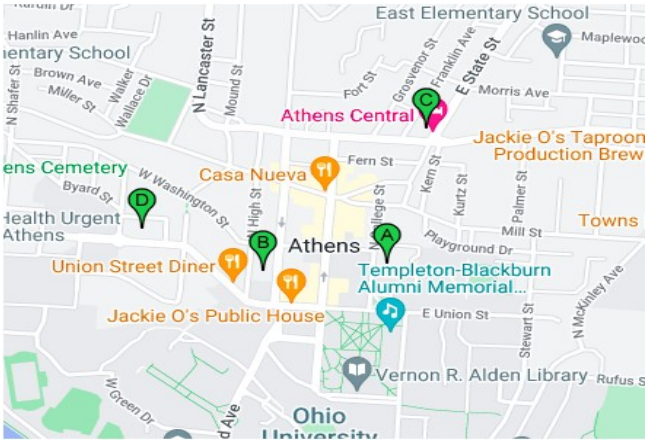
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΕΦΑΡΜΟΓΗ ΔΙΑΧΕΙΡΙΣΗΣ ΣΥΜΒΑΝΤΩΝ ΔΙΑΚΟΠΩΝ»



Του φοιτητή
Ιορδανίδη Βασιλείου
Αρ. Μητρώου: 144388

Επιβλέπων
Ιγνάτιος Δεληγιάννης
Βαθμίδα: Καθηγητής

Ημερομηνία 05/01/2023

Τίτλος Δ.Ε. Εφαρμογή διαχείρισης συμβάντων διακοπών
Κωδικός Δ.Ε. 21304
Ονοματεπώνυμο φοιτητή Βασίλειος Ιορδανίδης
Ονοματεπώνυμο εισηγητή Ιγνάτιος Δεληγιάννης
Ημερομηνία ανάληψης Δ.Ε. 22/09/2021
Ημερομηνία περάτωσης Δ.Ε. 05/01/2023

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

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

«Στην οικογένειά μου και τους φίλους μου»

Πρόλογος

Ο λόγος που επέλεξα αυτήν την εργασία ήταν για να έρθω σε επαφή με μοντέρνες τεχνολογίες που χρησιμοποιούνται για την ανάπτυξη εφαρμογών προκειμένου να τις κατανοήσω καλύτερα και να βελτιωθώ στο κομμάτι της ανάπτυξης λογισμικού. Η χρήση νέων αρχιτεκτονικών σχεδίασης όπως είναι το MVI και το DDD με βοήθησε στο να κατανοήσω την σημαντικότητά τους όσο αναφορά την συντήρηση και την επέκταση του κώδικα καθώς παρατήρησα αλλαγή στον τρόπο σκέψης μου κατά την διάρκεια της υλοποίησης μιας εφαρμογής, όσο αναφορά τη δομή του. Τέλος, η επιλογή στην χρήση του Docker για τη βάση δεδομένων μου φάνηκε ιδιαίτερα ενδιαφέρουσα επιλογή επειδή μου παρείχε πολύ μεγάλη απλότητα και ευελιξία.

Περίληψη

Το αντικείμενο της πτυχιακής εργασίας πραγματεύεται την ανάπτυξη μίας διαδικτυακής εφαρμογής κατά την οποία ο χρήστης θα μπορεί, βάσει GPS, να αντλεί πληροφορίες και να τις αποθηκεύει όπως την τοποθεσία του μέρους που βρίσκεται, φωτογραφίες, βαθμό ενδιαφέροντος και κριτικές για πιθανά σημεία ενδιαφέροντος. Η ανάπτυξη της εφαρμογής χωρίστηκε σε δύο σκέλη. Ένα σκέλος αποτελεί το κομμάτι του UI κατά το οποίο χρησιμοποιήθηκε το framework της Angular και η αρχιτεκτονική MVI. Το άλλο σκέλος αποτελεί το API και η βάση δεδομένων, στο οποίο χρησιμοποιήθηκαν η αρχιτεκτονική Domain Driven Design και τεχνολογίες όπως είναι η Java σε συνδυασμό με την βιβλιοθήκη Eclipse Vert.x και την ReactiveX για το API και Docker σε συνδυασμό με PostgreSQL για την βάση δεδομένων.

«Vacation’s event management app»

«Vasileios Iordanidis»

Abstract

The subject of this thesis is about developing a web application, in which, the user, provided with the help of GPS,will be able to retrieve and store information about their given location and also images, reviews, score about possible points of interest. The application is divided into two sections. One section consists of the UI, which was developed using a framework known as Angular and the MVI design pattern. The other section consists of the API and the database, where the Domain Driven Design was used as well technologies such as Java with the Eclipse Vert.X library and ReactiveX and as for the database Docker with PostgreSQL.

Ευχαριστίες

Ευχαριστώ πολύ την οικογένειά μου και τους φίλους μου που μου συμπαραστάθηκαν κατά την διάρκεια της υλοποίησης της πτυχιακής μου εργασίας καθώς και τον κ. Δεληγιάννη που με εμπιστεύτηκε δίνοντάς μου την ευκαιρία να συνεργαστούμε παρέχοντας μου το θέμα της εργασίας.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
UI	User Interface
DDD	Domain Driven Design
MVI	Model View Intent
MVC	Model View Controller
MVP	Model View Presenter
DI	Dependency Injection
API	Application Programming Interface
GPS	Global Positioning System

Κεφάλαιο 1ο: Εισαγωγή

1.1 Γενική Περιγραφή Εφαρμογής

Η τρέχουσα πτυχιακή εργασία έχει να κάνει με την δημιουργία εφαρμογής διαχείρισης συμβάντων διακοπών. Κατά την χρήση της εφαρμογής ο χρήστης έχει την δυνατότητα να ψάξει για διάφορες πόλεις της Ελλάδας και να δει τα αξιοθέατα για κάθε μία από αυτήν. Στην συνέχεια δίνεται η δυνατότητα να αποθηκεύσει τα επιθυμητά αξιοθέατα, εφόσον προηγουμένως έχει δημιουργήσει λογαριασμό και είναι συνδεδεμένος σε αυτόν, μπορώντας έτσι να τα ανακτήσει αργότερα.

1.2 Δομή Κεφαλαίων

Η δομή της πτυχιακής χωρίζεται σε δώδεκα κεφάλαια. Στο πρώτο κεφάλαιο περιγράφεται η έννοια του Docker και των πλεονεκτημάτων του κάνοντας αναφορά στην αρχιτεκτονική του.

Στο δεύτερο κεφάλαιο αναφέρεται η έννοια του API επισημαίνοντας επιμέρους έννοιες που το απαρτίζουν όπως η βάση δεδομένων, η γλώσσα συγγραφής τους και το build tool της.

