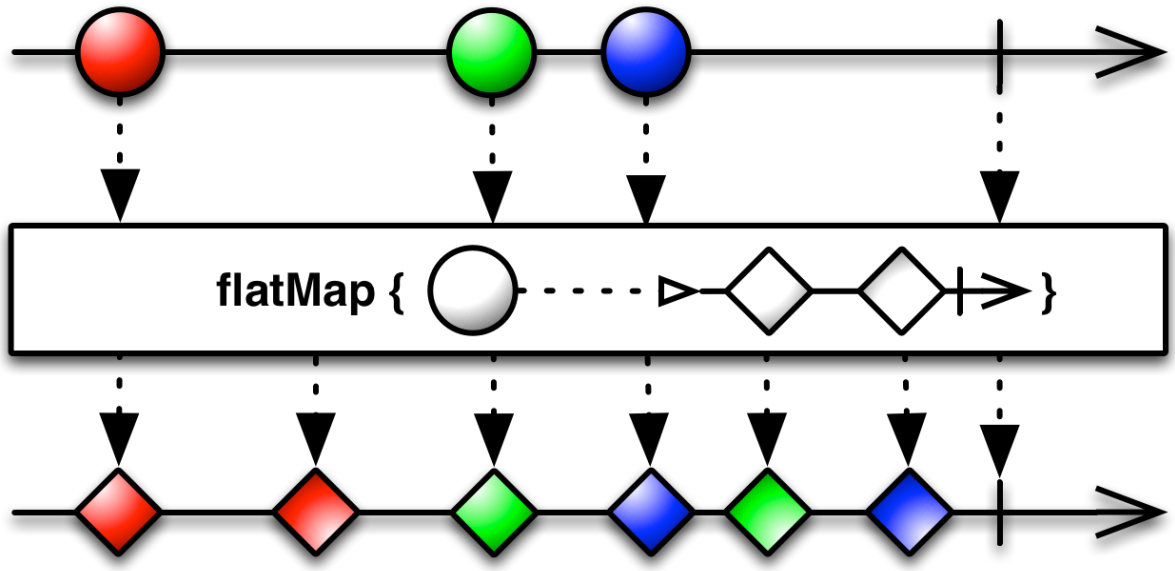
```
Next: 1 is a number
Next: 2 is a number
Next: 3 is a number
Next: 4 is a number
Next: 5 is a number
Sequence complete.
```

6.4.7.2.3 FlatMap

Πέρνει τα στοιχεία ενός Observable και τα μετατρέπει και τα αντικαθιστά με τα στοιχεία ενός άλλου Observable.

Δηλαδή, εφαρμόζει κάποιες λειτουργίες στα στοιχεία του αρχικού Observable ώστε να δημιουργήσει και να εκπέμψει στοιχεία άλλου Observable.

Η λειτουργία της φένεται και στη εικόνα 27.



Εικόνα 27: Λειτουργία flatMap

Για παράδειγμα, έστω ότι έχουμε την παρακάτω μέθοδο η οποία επιστρέφει μια λίστα από ιστοσελίδες βασισμένες σε μια λέξη κλειδί:

Observable<List<String>> query(String text);

Θέλουμε να κάνουμε το σύστημά μας να επιστρέφει τα αποτελέσματα από τη λέξη κλειδί που θα εισάγει ο χρήστης.

```
query("The rise")
    .flatMap(new Func1<List<String>, Observable<String>>() {
        @Override
        public Observable<String> call(List<String> urls) {
            return Observable.from(urls);
        }
    }).subscribe(new Subscriber<String>() {
        @Override
        public void onNext(String item) {
            System.out.println("Next: " + item);
        }

        @Override
        public void onError(Throwable error) {
            System.err.println("Error: " + error.getMessage());
        }

        @Override
        public void onCompleted() {
            System.out.println("Sequence complete.");
        }
    });
```

Παρατηρούμε ότι χρησιμοποιείται ο flatMap τελεστής ο οποίος εφαρμόζεται στην *query()* μέθοδο και για κάθε στοιχείο που αυτή επιστρέφει καλεί την *Observable.from()* ώστε να στείλει στον subscriber το νέο Observable για το κάθε προηγούμενο στοιχείο. Η *Observable.from()* μέθοδος πέρνει τα στοιχεία μια συλλογής και τα εκπέμπει ένα τη φορά. Οπότε ο Subscriber δεν δέχεται μια λίστα από

αλφαριθμητικά, αλλά μια σειρά από αλφαριθμητικά τα οποία στέλνονται απο την *Observable.from().*

Ο συγκεκριμένος τελεστής ίσως είναι από τους πιο δύσκολους να καταλάβει κάποιος καινούργιος στην RxJava. Αλλά μόλις εξοικειωθεί μαζί του θα αρχίσουν να ξεκαθαρίζουν όλα περισσότερο.

6.4.7.3 Schedulers

Έχουμε μια εφαρμογή η οποία εκτελεί HTTP ερωτήματα σε ένα REST API. Κάποια από αυτά ίσως διαρκέσουν αρκετή ώρα, οπότε θα πρέπει να εκτελούμε αυτά τα ερώτηματα σε άλλο Thread. Στην RxJava μπορούμε να πούμε στο Observable σε ποιό Thread να εκτελεστεί με την *subscribeOn()* μέθοδο και σε ποιό thread να εκτελεστεί ο Subscriber με την *observeOn()* μέθοδο.

Για λειτουργίες που θέλουμε να γίνουν στο παρασκήνιο, μπορούμε να χρησιμοποιήσουμε τον *Schedulers.io()* ενώ για λειτουργίες που θέλουμε να γίνουν στο Main thread μπορούμε να χρησιμοποιήσουμε τον *AndroidSchedulers.mainThread().*

Για παράδειγμα, έστω ότι έχουμε μια μέθοδο η οποία δημιουργεί ένα *Bitmap* απο ένα URL, τότε θα δημιουργήσουμε το Bitmap σε ένα background Thread, και θα το διαχειριστούμε στη συνέχεια στο Main thread.

```
myObservableService.retrieveImage(url)
   .subscribeOn(Schedulers.io())
   .ObservableOn(AndroidSchedulers.mainThread())
   .subscribe(new Subscriber<Bitmap>() {
      @Override
      public void onNext(Bitmap image) {
         //do something onNext
      }
```

```
    @Override
    public void onError(Throwable error) {
        //do something onError
    }

    @Override
    public void onCompleted() {
        //do something onCompleted
    }
});
```

6.4.7.4 Lambdas

Ο κώδικας που έχουμε χρησιμοποιήσει μέχρι στιγμής στην RxJava μπορεί να απλοποιηθεί χρησιμοποιώντας Lambdas [25].

Τα Lambdas είναι χαρακτηριστικό της Java 8, όμως στο Android δεν υπάρχει ακόμη ολοκληρωμένη υποστήριξη για τα χαρακτηριστικά της Java 8 , οπότε μπορούμε να χρησιμοποιήσουμε τη Retrolambda βιβλιοθήκη που μας προσφέρει τη λειτουργία των Lambdas στο Android. Ακολουθεί ένα παραδειγμα σε RxJava με χρήση Lambdas:

```
Observable.just("Hello, world!")
    .subscribe(s -> System.out.println(s));
```

Αποτέλεσμα:

Hello, world!

6.4.7.5 Χρήση στο Android

Για να χρησιμοποιηθεί η RxJava σε μια Android εφαρμογή αρκεί στο build.gradle(Module:app) αρχείο να προσθέσουμε στο πεδίο dependencies τις παρακάτω γραμμές:

compile 'io.reactivex:rxandroid:1.2.1'

compile 'io.reactivex:rxjava:1.1.6'

Η RxAndroid περιέχει κάποιες συγκεκριμένες λειτουργίες για το Android, όπως για παράδειγμα τον *AndroidSchedulers.mainThread()* που είδαμε παραπάνω, δεν είναι απαραίτητο για τη χρήση της RxJava στο Android αλλά διευκολύνει ακόμη περισσότερο το έργο του προγραμματιστή.

Η χρήση της RxJava στο Android αρχικά αφορούσε στην αντικατάσταση των AsyncTask και γενικά στο ότι είχε να κάνει με διαχείριση ασύγχρονων δεδομένων. Με το πέρασμα του χρόνο υιοθετήθηκε όλο και από περισσότερους προγραμματιστές, οι υποστηρικτές της αυξήθηκαν και σήμερα πρόκειται για ένα από τα πιο δημοφιλές εργαλεία στην ανάπτυξη Android εφαρμογών. Πλέον, μπορούμε να τη δούμε να εφαρμόζεται σε οποιοδήποτε σημείο στο κώδικα και να υλοποιεί οποιαδήποτε λειτουργία θέλουμε.

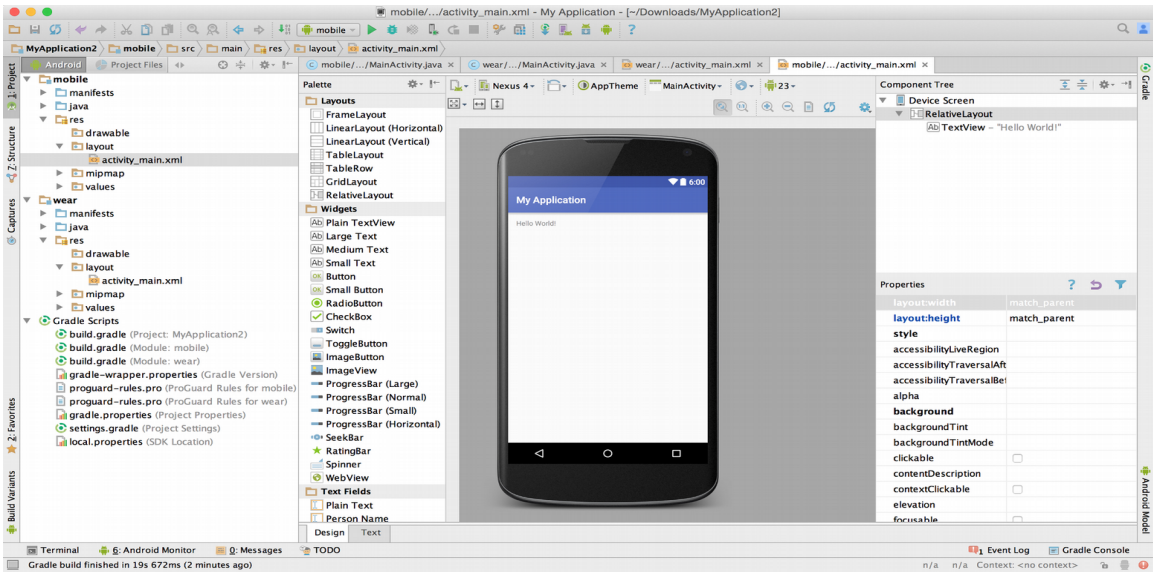
6.4.8 Android Studio

Το Android Studio είναι ένα ολοκληρωμένο προγραμματιστικό περιβάλλον (IDE) για ανάπτυξη εφαρμογών στην πλατφόρμα Android, βασισμένο στο Intellij IDEA.

Διατίθεται από τη Google και είναι το IDE το οποίο προτείνεται και επίσημα από την ίδια για την ανάπτυξη εφαρμογών Android.

Προσφέρει χαρακτηριστικά όπως:

- Υποστήριξη Gradle
- Πλούσιο επεξεργαστή κειμένου
- Πλούσιο επεξεργαστή διάταξης(layout) με επιλογή για προεπισκόπηση της διάταξης σε ποικίλες διαστάσεις οθονών.
- Ενσωματωμένη υποστήριξη για την Cloud πλατφόρμα της Google.
- Διευρυμένη υποστήριξη προτύπων για τις υπηρεσίες της Google και για διάφορους τύπους συσκευών.
- Εικονή Android συσκευή(Emulator) για εκτέλεση και αποσφαλμάτωση εφαρμογών.



Εικόνα 28: Στιγμιότυπο οθόνης Android Studio

6.5 Επίλογος

Συνοψίζοντας, σε αυτό το κεφάλαιο έγινε μια προσπάθεια να περιγραφούν αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν προκειμένου να ολοκληρωθεί επιτυχώς η Android εφαρμογή. Σύμφωνα λοιπόν με όλα τα παραπάνω είμαστε έτοιμοι να περιγράψουμε την αναλυτική λειτουργία της.

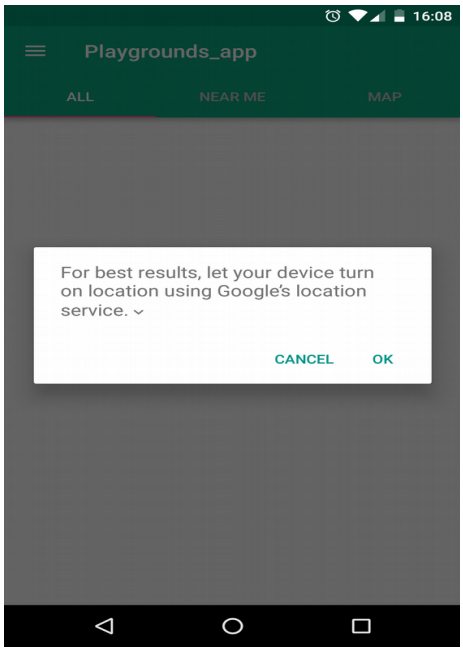
7 Αναλυτική λειτουργία και χρήση της εφαρμογής

7.1 Εισαγωγή

Στο έβδομο και τελευταίο κεφάλαιο αυτής της πτυχιακής εργασίας θα περιγραφεί η αναλυτική λειτουργία της εφαρμογής με τη χρήση πολλών στιγμιοτύπων οθόνης(screenshot) ώστε να γίνει όσο το δυνατόν πιο κατανοητή στον αναγνώστη.

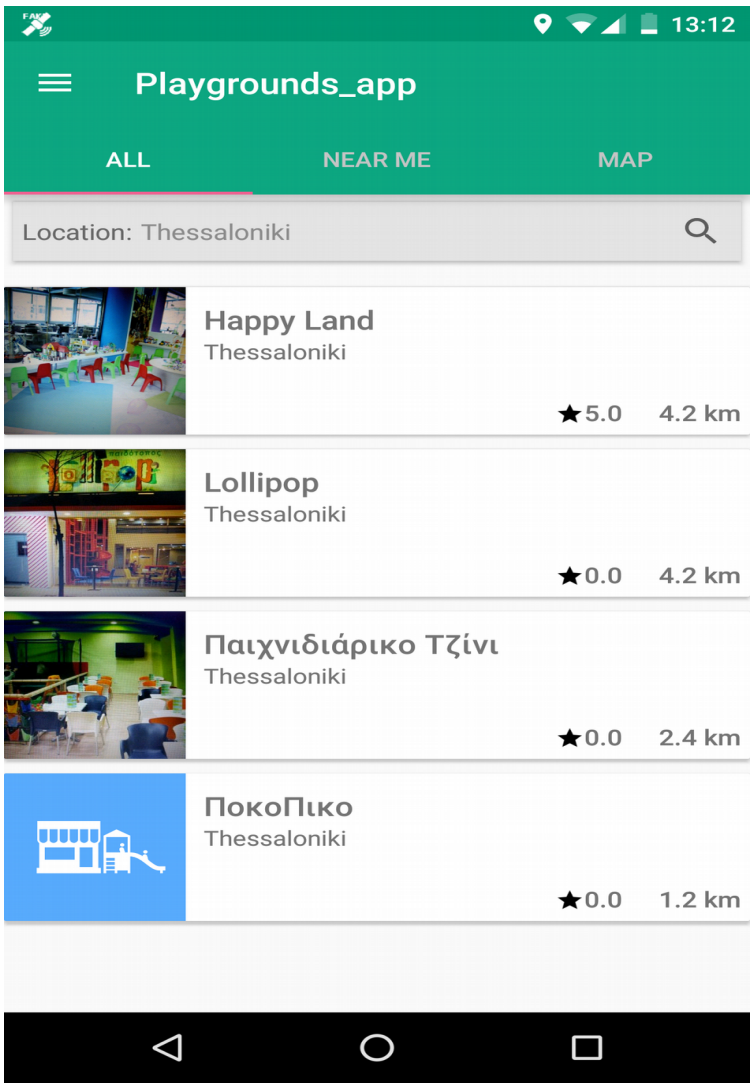
7.2 Αρχική Οθόνη

Η λειτουργία της εφαρμογής βασίζεται εξολοκλήρου στη τοποθεσία του χρήστη. Οπότε, μόλις ο χρήστης εκτελέσει την εφαρμογή ο πρώτος έλεγχος που γίνεται είναι εάν υπάρχει σύνδεση στο διαδίκτυο και εάν η τοποθεσία(GPS) της συσκευής είναι ενεργοποιημένη. Στη περίπτωση που η τοποθεσία δεν είναι ενεργοποιημένη εμφανίζεται το Dialog της εικόνας 29 και ζητά από το χρήστη να επιτρέψει στην εφαρμογή να την ενεργοποιήσει.



Εικόνα 29: Άδεια για ενεργοποίηση τοποθεσίας

Εφόσον ενεργοποιηθεί ή είναι ήδη ενεργοποιημένη η τοποθεσία, εμφανίζεται η οθόνη της εικόνας 30 η οποία αποτελεί και το κύριο μέρος της εφαρμογής.



Εικόνα 30: Κύρια οθόνη

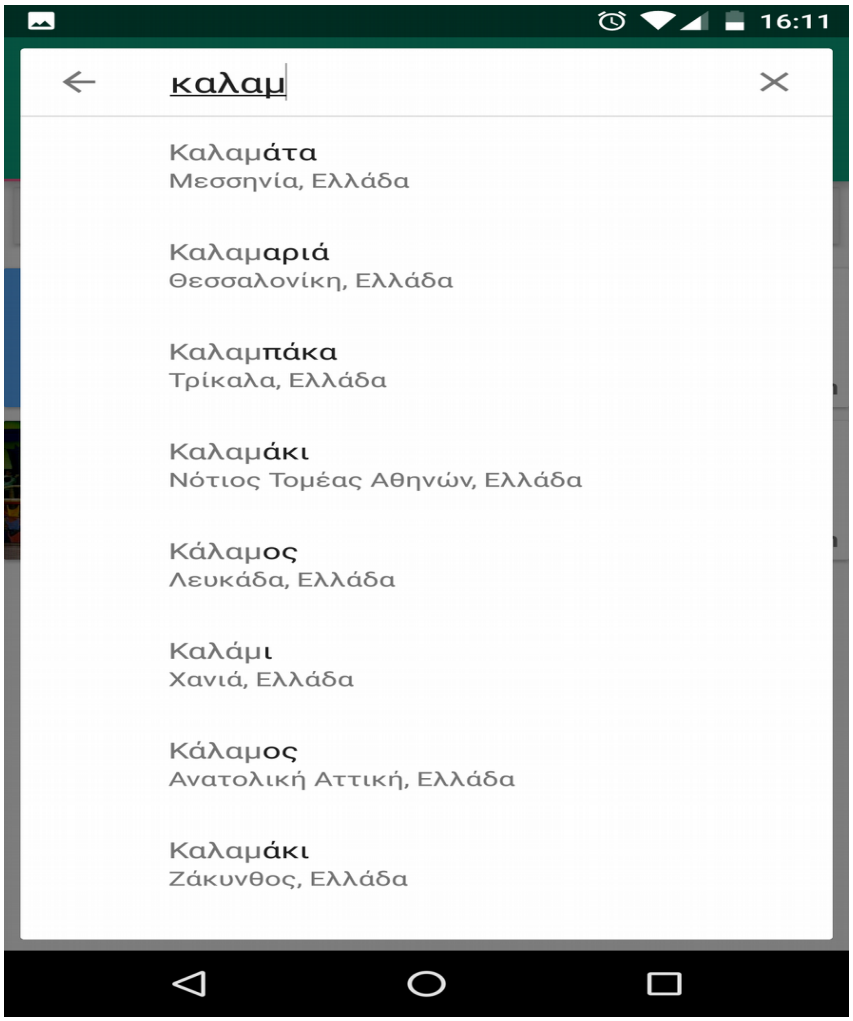
Συγκεκριμένα βλέπουμε τρείς καρτέλες ALL, NEAR ME, MAP με την ALL να είναι η προεπιλεγμένη.

7.3 ALL

Η καρτέλα ALL, που αποτελεί την προεπιλεγμένη καρτέλα της αρχική οθόνης περιέχει μια λίστα στην οποία εμφανίζονται όλοι οι παιδότοποι που υπάρχουν στην τοποθεσία που βρίσκεται ο χρήστης σε μορφή καρτών. Οι παιδότοποι ταξινομούνται βάσει καλύτερης βαθμολογίας, και σε κάθε κάρτα παιδότοπου αναφέρεται το όνομα του, μια μικρογραφία της εικόνας προφίλ του, η γενική βαθμολογία του καθώς και η χιλιομετρική απόσταση που απέχει από τη τοποθεσία του χρήστη.