Στο τρίτο κεφάλαιο γίνεται περιγραφή του Eclipse Vert.x, του εργαλείου που χρησιμοποιήθηκε προκειμένου να αναπτυχθεί το API αναφέροντας τα κύρια δομικά στοιχεία του.

Στο τέταρτο κεφάλαιο σχολιάζεται η βιβλιοθήκη ReactiveX και συγκεκριμένα η συλλογή που αφορά την Java, δηλαδή η RxJava εξηγώντας βασικές έννοιες και τύπους.

Στο πέμπτο κεφάλαιο εξηγείται η αρχιτεκτονική Domain Driven Design περιγράφοντας βασικές έννοιες και αναφέροντας την αντιστοιχία αυτών των εννοιών με την εφαρμογή.

Στο έκτο κεφάλαιο σχολιάζεται το framework της Angular, αναφέροντας τις βασικές έννοιες που το αποτελούν.

Στο έβδομο κεφάλαιο γίνεται αναφορά στην TypeScript, η οποία αποτελεί την γλώσσα που γράφεται η Angular.

Στο όγδοο κεφάλαιο περιγράφεται η βιβλιοθήκη της RxJs, η οποία αποτελεί τμήμα της ReactiveX, αναφέροντας σημαντικές έννοιες όπως είναι το Subject, το Observable και τα memory leaks.

Στο ένατο κεφάλαιο γίνεται σχολιασμός του προτύπου Model View Intent εξηγώντας τα δομικά του χαρακτηριστικά και τον κύκλο ζωής των ενεργειών.

Στο δέκατο κεφάλαιο γίνεται σχολιασμός κάποιων τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη της διεπαφής του χρήστη και προστίθενται εικόνες για καλύτερη κατανόηση.

Στο τελευταίο κεφάλαιο περιγράφονται τα συμπεράσματα και οι βελτιώσεις που προέκυψαν από την ανάπτυξη της εφαρμογής.

Κεφάλαιο 2ο: Docker

2.1 Εισαγωγή

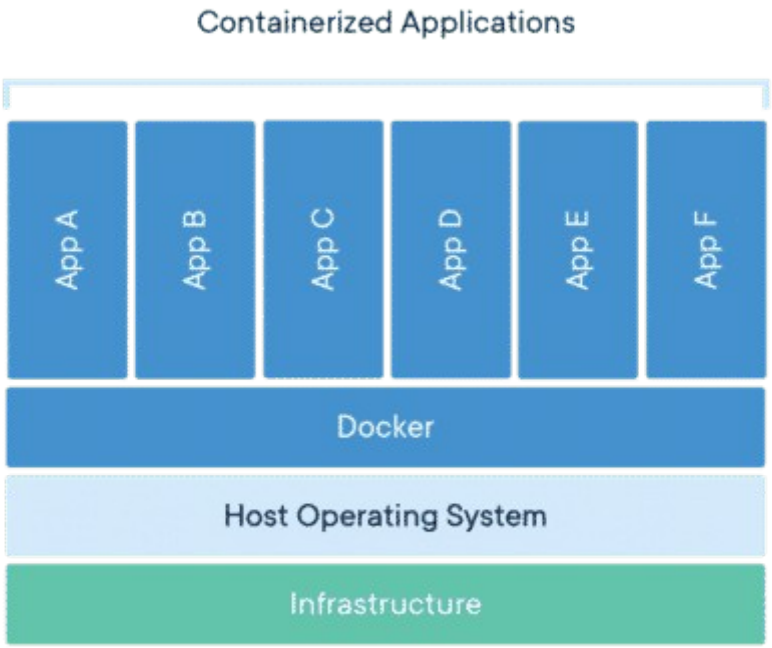
Το Docker είναι μία open source πλατφόρμα με την οποία πραγματοποιείται εικονοποίηση container, δηλαδή είναι σαν πολύ ελαφρύ Virtual Machine. Εκτός από την δημιουργία container το Docker παρέχει και μία ροή εργασίας μέσω της οποίας παρέχεται η δυνατότητα να δημιουργηθούν container τα οποία περιέχουν εφαρμογές μέσα σε άλλα containers και να τα μοιραστούν μεταξύ των προγραμματιστών.

2.2 Κύρια χαρακτηριστικά

Τα κύρια χαρακτηριστικά του είναι τα εξής τέσσερα στοιχεία. Το πρώτο στοιχείο αποτελεί το dockerfile που περιέχει τις εντολές προκειμένου να δημιουργηθεί το δεύτερο στοιχείο που λέγεται image. Το image αποτελεί το αρχείο που χρησιμοποιείται προκειμένου να εκτελεστεί το container, το τρίτο κύριο στοιχείο, το οποίο με τη σειρά του περιέχει τις εφαρμογές που θέλουμε. Στην περίπτωση που χρειάζεται να επικοινωνούν μεταξύ τους δύο container γίνεται χρήση του network, αποτελώντας έτσι το τελευταίο κύριο στοιχείο του Docker.

2.3 Πλεονεκτήματα και χρήσεις

Σημαντικό πλεονέκτημα του Docker αποτελεί η κατανάλωση πόρων. Συγκεκριμένα, χρησιμοποιώντας τεχνολογίες όπως αυτές του πυρήνα του Linux όπως είναι τα cgroups και kernel namespaces δίνεται η δυνατότητα σε ανεξάρτητα containers να εκτελούνται στο ίδιο λειτουργικό σύστημα έτσι αποφεύγεται η επιπλέον κατανάλωση πόρων που θα απαιτούσε ένα virtual machine. Παραδείγματος χάρη, ένα virtual machine τρέχει σε ένα λειτουργικό σύστημα και το ίδιο αποτελεί ένα λειτουργικό σύστημα ενώ αντιθέτως ένα container χρησιμοποιεί το host λειτουργικό, χρησιμοποιώντας αντίστοιχα τον πυρήνα του host διότι δεν διαθέτει το ίδιο δικό του. Με αυτόν τον τρόπο σπαταλώνται μόνο οι αναγκαίοι πόροι προκειμένου να τρέξουν τα container.