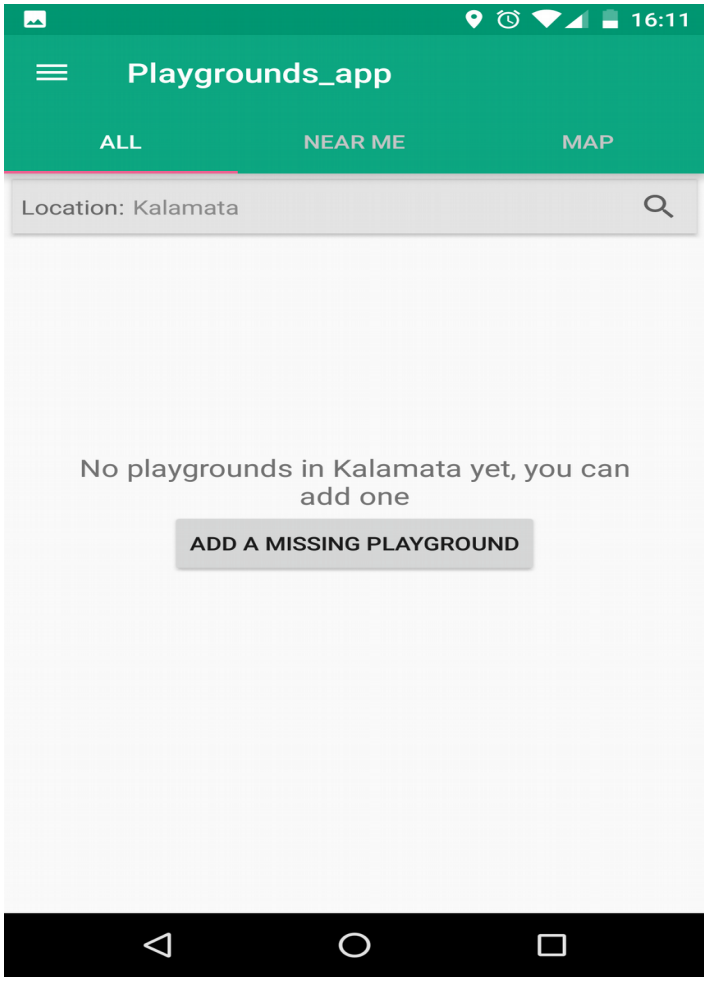
7.4 Αναζήτηση

Το πλαίσιο στη καρτέλα ALL το οποίο αναγράφει *'Location: Thessaloniki'* αναφέρεται στη τρέχων τοποθεσία στην οποία ανήκουν οι παραπάνω παιδότοποι. Η τιμή αυτού του πεδίου ενημερώνεται αρχικά από τη τοποθεσία του χρήστη(GPS). Πατώντας πάνω σε αυτό το πλαίσιο δίνεται η δυνατότητα στο χρήστη να αναζητήσει παιδότοπους σε άλλες περιοχές, όπως αυτό φένεται στην εικόνα 31. Η αναζήτηση παρέχει λειτουργία αυτόματης συμπλήρωσης περιοχών από τη Google για καλύτερα αποτελέσματα.



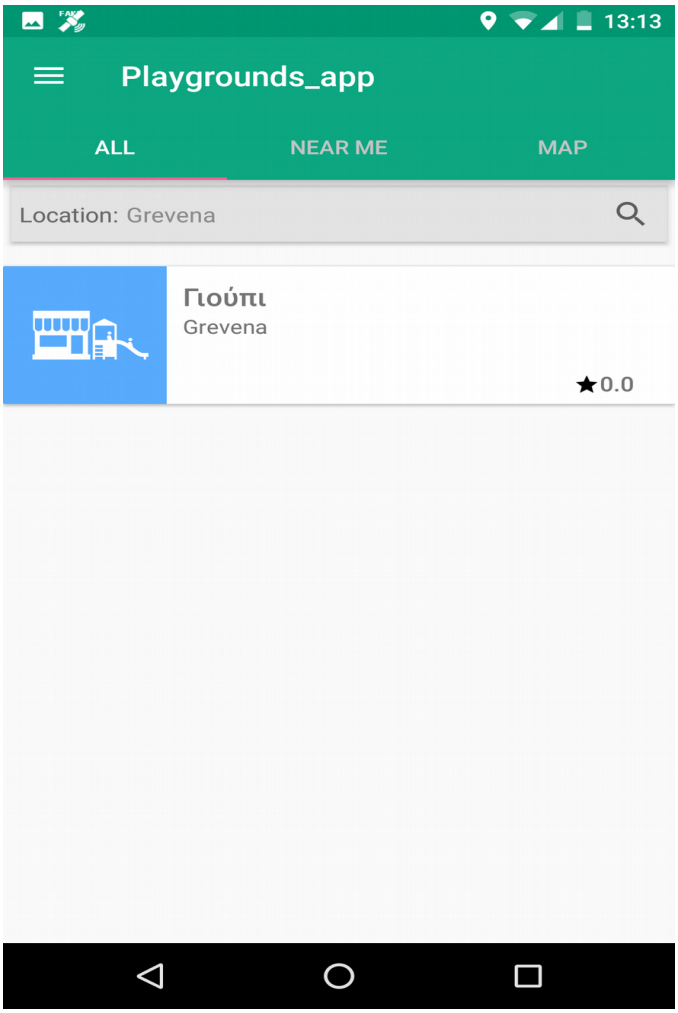
Εικόνα 31: Αναζήτηση περιοχών

Εάν σε κάποια περιοχή δεν έχει καταχωρηθεί κάποιος παιδότοπος εμφανίζεται μήνυμα το οποίο αναφέρει ότι δεν βρέθηκαν παιδότοποι και προτρέπει τον χρήστη εάν επιθυμεί να προσθέσει κάποιον παιδότοπο που λείπει. Εικόνα 32.



Εικόνα 32: Δεν βρέθηκαν παιδότοποι

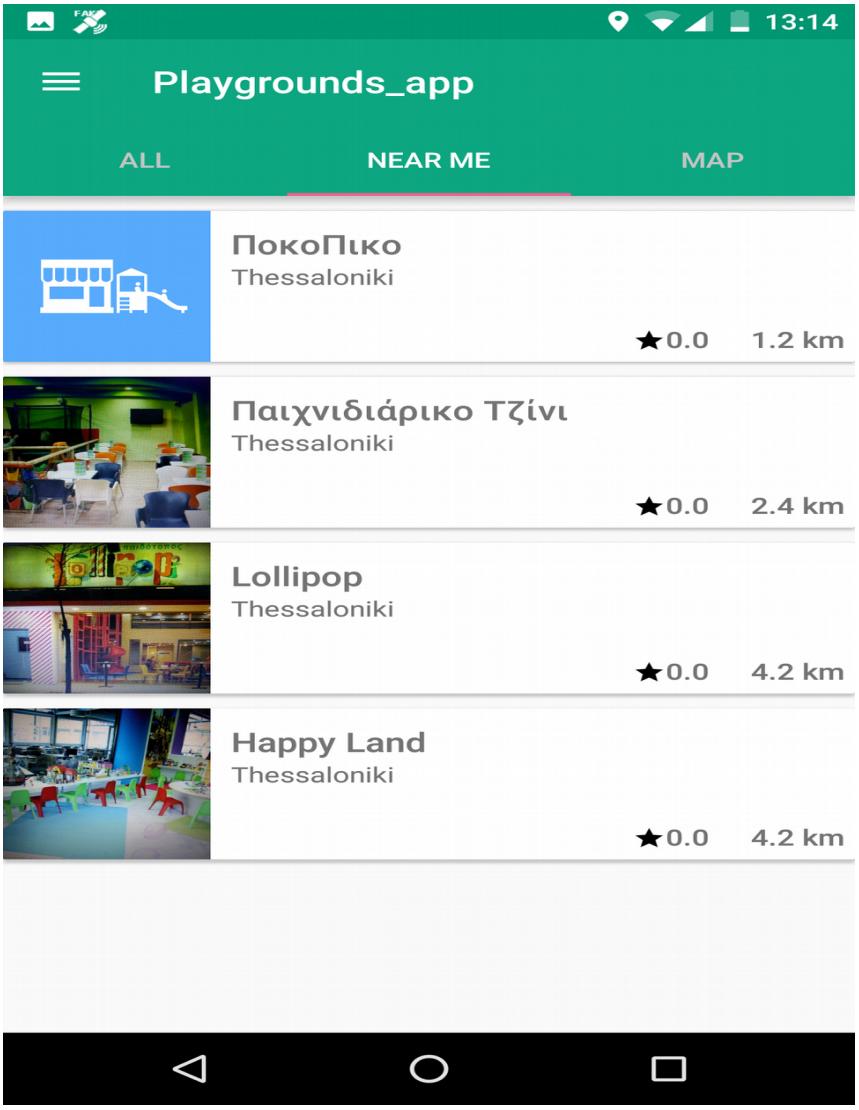
Όταν ο χρήστης αναζητά παιδότοπους σε άλλες περιοχές, πέρα από αυτήν στην οποία βρίσκεται, στις κάρτες των παιδότοπων δεν εμφανίζεται η χιλιομετρική απόσταση του παιδότοπου απο τη τοποθέσια του χρήστη καθώς δε θα προσφέρει κάποια χρήσιμη πληροφορία.