Εικόνα 1.1 Αρχιτεκτονική container

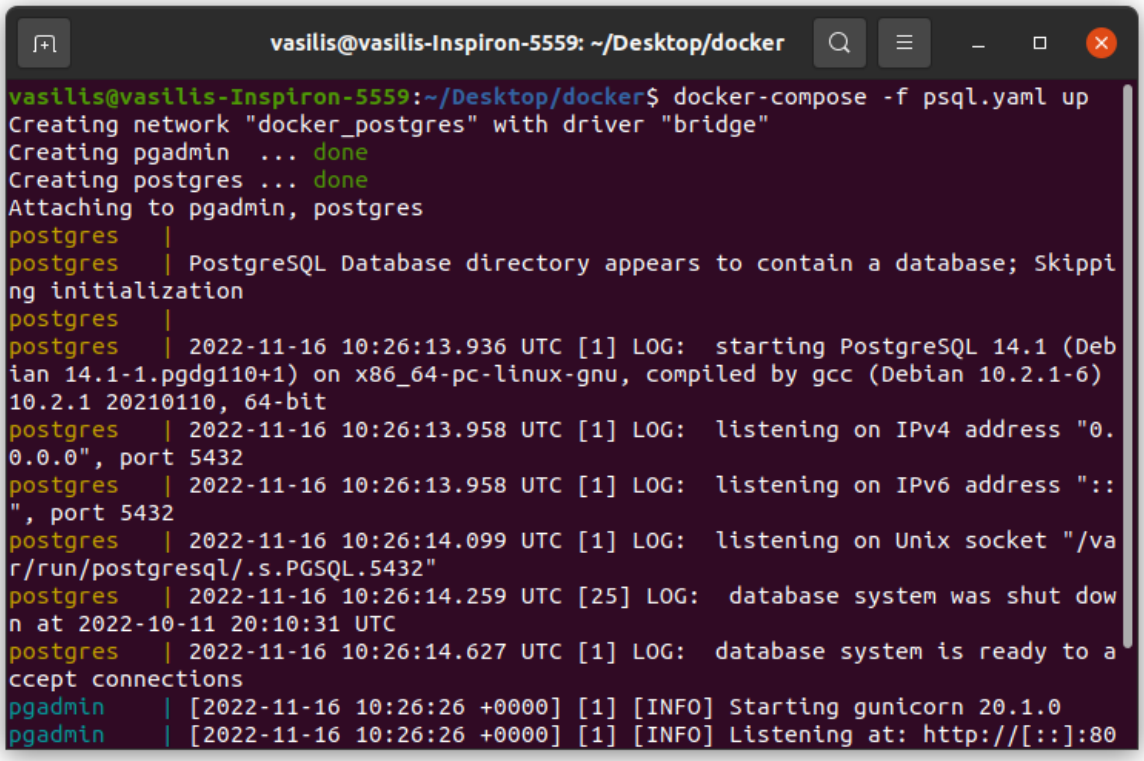
Άλλη χρήση του Docker που είναι ιδιαίτερα χρήσιμη αποτελεί το continuous integration και το continuous deployment (CI/CD). Με τον όρο continuous integration εννοείται η διαδικασία ενσωμάτωσης των αλλαγών στον κώδικα ενώ με τον όρο continuous deployment εννοείται η αυτοματοποίηση της έκδοσης του νέου λογισμικού που προκύπτει από τις αλλαγές. Αναλυτικότερα, ο προγραμματιστής μπορεί να μιμηθεί τον κώδικα παραγωγής πολύ γρήγορα με την βοήθεια του Docker. Παραδείγματος χάρη, έστω ότι χρησιμοποιείται μία στοίβα λογισμικού τύπου MEAN, δηλαδή αποτελείται από τις τεχνολογίες MongoDB, Express, Angular, και NodeJs, ο προγραμματιστής μπορεί χρησιμοποιώντας το Docker να δημιουργήσει το αντίστοιχο περιβάλλον ανάπτυξης πολύ εύκολα και γρήγορα με την χρήση των container δίχως να χρειαστεί η εγκατάσταση των τεχνολογιών στον υπολογιστή του. Επιπλέον, το ίδιο ισχύει και την περίπτωση αντικατάστασης των container με άλλα νέα καθώς με την χρήση του Docker η διαδικασία αυτή επισπεύδεται εξοικονομώντας έτσι πολύτιμο χρόνο.

2.4 Χρήση στην εφαρμογή

Στην συγκεκριμένη περίπτωση, προκειμένου να δημιουργηθεί η βάση δεδομένων δημιουργούμε ένα αρχείο yaml στο οποίο εισάγονται τα χαρακτηριστικά που πρόκειται να χρησιμοποιηθούν, σε μορφή key – value όπως φαίνεται στο παρακάτω σχήμα, περιγράφοντας πληροφορίες όπως το image που θα γίνει download από το docker image repository, μεταβλητές περιβάλλοντος όπως το port που θα τρέχει η βάση, συγκεκριμένη έκδοση PostgreSQL και ούτω κάθε εξής.

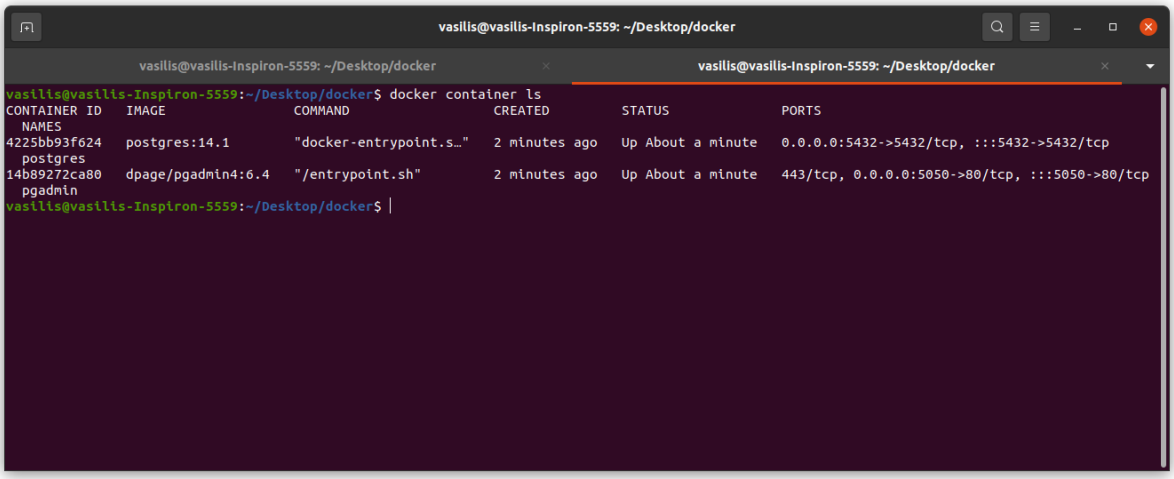
```yaml
version: "3.3"

services:

  psql:
    image: postgres:14.1
    container_name: postgres
    restart: always
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: ${POSTGRES_USER:-postgres}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-postgres}
      PGDATA: /data/postgres
    networks:
      - postgres
    volumes:
      - postgres:/data/postgres

  pgadmin:
    container_name: pgadmin
    image: dpage/pgadmin4:6.4
    restart: always
    ports:
      - "${PGADMIN_PORT:-5050}:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: ${PGADMIN_DEFAULT_EMAIL:-postgres@postgres.org}
      PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_DEFAULT_PASSWORD:-postgres}
    volumes:
      - pgadmin:/var/lib/pgadmin
    networks:
      - postgres

volumes:
  postgres:
  pgadmin:

networks:
  postgres:
    driver: bridge

```

Εικόνα 1.2 Παράδειγμα αρχείου YAML



Εικόνα 1.3 Εκτέλεση εντολής docker-compose

Εκτελώντας από το terminal την εντολή docker compose -f psql.yaml up, ενώ έχει γίνει η κατάλληλη αλλαγή στο αντίστοιχο directory που βρίσκεται το αρχείο, “σηκώνεται” το service που περιέχει την PostgresSQL και το αντίστοιχο UI tool δηλαδή το PgAdmin.



Εικόνα 1.4 Εμφάνιση ενεργών container

2.5 Επίλογος

Το Docker αποτελεί ένα εργαλείο εικονοποίησης container. Τα κύρια δομικά του στοιχεία είναι τα dockerfile, images και container. Παρέχει αρκετή ευελιξία στην ανάπτυξη λογισμικού λόγω των δυνατοτήτων που παρέχονται από τα container και την ικανότητά τους να προσομοιώνουν το

περιβάλλον της παραγωγής. Στην εφαρμογή χρησιμοποιώντας ένα αρχείο yaml μπορούμε να δημιουργήσουμε το περιβάλλον που λειτουργεί η βάση δεδομένων.

Κεφάλαιο 3ο: Application Program Interface

3.1 Εισαγωγή

Με τον όρο back end περιγράφεται η έννοια του server-side κομματιού στην ανάπτυξη μιας web εφαρμογής, και αποτελεί το τμήμα που δεν γίνεται να αλληλεπιδράσει ή να δει ο χρήστης. Στο συγκεκριμένο κομμάτι περιλαμβάνεται το API της εφαρμογής το οποίο είναι υπεύθυνο για την μεταφορά δεδομένων μεταξύ δεδομένων λογισμικού με έναν συγκεκριμένο τρόπο. Επίσης περιλαμβάνεται η βάση δεδομένων η οποία είναι υπεύθυνη για να αποθηκεύει τα επιθυμητά δεδομένα του χρήστη.

Το API αποτελεί κατά μία έννοια έναν μεσάζοντα μεταξύ προϊόντων λογισμικού και καθορίζει τον τρόπο που θα ανταλλάσσονται οι πληροφορίες μεταξύ τους χρησιμοποιώντας requests και responses. Υπάρχουν διάφορα είδη API όπως είναι τα SOAP API, τα οποί χρησιμοποιούν το Simple Object Access Protocol και ανταλλάσουν πληροφορίες χρησιμοποιώντας XML, τα WebSockets API τα οποία χρησιμοποιούν αντικείμενα JSON για την μεταφορά πληροφοριών και τα REST API που χρησιμοποιούν HTTP αιτήματα, και αποτελούν τα πιο διαδεδομένα.

Για την εφαρμογή χρησιμοποιείται ένα REST API εφόσον η αλληλεπίδραση μεταξύ client – server γίνεται μέσω http αιτημάτων όπως GET, POST, DELETE, και εφόσον έτσι δίνεται η δυνατότητα να χρησιμοποιηθεί η μνήμη cache για να αποφεύγονται άσκοπα αιτήματα, παρέχεται η ευελιξία στην επιλογή του τρόπου απεικόνισης των δεδομένων (JSON, XML, plain text), και γενικότερα είναι εύχρηστος τρόπος.

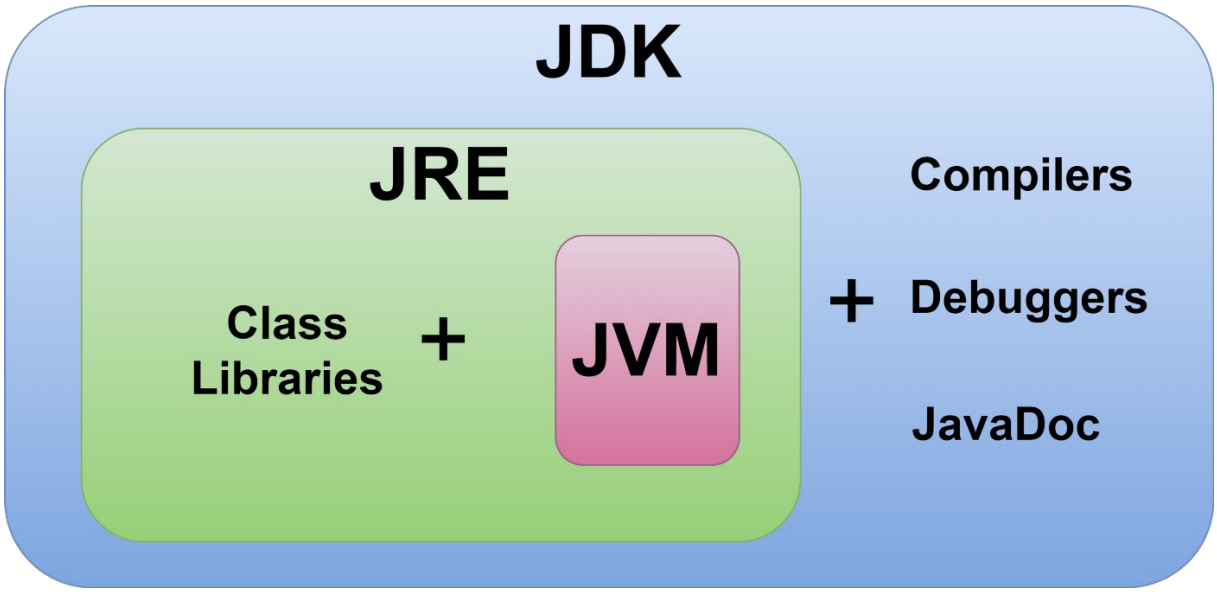
3.2 Βάση δεδομένων

Το πρώτο ζήτημα που προκύπτει στην ανάπτυξη της εφαρμογής είναι το είδος της βάσης δεδομένων που θα χρησιμοποιηθεί. Δεδομένου ότι τα δεδομένα δεν αλλάζουν συχνά και δεν προσθέτονται νέα χαρακτηριστικά και εφόσον είναι επιθυμητό οι ενέργειες που αφορούν τη βάση να πληρούν τις ACID (Atomicity, Consistency, Isolation, Durability) αρχές, επιλέγεται η χρήση σχεσιακής βάσης δεδομένων PostgresSQL. Αντιθέτως, εάν τα δεδομένα ήταν μεταβαλλόμενα ή προσθέτονταν συχνά νέα χαρακτηριστικά θα χρησιμοποιούνταν μη σχεσιακή βάση δεδομένων, η οποία θα πρόσφερε περισσότερη ευελιξία καθώς πληροί τις αρχές BASE (Basically Available, Soft State, Eventual Consistency).