Εικόνα 33: Αποτέλεσμα αναζήτησης σε άλλη περιοχή

7.5 NEAR ME

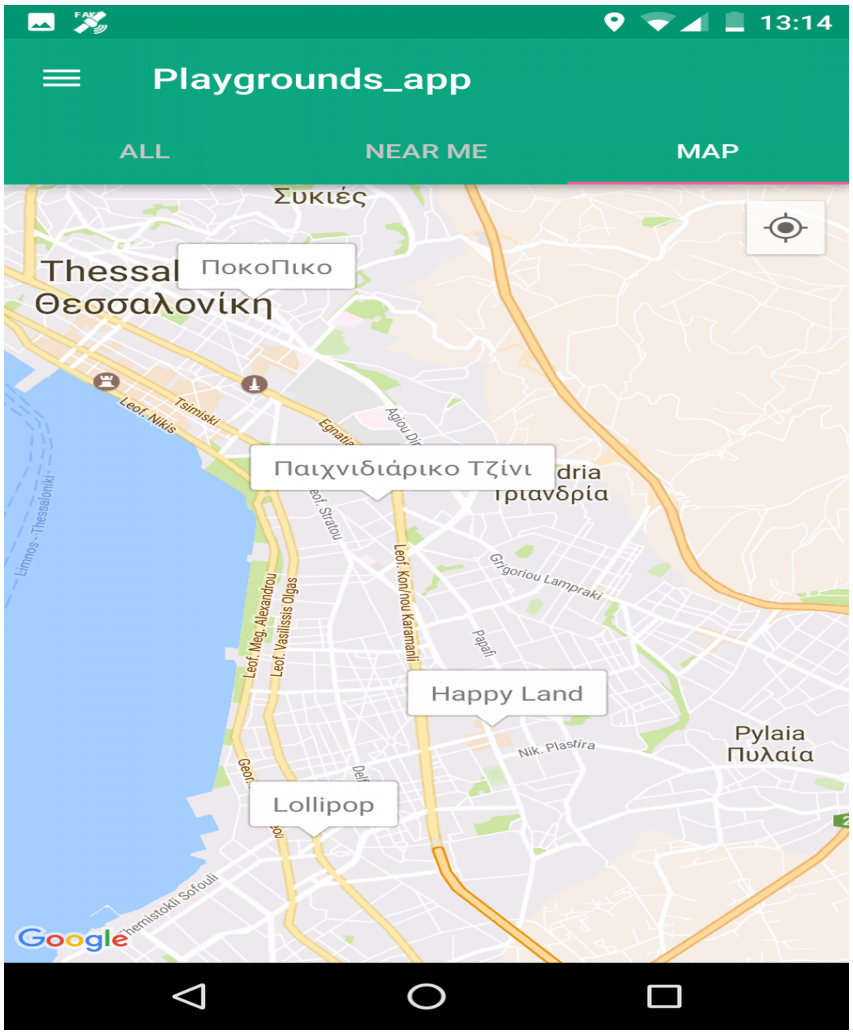
Εάν ο χρήστης μετακινηθεί στην NEAR ME καρτέλα, εμφανίζονται οι ίδιες πληροφορίες όπως και στην ALL καρτέλα αλλά αυτή τη φορά οι παιδότοποι είναι ταξινομημένοι βάσει χιλιομετρικής απόστασης από τη τοποθεσία του χρήστη. Επίσης, το πλαίσιο αναζήτης δεν είναι διαθέσιμο εδώ. Εικόνα 34.



Εικόνα 34: NEAR ME καρτέλα

7.6 MAP

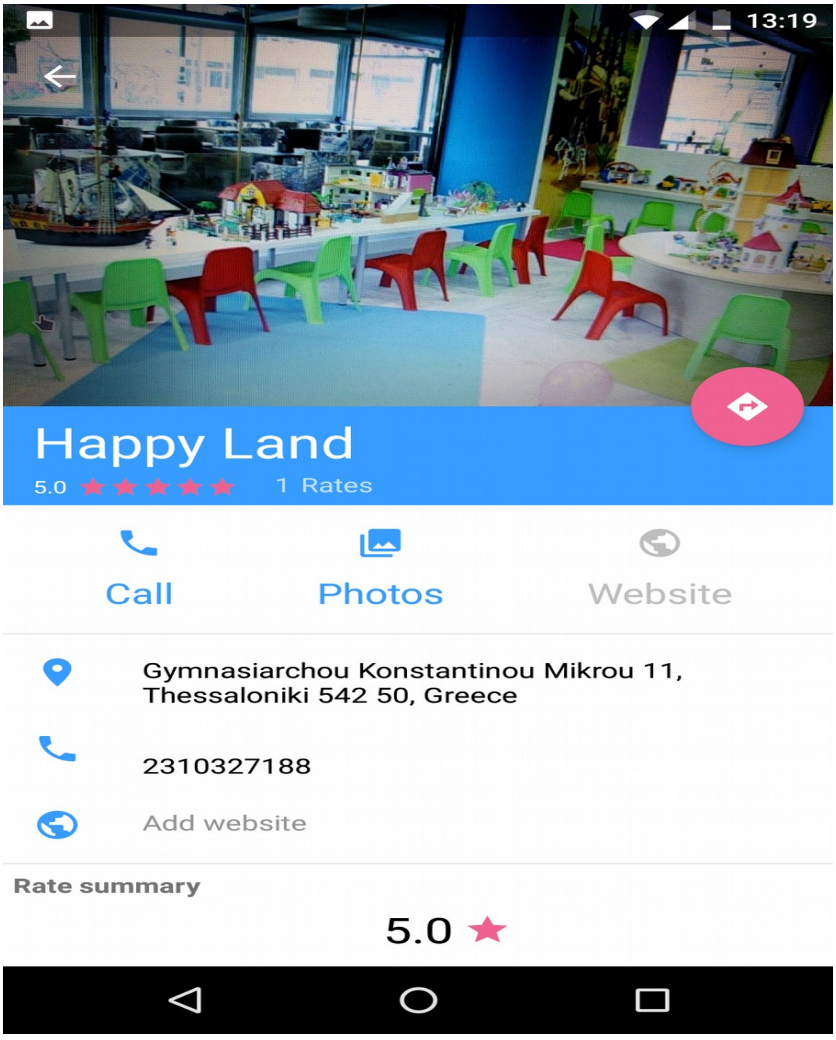
Η καρτέλα MAP αποτελείται από ένα Google Map στο οποίο φένεται η τοποθεσία του χρήστη και οι τοποθεσίες όλων των παιδότοπων με το όνομα τους. Ο χρήστης μπορεί να περιηγηθεί κανονικά στο χάρτη, του δίνεται η δυνατότητα να μεταφερθεί σε άλλη περιοχή είτε να εφαρμόσει zoom-in/out σε αυτόν. Εικόνα 35.



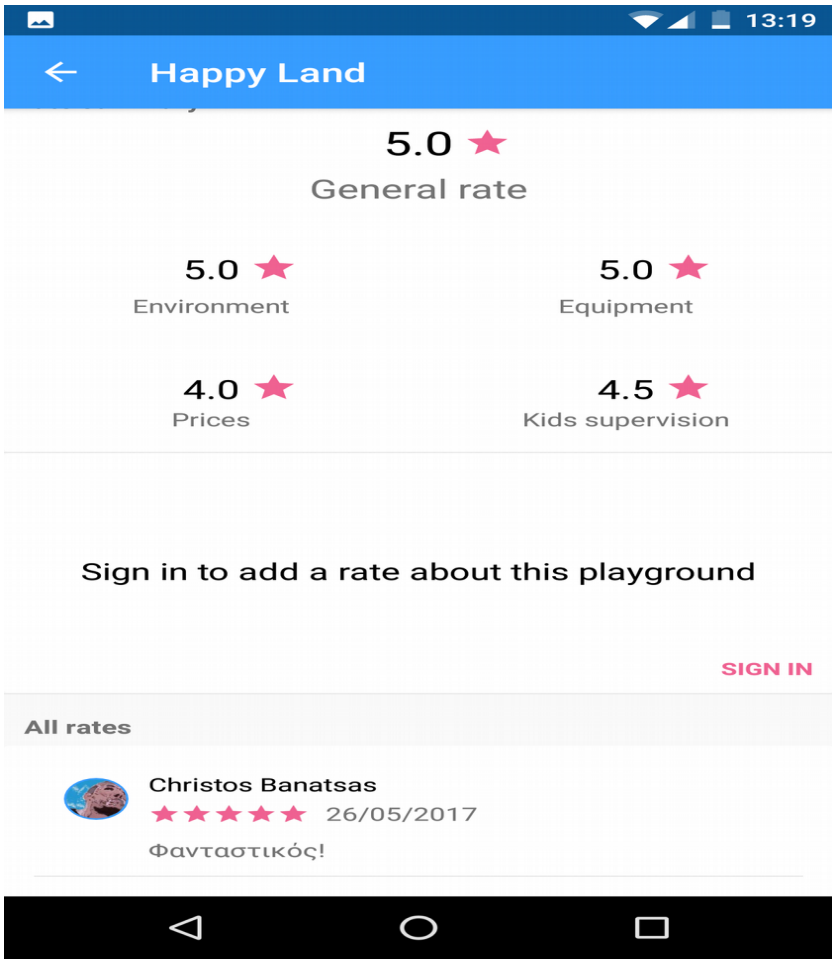
Εικόνα 35: MAP καρτέλα

7.7 Λεπτομέρειες παιδότοπου

Σε οποιαδήποτε από τις 3 καρτέλες εάν ο χρήστης επιλέξει έναν παιδότοπο μεταφέρεται σε μια νέα οθόνη στην οποία παρουσιάζονται λεπτομέρειες για το συγκεκριμένο παιδότοπο. Εικόνα 36, 37.



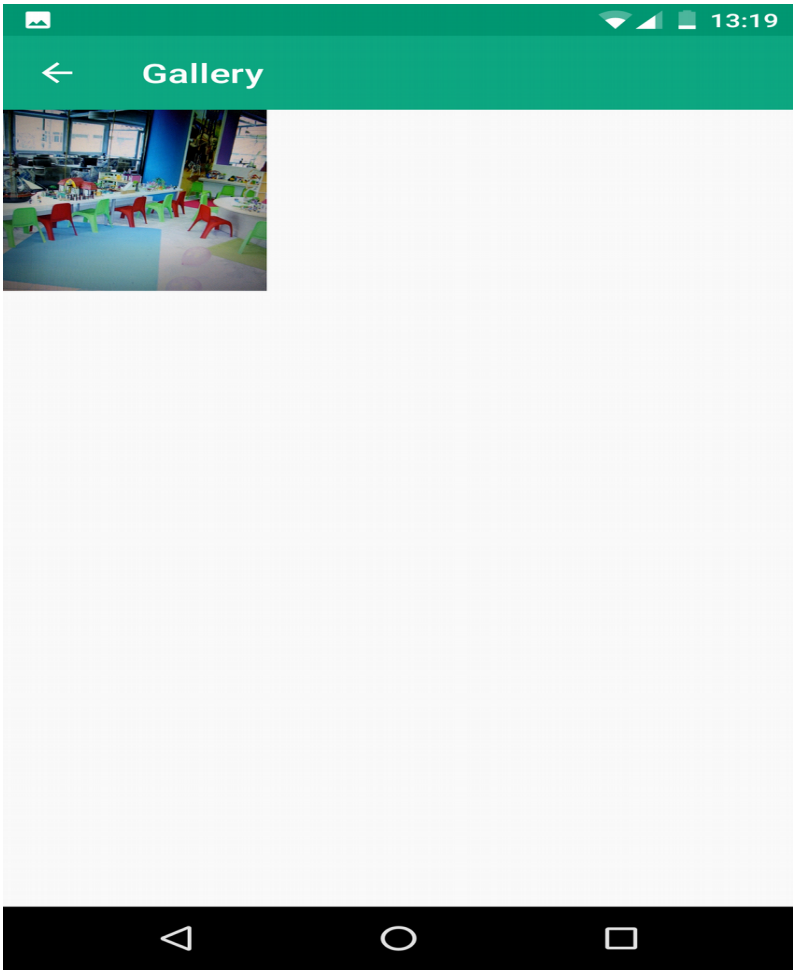
Εικόνα 36: Λεπτομέρειες παιδότοπου 1



Εικόνα 37: Λεπτομέρειες παιδότοπου 2

Οπως φένεται και από τις παραπάνω εικόνες οι λεπτομέρειες ενός παιδότοπου αποτελούνται από: τη διεύθυνση του παιδότοπου, τις οδηγίες (οι οποίες ανήγουν τη Google Maps εφαρμογή) για να βρεθεί εκεί ο χρήστης, το τηλέφωνο, τις φωτογραφίες, τον ιστότοπο, το σύνολο και το μέσο όρο των βαθμολογιών καθώς και τα σχόλια (εάν υπάρχουν) που έχουν αφήσει άλλοι χρήστες. Πιο συγκεκριμένα, εφόσον υπάρχει διαθέσιμο τηλέφωνο φωτογραφίες ή ιστότοπος για έναν παιδότοπο ο χρήστης επιλέγωντας κάποιο από τα εικονίδια *Call*, *Photos*, *Website* της εικόνας 36 θα εκτελεστεί και μια αντίστοιχη ενέργεια. Για το *Call* θα ανοίξει η εφαρμογή κλήσεων

έχωντας πλητρολογήσει το τηλέφωνο του παιδότοπου ώστε ο χρήστης να πραγματοποιήσει τη κλήση, για το *Photos* θα εμφανιστεί η συλλογή των φωτογραφιών για το συγκεκριμένο παιδότο, εικόνα 38. Η ίδια ενέργεια θα πραγματοποιηθεί εάν ο χρήστης πατήσει πάνω στην διαθέσιμη εικόνα, ενώ για το *Website* θα ανοίξει μια νέα καρτέλα με τον ιστότοπο του παιδότοπου στον προεπιλεγμένο broswer της συσκευής.