Η PostgreSQL είναι μία δωρεάν και open source πλατφόρμα διαχείρισης σχεσιακών βάσεων δεδομένων. Αρχικά ονομαζόταν POSTGRES, αναφερόμενο στην προέλευσή του ως διάδοχος της βάσης δεδομένων Ingres που αναπτύχθηκε στο Πανεπιστήμιο της Καλιφόρνιας στο Μπέρκλεϊ. Το 1996, το έργο μετονομάστηκε σε PostgreSQL για να αντικατοπτρίζει την υποστήριξή του για SQL. Μετά από μια αναθεώρηση το 2007, η ομάδα ανάπτυξης αποφάσισε να διατηρήσει το όνομα PostgreSQL. Η επιλογή της Postgres προέκυψε ύστερα σύγκριση με την MySQL. Το γεγονός ότι η Postgres είναι δωρεάν προσφέροντας όλα τα χαρακτηριστικά της στον χρήστη, σε αντίθεση με την MySQL η οποία παρέχει δωρεάν μόνο το community edition, οδήγησε στην επιλογή της. Επιπλέον, ύστερα από μία έρευνα που είχε λάβει χώρα το 2015 από τον Matloob Khushi, ερευνητή του πανεπιστημίου του Σίδνεϊ, προκύπτει ότι η Postgres έχει καλύτερη απόδοση σε σχέση με την MySQL.

3.3 Java

Για τον σχεδιασμό του API χρησιμοποιείται η υψηλού επιπέδου αντικειμενοστρεφής γλώσσα προγραμματισμού Java. Κύριο γνώρισμά της αποτελεί η μεταφερσιμότητα, το γεγονός δηλαδή του να μπορεί να εκτελεστεί ένα πρόγραμμα σε πολλές μηχανές ανεξάρτητα από το λειτουργικό σύστημα, και οφείλεται στο JVM, ένα από τα δομικά μέρη του Java Development Kit (JDK) και συγκεκριμένα του JRE (Java Runtime Environment). Η επιλογή της έγινε βάσει ήδη υπάρχουσας εξοικείωσης οδηγώντας έτσι σε ταχύτερους ρυθμούς ανάπτυξης της εφαρμογής, εφόσον έτσι δεν αφιερώθηκε χρόνος στην εκμάθηση και εξοικείωση με άλλη γλώσσα.



Εικόνα 3.1 Java Development Kit

3.4 Gradle

Μία από τις επιλογές κατά την δημιουργία του API είναι η επιλογή του build tool που θα χρησιμοποιηθεί. Το συγκεκριμένο εργαλείο είναι υπεύθυνο για την διαχείριση των dependecies των πακέτων του compile και της διεξαγωγής αυτοματοποιημένων test. Τα πλεονεκτήματα που παρέχει είναι η βελτίωση της ποιότητας του τελικού build, η απαλοιφή περιττών task, η παροχή δυνατότητας για Continuous Integration και Continuous Testing.

Οι επιλογές που ήταν διαθέσιμες ήταν αυτές μεταξύ του Gradle και του Apache Maven. Ωστόσο συγκρίνοντας τα δύο εργαλεία μεταξύ τους προκύπτει το εξής συμπέρασμα. Το Gradle, παρόλο που έχει μεγαλύτερη καμπύλη μάθησης, είναι πιο ευέλικτο όσο αναφορά τις γλώσσες προγραμματισμού που υποστηρίζει, σε αντίθεση με το Maven που χρησιμοποιείται αποκλειστικά για Java εφαρμογές και επίσης είναι πολύ πιο γρήγορο . Επομένως επιλέχτηκε ως build tool το Gradle.

3.5 Google Maps API

Προκειμένου να αντληθούν οι πληροφορίες που αφορούν τα σημεία ενδιαφέροντος επιλέχτηκε το Google Maps API. Ο κύριος λόγος επιλογής ήταν το γεγονός ότι παρείχε εκτεταμένο documentation για την ορθή χρήση του καθώς και πολλά παραδείγματα για την κάθε βιβλιοθήκη αντίστοιχα. Συγκεκριμένα, γίνεται χρήση της βιβλιοθήκης GooglePlaces.

3.6 Επίλογος

Σε αυτό το κεφάλαιο περιγράφηκε η έννοια του API καθώς και η επιλογή συγκεκριμένων τεχνολογιών τις οποίες έπρεπε να παρθούν, όπως ήταν η φύση του API, η γλώσσα που χρησιμοποιήθηκε για την ανάπτυξη του καθώς και το build tool της. Αναφέρθηκε επίσης το είδος και η επιλογή της βάσης δεδομένων.

Κεφάλαιο 4ο: Eclipse Vert.x

4.1 Εισαγωγή

Το Vert.x αποτελεί ένα εργαλείο ανάπτυξης λογισμικού από την Eclipse. Είναι open source, polyglot καθώς υποστηρίζει γλώσσες JVM και μη, όπως για παράδειγμα Ruby, Python, Javascript, Groovy και Java. Αποτελεί εργαλείο αντιδραστικού προγραμματισμού (reactive), δηλαδή σχετίζεται με ασύγχρονα ρεύματα και αλλαγές που προκύπτουν από τυχόν συμβάντα.

Το Vert.x δεν αποτελεί framework αλλά εργαλείο. Αυτό σημαίνει ότι κατά την ανάπτυξη μίας εφαρμογής ο προγραμματιστής μπορεί να χρησιμοποιήσει μόνο τις βιβλιοθήκες και τα modules που είναι απαραίτητα. Το γεγονός αυτό το καθιστά ένα σύνθετο αλλά εύκολα ενσωματώσιμο εργαλείο που είναι παράλληλα οικονομικό από άποψη πόρων συγκριτικά με την χρήση κάποιου framework .

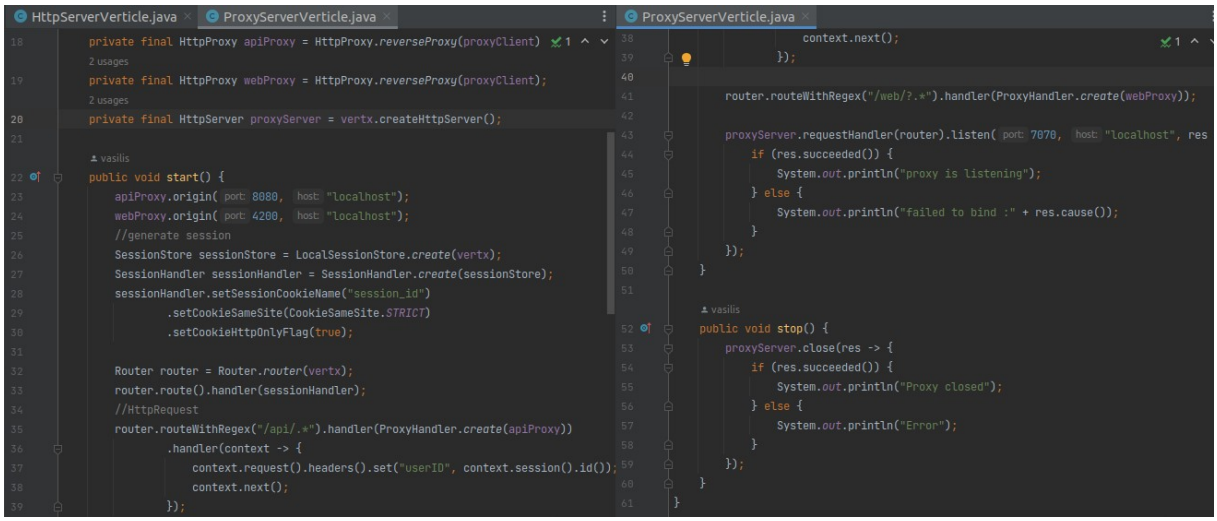
4.2 Verticle

Κύριο χαρακτηριστικό του Vert.x αποτελεί το Verticle, το οποίο είναι ένα κομμάτι κώδικα που τρέχει μέσω του Vert.x instance, το σημαντικότερο αντικείμενο μέσω του οποίου δημιουργούνται τα verticles και εκτελούνται. Είναι βασισμένο στο σχεδιαστικό πρότυπο Actor, κατά το οποίο ένας Actor βάσει του μηνύματος που θα λάβει μπορεί παράλληλα να δημιουργήσει και να στείλει μήνυμα σε άλλους Actors, να δημιουργήσει άλλους Actors, να ορίσει την ενέργεια που θα πραγματοποιηθεί βάσει του ακόλουθου μηνύματος που θα λάβει. Το συγκεκριμένο πρότυπο δίνει ιδιαίτερη έμφαση στις ασύγχρονες ενέργειες .

Υπάρχουν δύο ειδών Verticle, τα standard verticles και τα worker verticles. Τα worker verticles εκτελούνται κάνοντας χρήση ενός thread από το Vert.x worker thread pool και χρησιμοποιούνται για την εκτέλεση blocking κώδικα. Στα standard verticles, από την άλλη, εκχωρείται ένα event loop thread όταν δημιουργούνται και όταν καλείται η μέθοδος start, προκειμένου να ξεκινήσει η εκτέλεση του, καλείται με αυτό το event loop. Έτσι, όταν καλείται μία μέθοδος ή ένας handler, δηλαδή ένας χειριστής για το αντίστοιχο συμβάν, από το αντίστοιχο event loop το Vertx βεβαιώνει ότι όλα θα εκτελεστούν στο ίδιο event loop. Με αυτό τον τρόπο η εφαρμογή λειτουργεί σε ένα thread αποφεύγοντας έτσι τυχόν αδιέξοδα και απλοποιώντας τη διαχείριση πολλών νημάτων. Στην δεδομένη περίπτωση γίνεται χρήση δύο standard verticle του HttpServerVerticle και του ProxyServerVerticle.

Σκοπός του HttpServerVerticle είναι να κάνει expose τα endpoints και μέσω της βιβλιοθήκης Vert.x web να αντιστοιχήσει τις http μεθόδους με το κατάλληλο endpoint, προκειμένου να χειριστεί το κάθε αίτημα κατάλληλα και να επιστραφεί η κατάλληλη απάντηση και ο κατάλληλος κωδικός http status.

Το ProxyServerVerticle από την άλλη είναι ο server που τρέχει η ολοκληρωμένη εφαρμογή και το συνδετικό στοιχείο μεταξύ του http server της Angular, ο οποίος ακούει στη θύρα 4200, και του http server που βρίσκεται στο ομώνυμο verticle και ακούει στη θύρα 8080. Συγκεκριμένα, δρομολογεί τα αιτήματα βάσει του endpoint τους στέλνοντας έτσι τα αιτήματα με κατάληξη /web στον server της Angular και τα αιτήματα με κατάληξη /api στο http server verticle. Με αυτόν τον τρόπο απαλείφεται η ανάγκη χρήσης του cross origin resource sharing (CORS) handler στον http server καθώς τα script του development server της angular δεν θα λειτουργούσαν σε συνδυασμό με αυτόν, εξαιτίας του ότι έχουν διαφορετική θύρα στην οποία ακούνε.



Εικόνα 4.1 Παράδειγμα ProxyServerVerticle

4.3 EventBus

Ένα άλλο σημαντικό χαρακτηριστικό του Vert.x αποτελεί το Event Bus. Αν και δεν ήταν απαραίτητο στην συγκεκριμένη υλοποίηση, το Event Bus είναι υπεύθυνο για την επικοινωνία μεταξύ των verticles μέσω της αποστολής και παραλαβής μηνυμάτων και τον χειρισμό των handler. Τις περισσότερες φορές υπάρχει ένα Event Bus για κάθε Vertx instance που έχουμε.

4.4 Vert.x web

Εκτός από τα χαρακτηριστικά που προαναφέρθηκαν και αποτελούν κύρια δομικά στοιχεία του Vert.x Core, αξίζει να γίνει αναφορά στο Vert.x Web. Το Vert.x Web αποτελεί συλλογή που εμπλουτίζει το Vert.x Core με επιπλέον δυνατότητες που οδηγούν στην ανάπτυξη διαδικτυακών εφαρμογών, παρέχοντας μεγαλύτερη ευκολία μέσω των επιπρόσθετων λειτουργιών.

Το κύριο και πιο ευρέως χρησιμοποιημένο χαρακτηριστικό της συλλογής αυτής αποτελεί η κλάση του Router. Το Router αποτελεί ένα αντικείμενο το οποίο περιέχει μηδέν ή περισσότερα Routes, δηλαδή ένα σετ από κριτήρια που καθορίζουν τον χειρισμό των εισερχόμενων http αιτημάτων. Αρχικά το Router θα διακόψει το εισερχόμενο αίτημα προκειμένου να βεβαιωθεί ότι δεν θα υπάρξουν απώλειες όσο αναφορά το σώμα του αιτήματος ή τυχόν ενημερώσεις του πρωτοκόλλου επικοινωνίας. Ύστερα θα βρει το κατάλληλο route στο οποίο θα σταλεί. Σε κάθε route μπορεί να υπάρχει ένας χειριστής ο οποίος καθορίζει το τι ενέργειες θα γίνουν όσο αναφορά το αίτημα και ανάλογα είτε τερματίζεται είτε μεταφέρεται σε άλλον κατάλληλο χειριστή.