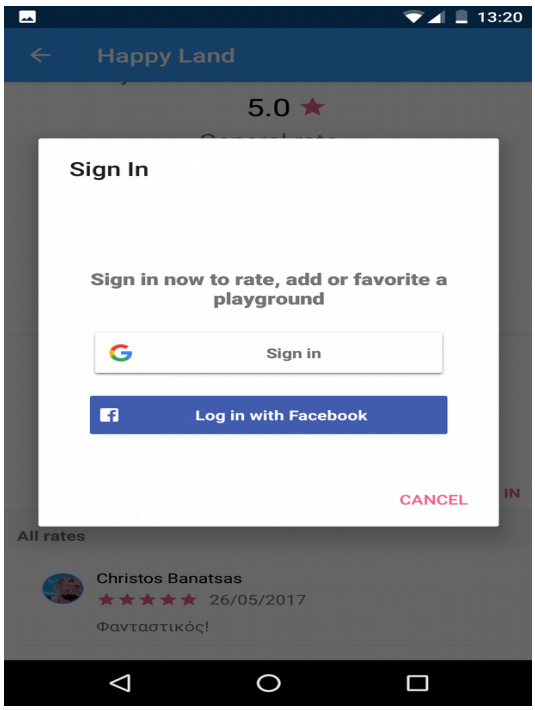


Εικόνα 38: Συλλογή φωτογραφιών για έναν παιδότοπο

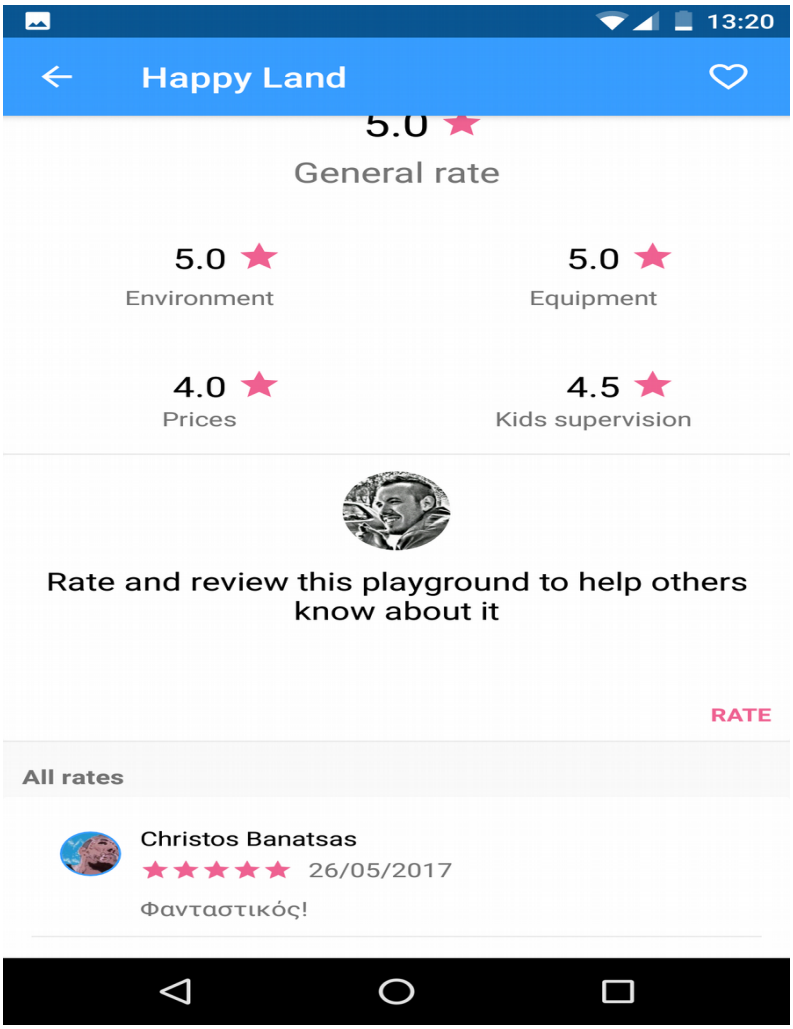
7.7.1 Σύνδεση στην εφαρμογή

Παρατηρούμε πως στην εικόνα 36 υπάρχει μια κάρτα η οποία περιέχει το μήνυμα: '*Sign in to add a rate about this playground*'. Πατώντας το *SIGN IN* θα εμφανιστεί ένα Dialog από το οποίο ο χρήστης μπορεί να συνδεθεί στην εφαρμογή μέσω του Facebook ή του Google λογαριασμού του, εικόνα 39. Εφόσον ο χρήστης συνδεθεί, το περιεχόμενο της συγκεκριμένης κάρτας θα αλλάξει και πλέον θα περιέχει τη φωτογραφία του, και ένα μήνυμα το οποίο θα τον προτρέπει να βαθμολογήσει το παιδότοπο. Εικόνα 40. 💬

Εκτός από την αλλαγή στη συγκρεκριμένη κάρτα, εφόσον ένας χρήστης συνδεθεί, θα προστεθεί και ένα νέο εικονίδιο(favorite) στο Toolbar από το οποίο ο χρήστης μπορεί να εισάγει το συγκεκριμένο παιδότοπο στα αγαπημένα του και ένα νέο πεδίο '*Edit Current info'* από το οποίο μπορεί να ανανεώσει τις πληροφορίες κάποιου παιδότοπου.



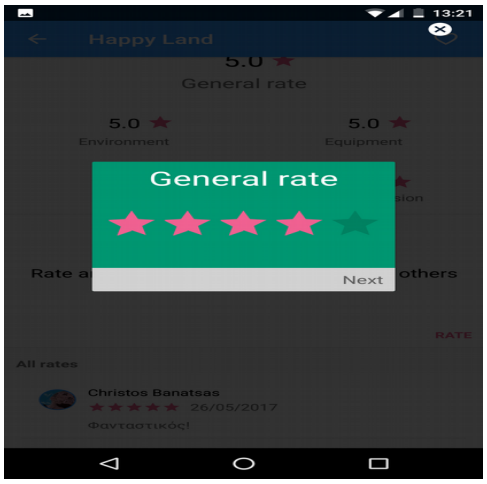
Εικόνα 39: Σύνδεση στην εφαρμογή



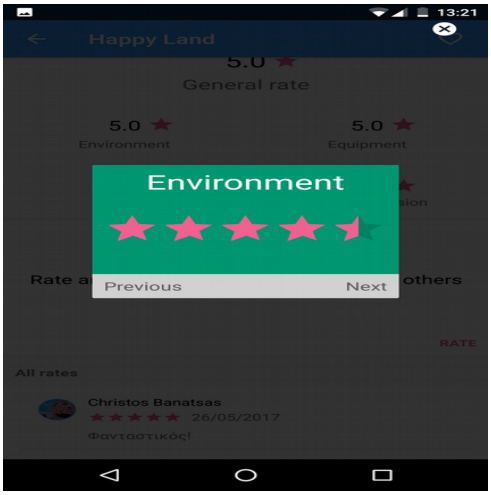
Εικόνα 40: Επιτυχής σύνδεση

7.7.2 Βαθμολόγηση παιδότοπου

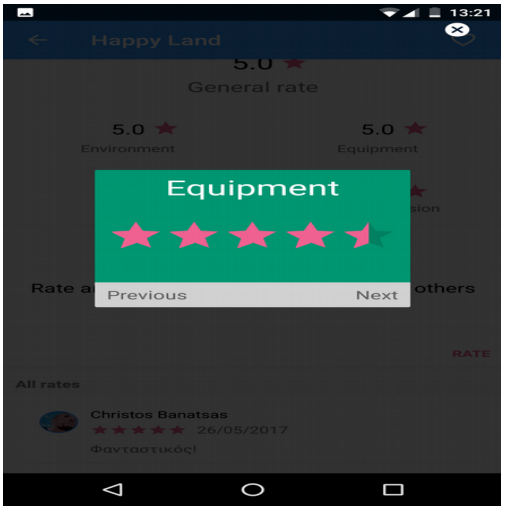
Εάν ο χρήστης επιλέξει να βαθμολογήσει το παιδότοπο εμφανίζονται κάποια Dialogs τα οποία ζητάνε μια βαθμολογία για διάφορες κατηγορίες του παιδότοπου καθώς και ένα προαιρετικό σχόλιο. Εικόνες 41, 42, 43, 44, 45, 46. Ο χρήστης μπορεί να πλοηγείται στα παραπάνω Dialogs επιλέγοντας το *Next/Previous* που υπάρχουν στο καθένα.