Το είδος του αντικειμένου που δέχεται ο χειριστής ονομάζεται RoutingContext, και περιέχει αναφορές σε ένα HttpRequest και HttpResponse, έννοιες που συναντιούνται και στο Vertx Core, όπως και άλλες έννοιες που διευκολύνουν την ανάπτυξη της εφαρμογής. Για κάθε αίτημα που δρομολογείται υπάρχει μια μοναδική ύπαρξη RoutingContext και η ίδια μεταβιβάζεται σε όλους τους χειριστές του συγκεκριμένου αιτήματος.

4.5 Επίλογος

Το VertX αποτελεί ένα εργαλείο που αναπτύχθηκε από την Eclipse και χρησιμοποιείται για την ανάπτυξη ασύγχρονων και αντιδραστικών εφαρμογών. Είναι πολυγλωσσικό και αποτελεί εύκολα ενσωματώσιμο και οικονομικό από άποψη πόρων. Τα κύρια χαρακτηριστικά του είναι τα Verticles τα

οποία διακρίνονται σε δύο κατηγορίες, τα Standard και τα Worker, όπως επίσης και το EventBus το οποίο χρησιμοποιείται για την επικοινωνία μεταξύ των Verticles. Για την ανάπτυξη διαδικτυακών εφαρμογών χρησιμοποιείται το Vertx Web καθώς η ύπαρξη κλάσεων όπως αυτή του Router κάνει τον χειρισμό των http αιτημάτων πολύ ευκολότερη.

Κεφάλαιο 5ο: RxJava

5.1 Το πρότυπο του Παρατηρητή

Η ReactiveX αποτελεί μία βιβλιοθήκη από συλλογές που επιτρέπουν τη δημιουργία ασύγχρονων εφαρμογών που αντιδρούν στα γεγονότα που προκύπτουν μέσω ακολουθιών. Είναι βασισμένη στο πρότυπο παρατηρητή, ένα από τα τέσσερα πρότυπα της “Συμμορίας των Τεσσάρων” και είναι γνωστό ως Publish-and-Subscribe, και χρησιμοποιείται κυρίως για την υλοποίηση κατανεμημένων συστημάτων χειρισμού συμβάντων, σε λογισμικό "οδηγούμενο από συμβάντα". Κύρια δομικά χαρακτηριστικά αποτελούν το Αντικείμενο (Subject) και ο Παρατηρητής (Observer) μεταξύ των οποίων υπάρχει συσχέτιση ένα προς πολλά. Στόχος του πρότυπου είναι η μεγιστοποίηση της ευελιξίας του συστήματος στον μεγαλύτερο δυνατό βαθμό, πράγμα που επιτυγχάνεται καθώς υπάρχει δυνατότητα προσθήκης ενός παρατηρητή χωρίς να χρειάζεται να γίνουν αλλαγές στο παρακολουθούμενο αντικείμενο.

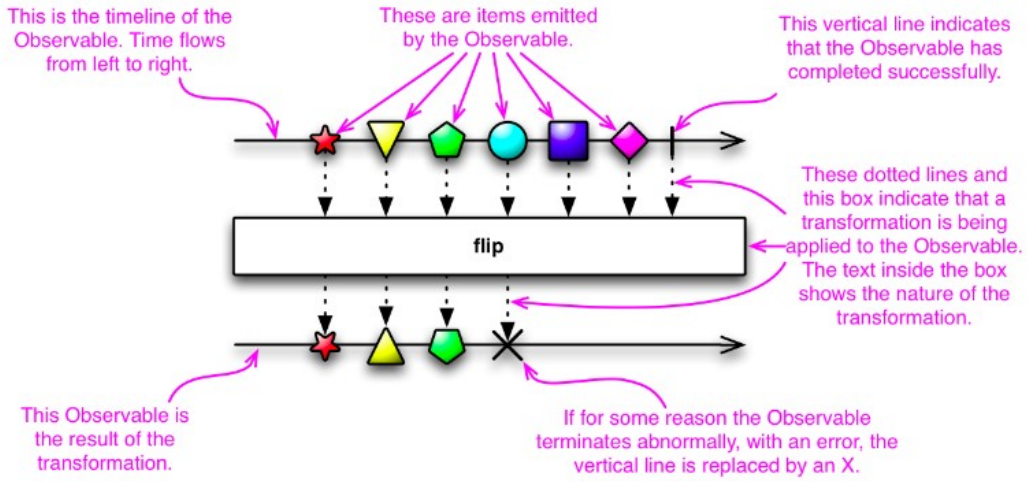
5.2 Κύρια χαρακτηριστικά

Η RxJava αποτελεί μία από τις γλώσσες που υποστηρίζει η βιβλιοθήκη ReactiveX και αφορά τη Java, όπως προκύπτει από την ονομασία. Είναι πολυγλωσσική και υποστηρίζει JVM γλώσσες όπως είναι η Groovy, η Jruby, η Scala, η Clojure και η Kotlin.

Σημαντική έννοια αποτελεί το backpressure. Συγκεκριμένα, κατά την εκτέλεση ασύγχρονων ενεργειών κάθε ενέργεια μπορεί να εκτελεστεί με διαφορετική ταχύτητα οπότε, προκειμένου να αποτραπεί η υπερφόρτωση αυτών των ενεργειών που θα εκδηλωνόταν σαν αυξημένη χρήση μνήμης, λόγω προσωρινής αποθήκευσης ή της ανάγκης για παράλειψη/απόρριψη δεδομένων (memory leak), γίνεται η χρήση του backpressure. Αποτελεί μία μορφή ελέγχου κατά την οποία η κάθε ενέργεια μπορεί να εκφράσει τα βήματα που είναι έτοιμα να εκτελεστούν, οδηγώντας έτσι σε περιορισμό μνήμης των ροών δεδομένων που δεν γνωρίζεται το πλήθος των βημάτων κατά την εκτέλεση μιας ενέργειας. Δηλαδή με την βοήθεια του backpressure μπορεί να ενημερωθεί ο upstream producer κατάλληλα ώστε να μειώσει τον ρυθμό παραγωγής νέων τιμών προκειμένου να προλαβαίνει ο downstream consumer να χειριστεί τις τιμές κατάλληλα.