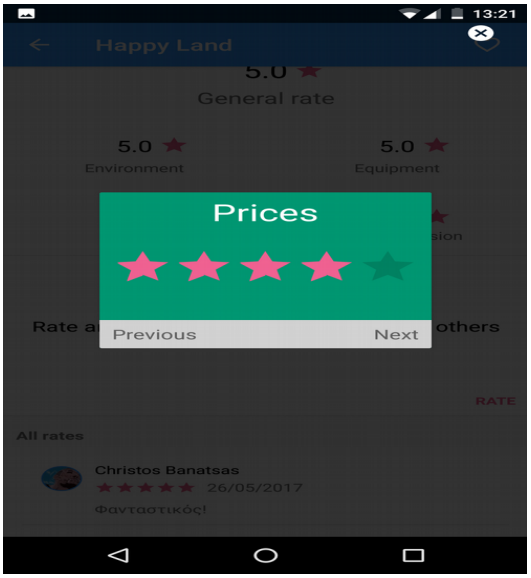
Εικόνα 41: Γενική βαθμολόγηση



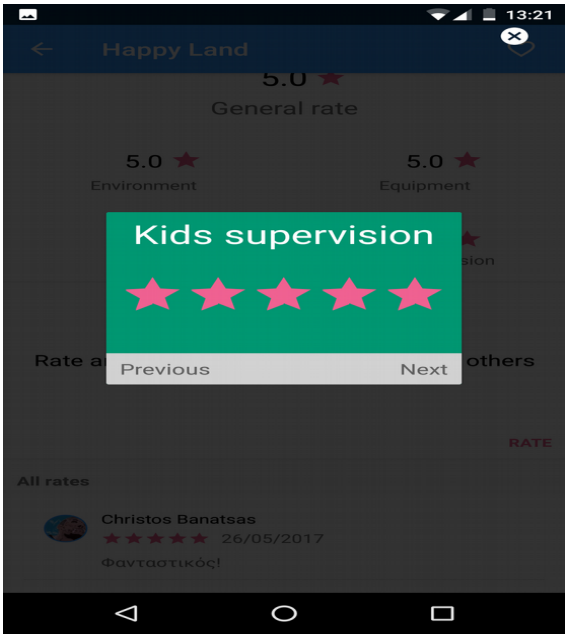
Εικόνα 42: Βαθμολόγηση περιβάλλοντος



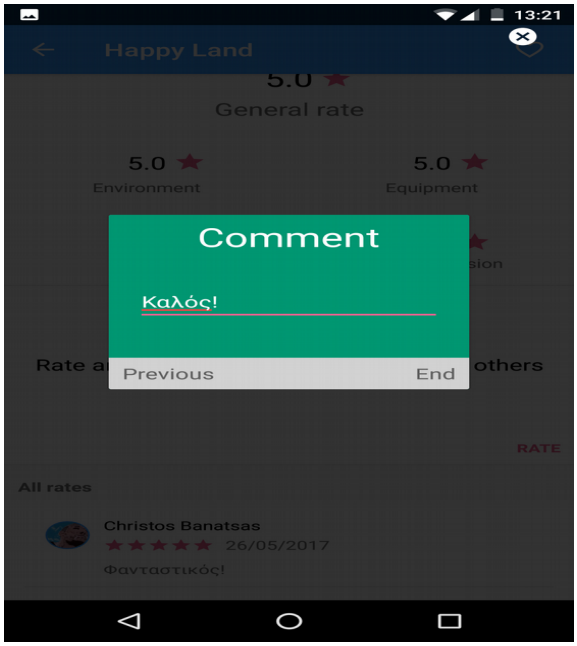
Εικόνα 43: Βαθμολόγηση εξοπλισμού



Εικόνα 44: Βαθμολόγηση τιμών

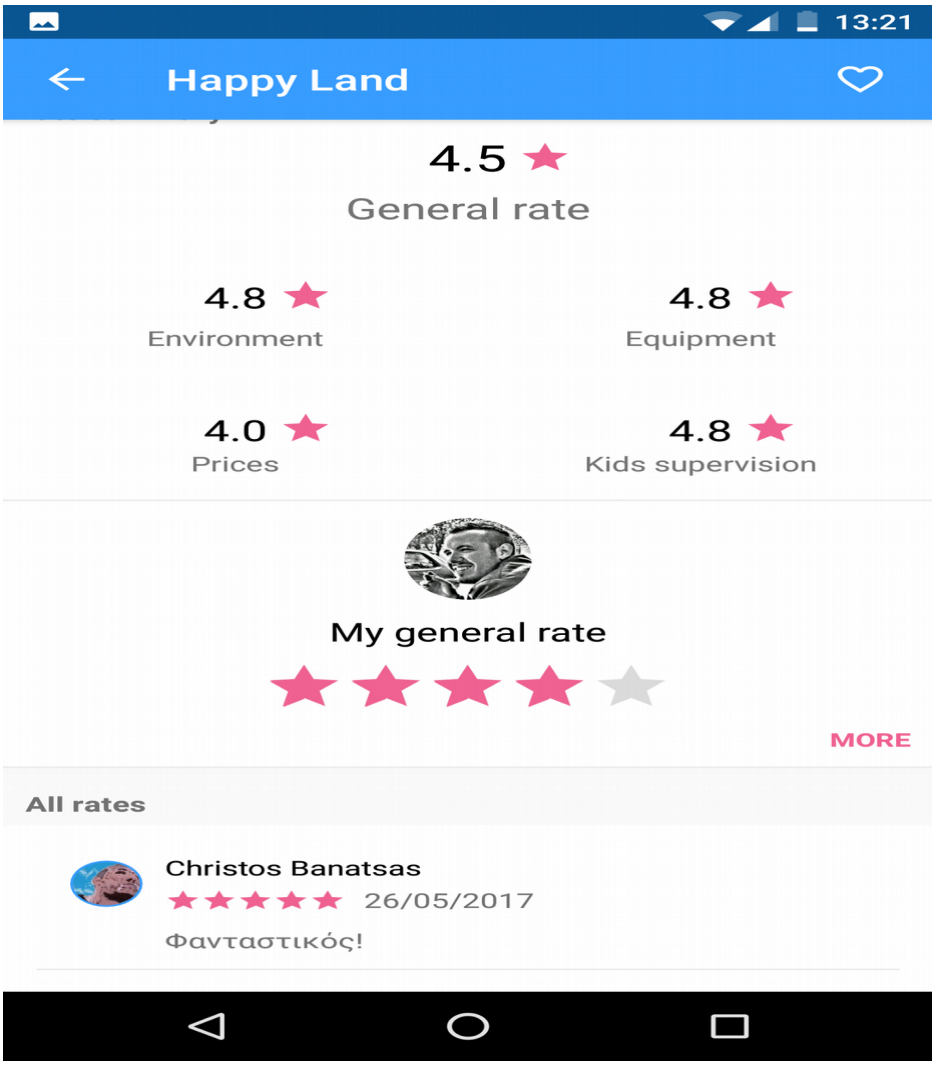


Εικόνα 45: Βαθμολόγηση επίβλεψης παιδιών

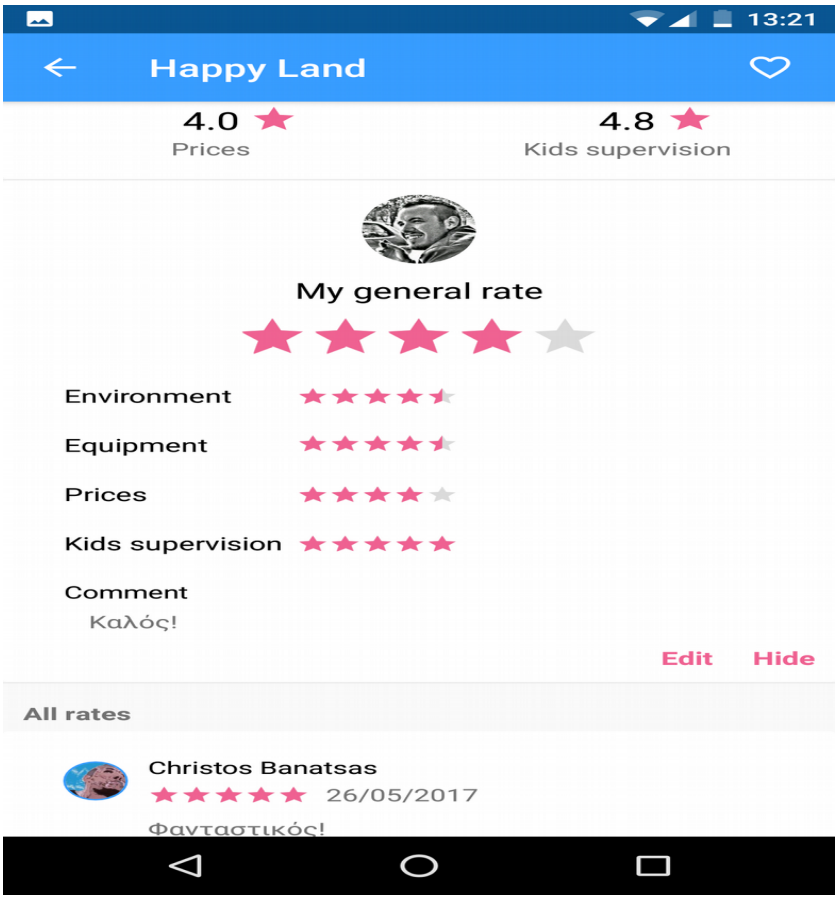


Εικόνα 46: Σχόλιο

Μετά τη βαθμολόγηση ενός παιδότοπου η κάρτα της βαθμολογίας του χρήστη θα ξανά αλλάξει, αυτή τη φορά θα περιέχει τη γενική βαθμολογία του και πατώντας στο *SHOW MORE* η κάρτα επεκτείνεται και παρουσιάζει τη βαθμολογία για κάθε κατηγορία ξεχωριστά καθώς και την επιλογή ενημέρωση της πατώντας στο *UPDATE*.
Εικόνες 47, 48.



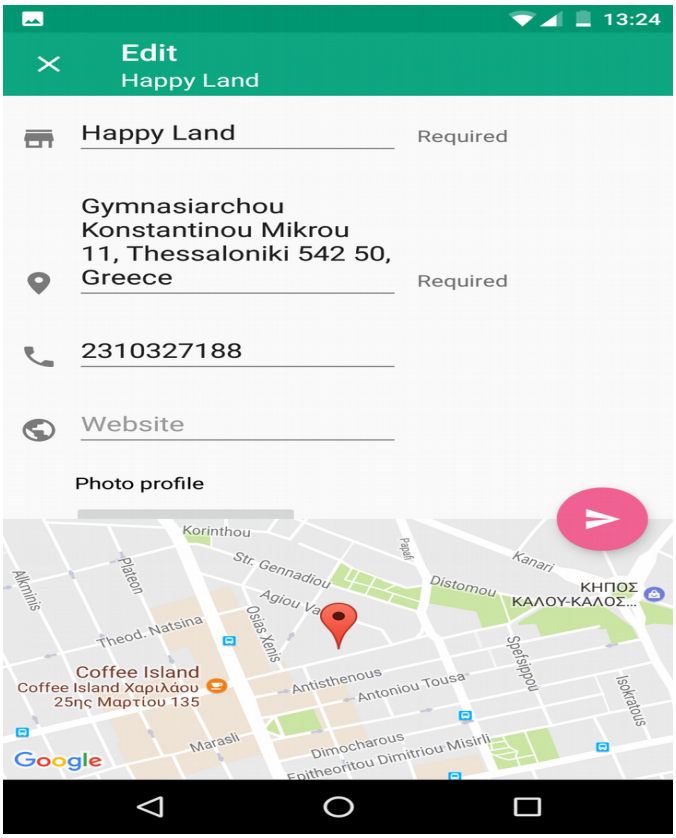
Εικόνα 47: Γενική βαθμολογία χρήστη



Εικόνα 48: Συνολική βαθμολογία χρήστη

7.7.3 Επεξεργασία πληροφοριών παιδότοπου

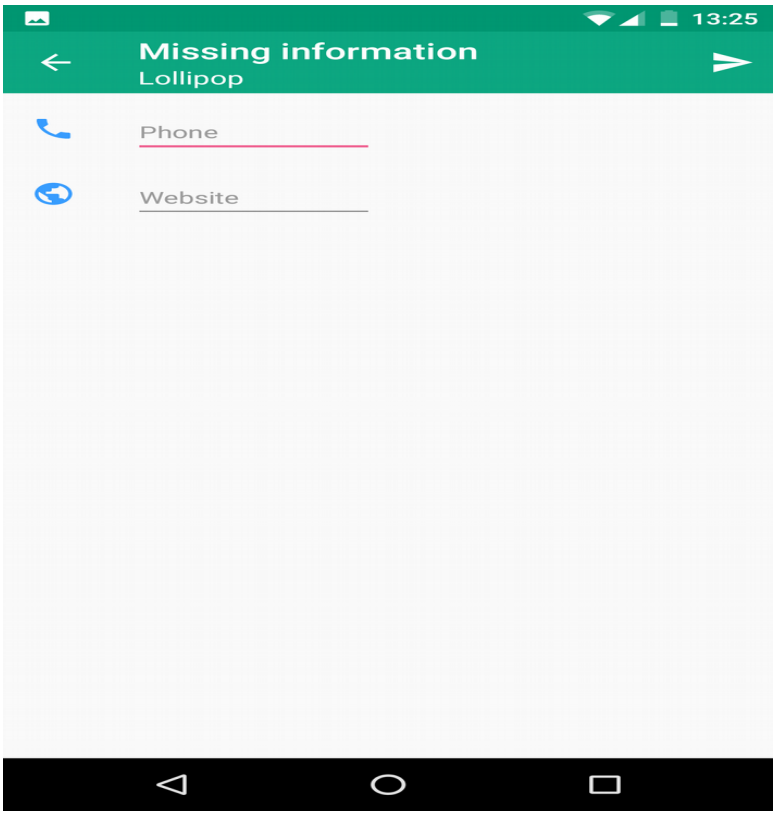
Ένας συνδεδεμένος χρήστης επιλέγοντας το '*Edit Current info*' πεδίο μεταφέρεται σε μια νέα οθόνη στην οποία υπάρχει μια φόρμα συμπλήρωσης η οποία περιέχει τα στοιχεία του παιδότοπου και έναν χάρτη με ένα σημείο το οποίο αναπαριστά τη διεύθυνση του παίδοτοπου στο χάρτη. Αλλάζοντας σημείο στο χάρτη ενημερώνεται και το αντίστοιχο πεδίο διεύθυνσης στη φόρμα καθώς και το αντίστροφο, αλλάζοντας διεύθυνση στη φόρμα ενημερώνεται και το σημείο στο χάρτη. Εικόνα 49.



Εικόνα 49: Επεξεργασία πληροφοριών παιδότοπου

7.7.4 Συμπλήρωση πεδίων που λείπουν

Ένας συνδεδεμένος χρήστης μπορεί επίσης να προσθέσει το τηλέφωνο ή/και τον ιστότοπο ενός παιδότοπου(εφόσον λείπουν) πατώντας στο αντίστοιχο 'Add phone' ή 'Add website' (Εικόνα 36), από τα οποία μεταφέρονται σε μια νέα οθόνη για τη συμπλήρωση αυτών. Εικόνα 50.



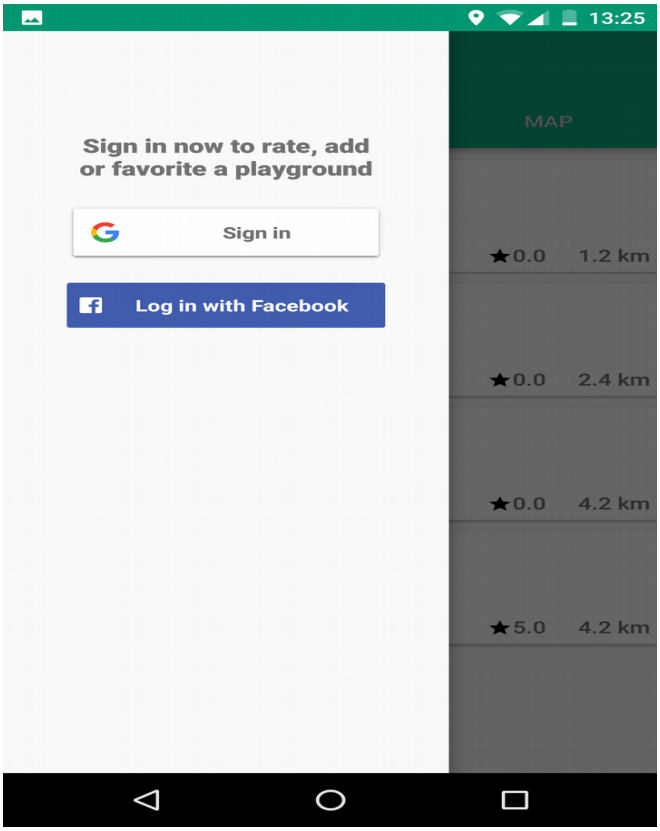
Εικόνα 50: Συμπλήρωση πεδίων που λείπουν

7.8 Λειτουργίες χρήστη

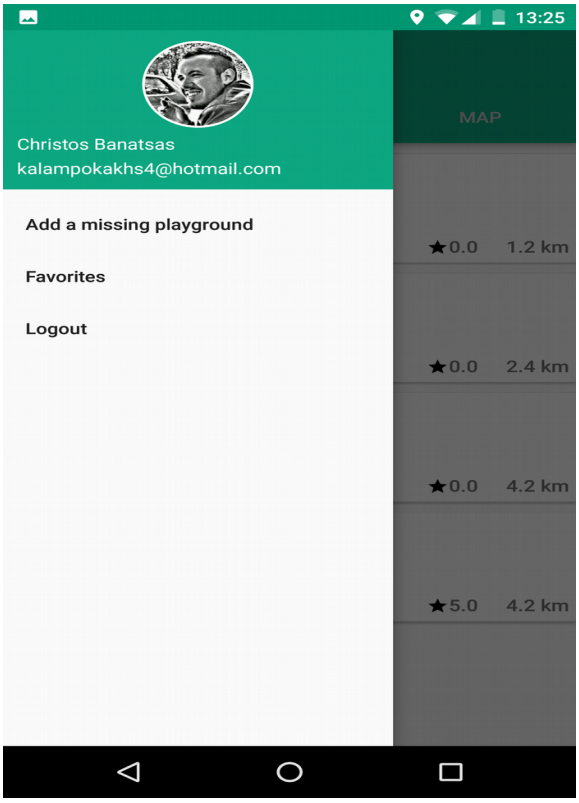
Επιστρέφοντας στη κύρια οθόνη της εφαρμογής, εικόνα 30, ένας χρήστης πατώντας στις τρεις οριζόντιες γραμμές (εικονίδιο αριστερά από το όνομα της εφαρμογής) ή κάνωντας swipe left από την καρτέλα ALL θα εμφανιστεί ένα νέο πλαίσιο απο το οποίο εάν δεν είναι συνδεδεμένος στην εφαρμογή, μπορεί να συνδεθεί ή εάν έχει ήδη συνδεθεί μπορεί να πραγματοποιήσει τις εξείς ενέργειες:

- Να εισάγει έναν νέο παιδότοπο.
- Να δει τα αγαπημένα του.
- Να αποσυνδεθεί.

Εικόνες 51, 52.

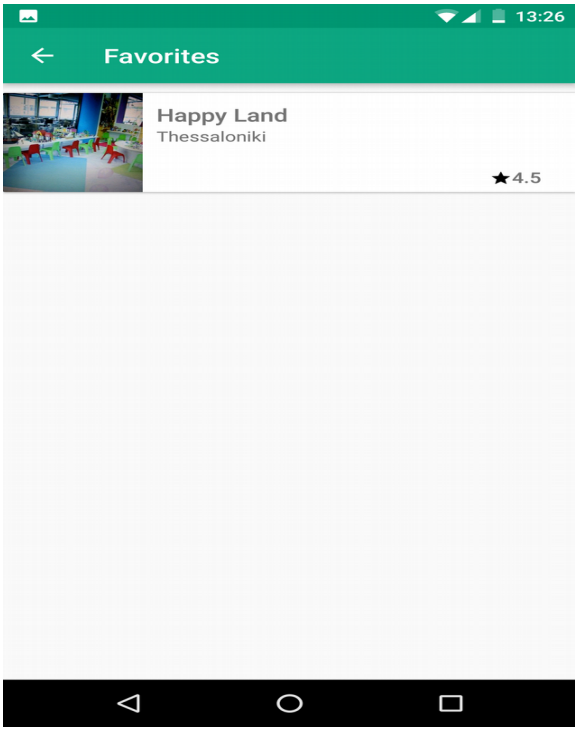


Εικόνα 51: Σύνδεση στην εφαρμογή



Εικόνα 52: Συνδεδεμένος χρήστης

Επιλέγοντας ο χρήστης να καταχωρήσει έναν νέο παιδότοπο θα μεταφερθεί στην ίδια οθόνη με αυτή του *Edit current Info* με τη διαφορά ότι η φόρμα θα έχει συμπληρωμένο μόνο το πεδίο της διεύθυνσης και το σημείο στο χάρτη με τη τρέχων τοποθεσία του χρήστη. Τέλος επιλέγοντας να δει τα αγαπημένα του, εμφανίζεται μια οθόνη παρόμοια με την καρτέλα ALL στην οποία υπάρχουν όλοι οι αγαπημένοι του παιδότοποι σε μορφή καρτών. Εικόνα 53.



Εικόνα 53: Αγαπημένα χρήστη

7.9 Επίλογος

Σε αυτό το κεφάλαιο έγινε μια αναλυτική παρουσίαση της λειτουργίας και χρήσης της εφαρμογής με χρήση στιγμιοτύπων οθόνης και αναλυτική περιγραφή του καθενός. Ακολουθούν τα συμπεράσματα και οι μελλοντικές επεκτάσεις της εφαρμογής.

8 Επίλογος

8.1 Σύνοψη και συμπεράσματα

Στη παρούσα πτυχιακή εργασία αναλύθηκε μια εφαρμογή για κινητά τηλέφωνα τα οποία χρησιμοποιούν το λειτουργικό σύστημα Android και η οποία επικοινωνεί με ένα REST API. Αποκομίσθηκαν σημαντικές γνώσεις τόσο για τη σχεδίαση όσο για τον προγραμματισμό Android εφαρμογών και REST API's με τη βοήθεια του Spring framework.

Τα κινητά τηλέφωνα καθιερώνονται όλο και περισσότερο στη καθημερινότητα των ανθρώπων, ενώ τα REST API's καθιερώνονται όλο και περισσότερο στις προτιμήσεις των προγραμματιστών για τη χρησιμοποίησή τους στη πλευρά του εξυπηρετητή. Πρόκειται για δύο τεράστιους τομείς άκρως δημοφιλείς και ενδιαφέρον με τους οποίους αξίζει κάποιος να επενδύσει το χρόνο του και να ασχοληθεί.

8.2 Μελλοντικές επεκτάσεις

Ο τρόπος με τον οποίο έχει υλοποιηθεί η εφαρμογή επιτρέπει την προσθήκη νέων χαρακτηριστικών.

Θα μπορούσε να προστεθεί στην εισαγωγή ενός νέου παιδότοπου ένα πεδίο το οποίο θα αναπαριστά την επίσημη σελίδα του παιδότοπου στο Facebook ή ακόμη και να υποστηρίζεται η προσθήκη νέου παιδότοπου από αναζήτηση στο Facebook. Τέλος θα ήταν χρήσιμη η λειτουργία για αναφορά ενός παιδότοπου με τη κατάλληλη αιτιολόγηση(ψευδή στοιχεία, λάθος τοποθεσία κ.α).

9 Βιβλιογραφία και εξωτερικοί σύνδεσμοι

9.1 Εξωτερικοί σύνδεσμοι

[1] https://maven.apache.org/

[2] http://groovy-lang.org/

[3] https://docs.gradle.org/current/userguide/[4]installation.html

[5] https://docs.mongodb.com/manual/installation/

[6] https://docs.mongodb.com/manual/indexes/

[7] https://docs.mongodb.com/manual/reference/program/[8]mongofiles/

[9] https://en.wikipedia.org/wiki/Aspect-oriented_programming

[10] https://en.wikipedia.org/wiki/Portlet

[11] https://en.wikipedia.org/wiki/Unit_testing

[12] https://en.wikipedia.org/wiki/Integration_testing

[13] https://www.jetbrains.com/idea/

[14] https://www.getpostman.com/

[15] https://filezilla-project.org/

[16] https://material.io/guidelines/

[17] https://www.polymer-project.org/

[18] https://en.wikipedia.org/wiki/God_object

[19] https://console.developers.google.com

[20] https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

[21] https://en.wikipedia.org/wiki/Observer_pattern

[22] https://en.wikipedia.org/wiki/Iterator_pattern

[23] https://en.wikipedia.org/wiki/Functional_programming

[24] http://reactivex.io/documentation/operators.html

[25] https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

9.2 Βιβλιογραφία

http://web.archive.org/web/20130116005443/http://tomayko.com/writings/rest-to-my-wife

https://en.wikipedia.org/wiki/Representational_state_transfer

https://docs.gradle.org/

https://en.wikipedia.org/wiki/Gradle

http://www.json.org/

https://docs.mongodb.com/manual/

https://en.wikipedia.org/wiki/NoSQL

https://www.javacodegeeks.com/2015/07/mysql-vs-mongodb.html

https://en.wikipedia.org/wiki/Basic_access_authentication

https://developers.google.com/maps/documentation/geocoding/intro

https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html

https://en.wikipedia.org/wiki/Spring_Framework

https://dzone.com/articles/why-springboot

https://projects.spring.io/spring-boot/

http://projects.spring.io/spring-data/

https://projects.spring.io/spring-security/

https://el.wikipedia.org/wiki/Android

https://developer.android.com/guide/platform/index.html

https://developer.android.com/develop/index.html

https://antonioleiva.com/mvp-android/

http://www.javapractices.com/topic/TopicAction.do?Id=205

https://el.wikipedia.org/wiki/Java

https://el.wikipedia.org/wiki/XML

https://developers.facebook.com/docs/android/

https://developers.google.com/identity/sign-in/android/start-integrating

http://greenrobot.org/eventbus/

http://square.github.io/retrofit/

https://github.com/ReactiveX/RxJava

http://blog.danlew.net/2014/09/15/grokking-rxjava-part-1/

http://reactivex.io/documentation/operators/filter.html

http://reactivex.io/documentation/operators/map.html

http://reactivex.io/documentation/operators/flatmap.html

Παράρτημα Α

Παρατίθεται το σχήμα της βάσης δεδομένων που χρησιμοποιήθηκε σε αυτή τη πτυχιακή εργασία και πιο συγκρεκριμένα στη μεριά του εξυπηρετητή, το REST API.
Αποτελείται απο τέσσερεις συλλογές (Collections). Την *playground* που αφορά τους παιδότοπους, την *user* που αφορά τους χρήστες και τις *fs.chunks, fs.files* οι οποίες είναι όπως παρουσιάστηκαν στο κεφάλαιο *GridFS* της MongoDB.

Playground Συλλογή

Ένα Έγγραφο στη playground Συλλογή ακολουθεί τη παρακάτω φόρμα:


```
{
    "_id" : ObjectId("58fdd58e73c9510c89eb45f2"),
    "_class" : "com.playgrounds.api.playground.model.Playground",
    "name" : "Playground_name",
    "city" : "thessaloniki",
    "address" : "Olumpou 90",
    "location" : {
        "type" : "Point",
        "coordinates" : [
            40.6638848,
            22.8049416
        ]
    },
    "phone" : NumberLong("2310876780"),
    "website" : "http://www.google.gr",
    "popularity" : 0.2,
    "date_added" : ISODate("2017-04-24T10:38:05.848Z"),
    "added_by" : "58bc714e73c95160c01e5e92",
    "imageURL" : "http://83.212.122.113:8080/playgrounds/images/58148adecce2c09ffda9d1c8",
    "images" : [ ],
    "rates_num" : 1,
    "general_rate" : 4,
    "general_environment" : 5,
```

```
    "general_equipment" : 4,
    "general_prices" : 4,
    "general_kids_supervision" : 5,
    "rate" : [
        {
            "user" : "58d2d5f773c95114c98bb108",
            "date" : ISODate("2017-05-25T14:39:44.393Z"),
            "general_rate" : 4,
            "environment" : 5,
            "equipment" : 4,
            "prices" : 4,
            "kids_supervision" : 5,
            "comment" : "OK"
        }
    ]
}
```


Κάθε έγγραφο της playground συλλογής αποτελείται από τα παρακάτω πεδία:

- **_id**

  Το μοναδικό ID κάθε παιδότοπου.

- **_class**

  Η Java κλάση από η οποία αντιστοιχεί σε αυτο το έγγραφο και απο την οποία δημιουργήθηκε.

- **name**

  Το όνομα του παιδότοπου.

- **city**

  Το όνομα της περιοχής κάθε παιδότοπου.

- **address**

  Η διεύθυνση του παιδότοπου.

- **location**

  Η γεωγραφική τοποθεσία του παιδότοπου. Ενα πεδίο το οποίο περιέχει ένα αντικείμενο με δύο πεδία. Το "type" και το "Coordinate". Το Coordinate προκειται για έναν πίνακα με τις γεωγραφικές συνταταγμένες του παιδότοπου. Το type θα εχει πάντοτε τη τιμή Point. Η μορφή αυτού του πεδίου είναι απαραίτητο να είναι πάντα αυτή καθώς το location αποτελεί ένα ευρετήριο τύπου "2dsphere" .

- **phone**

  Το τηλέφωνο του παιδότοπου.

- **website**

  Η ιστοσελίδα του παιδότοπου.

- **popularity**

  Η δημοτικότητα του παιδότοπου.

- **date_added**

  Η ημερομηνία που προστέθηκε ο παιδότοπος.

- **added_by**

  Το ID του χρήστη που πρόσθεσαι τον παιδότοπο.

- **imageURL**

  Το URL για την φωτογραφία προφίλ του παιδότοπου.

- **images**

  Λίστα με URL's με όλες τις υπόλοιπες διαθέσιμες φωτογραφίες του παιδότοπου

- **rates_num**

  Το σύνολο των βαθμολογιών του παιδότοπου.

- **general_rate**

  Ο μέσος όρος της γενική βαθμολογίας.

- **general_environment**

  Ο μέσος όρος τη γενικής βαθμολογίας για το περιβάλλον του παιδότοπου.

- **general_equipment**

  Ο μέσος όρος τη γενικής βαθμολογίας για τον εξοπλισμό του παιδότοπου.

- **general_kids_supervision**

  Ο μέσος όρος τη γενικής βαθμολογίας για την φύλαξη των παιδιών στο παιδότοπο.

- **general_prices**

  Ο μέσος όρος τη γενικής βαθμολογίας για τις τιμές του παιδότοπου.

- **rate**

  Λίστα με αντικείμενα με όλες τις βαθμολογίες από τους χρήστες. Κάθε αντικείμενο περιέχει το ID του χρήστη, την ημερομηνία που έγινε η βαθμολογία καθώς και την βαθμολογία για όλους τους τομείς του παιδότοπου μαζί με κάποιο σχόλιο.

User Συλλογή

Ένα Έγγραφο στη user Συλλογή ακολουθεί τη παρακάτω φόρμα:

```
{
    "_id" : ObjectId("58d2d5f773c95114c98bb108"),
    "_class" : "com.playgrounds.api.user.model.User",
    "username" : "chris",
    "email" : "chris@example.com",
    "favorites" : [    {
                        "playground" : "58e4ccf873c9510cb3de668e"
                       }
                ]
}
```

Κάθε έγγραφο της playground συλλογής αποτελείται από τα παρακάτω πεδία:

- **_id**
  Το μοναδικό ID κάθε χρήστη.

- **_class**
  Η Java κλάση από η οποία αντιστοιχεί σε αυτο το έγγραφο και απο την οποία δημιουργήθηκε.

- **username**
  Το ονομα χρήστη.

- **email**
  Το email του χρήστη.

- **favorites**
  Οι αγαπημένοι παιδότοποι του χρήστη. Λίστα από αντικείμενα τα οποία περιέχουν τα ID's των παιδότοπων.