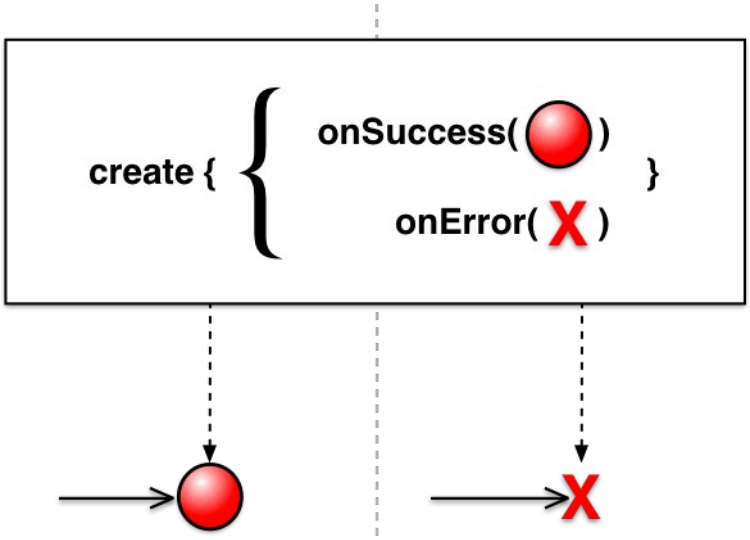
Οι βασικοί τύποι που παρέχονται από την RxJava προκύπτουν βάση της ροής δεδομένων που επιστρέφονται και τους περιορισμούς της. Συγκεκριμένα οι διαθέσιμοι τύποι είναι :

- Flowable : παρέχει 0 ως N ροές, υποστηρίζει backpressure και αντιδραστικά ρεύματα (reactive streams)
- Observable : παρέχει 0 ως N ροές, χωρίς υποστήριξη backpressure



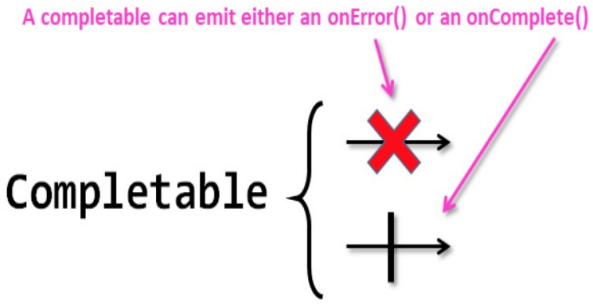
Εικόνα 5.1 Observable

- Single : μία ροή δεδομένων, επιστρέφει ένα αντικείμενο ή σφάλμα



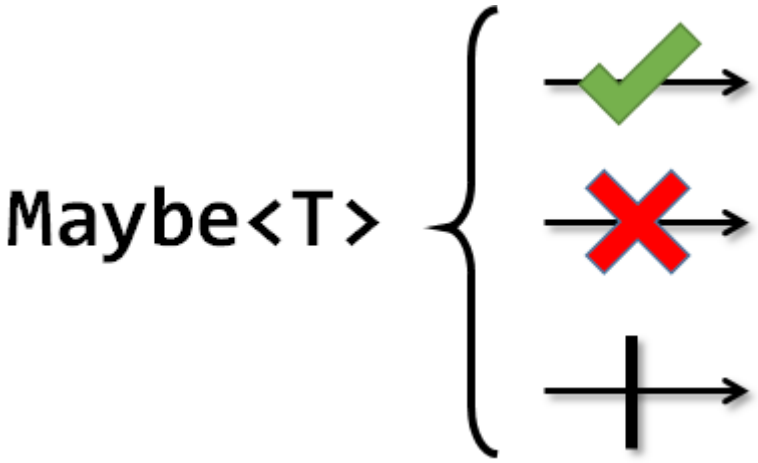
Εικόνα 5.2 Single

- Completable : μία ροή δεδομένων, δεν επιστρέφει αντικείμενο αλλά σηματοδοτεί την ολοκλήρωση ή το σφάλμα συμβάντος



Εικόνα 5.3 Completable

- Maybe : μία ροή δεδομένων, μπορεί να επιστρέψει ένα ή κανένα αντικείμενο και σφάλμα.



Εικόνα 5.4 Maybe

5.3 Upstream / Downstream

Η ροή δεδομένων στην RxJava αποτελείται από μία πηγή, η οποία εκπέμπει τιμές (producer), την οποία ακολουθούν από κανένα έως και περισσότερα ενδιάμεσα βήματα, μέσω συγκεκριμένων τελεστών για την επεξεργασία των τιμών που προκύπτουν και τέλος καταλήγουν σε έναν καταναλωτή δεδομένων (consumer) ή ένα βήμα συνδυασμού, όπου το βήμα είναι υπεύθυνο για την κατανάλωση της ροής δεδομένων με κάποιο μέσο. Η μετάβαση από τον έναν τελεστή στον επόμενο αποκαλείται downstream ενώ αντίθετα η μετάβαση από έναν τελεστή στον προηγούμενο upstream.

```
source
  .operator1()
  .operator2()
  .operator3()
  .subscribe(consumer)
```

Εικόνα 5.5 Upstream - Downstream

5.4 Scheduler

Οι τελεστές στην RxJava δεν συνάδουν με την χρήση τύπων, όπως του Thread ευθέως. Αντιθέτως, προκειμένου να χρησιμοποιηθούν πολλά νήματα (multithreading) κάποια Observables δέχονται σαν παράμετρο στους τελεστές τους Scheduler. Έτσι με αυτό τον τρόπο ο τελεστής εκτελεί ολοκληρωτικά ή τμηματικά τη δουλειά του σε έναν Scheduler. Οι Schedulers της RxJava είναι διαθέσιμοι σε όλες της JVM πλατφόρμες, ωστόσο μερικές όπως αυτή του Android έχει τους δικούς της τυπικούς Schedulers.

5.5 Επίλογος

Η ReactiveX είναι μία συλλογή από βιβλιοθήκες που εξυπηρετούν στην ανάπτυξη αντιδραστικών εφαρμογών και είναι βασισμένη στο πρότυπο του παρατηρητή. Κοινά χαρακτηριστικά μεταξύ των βιβλιοθηκών αποτελούν οι έννοιες των upstream και downstream που περιγράφουν τη σειρά των τελεστών, και η έννοια των Schedulers που χρησιμοποιούνται για παράλληλο προγραμματισμό με χρήση πολλών νημάτων. Η RxJava αποτελεί τμήμα της και χρησιμοποιείται για εφαρμογές που χρησιμοποιούν το JVM. Κύριοι τύποι αποτελούν τα Observables, Flowable, Completable, Single και

Maybe. Ο τύπος ενός Observable καθορίζεται από τον αριθμό παραγόμενων τιμών και την ύπαρξη ή όχι του backpressure, ενός μηχανισμού που ρυθμίζει την συχνότητα παραγωγής τιμών προκειμένου να προλάβει ο ακόλουθος τελεστής να τις χειριστεί σωστά.

Κεφάλαιο 6ο: Domain Driven Design

6.1 Εισαγωγή

Η αρχιτεκτονική που χρησιμοποιήθηκε για την ανάπτυξη του API είναι αυτή του Domain Driven Design. Ο όρος του Domain Driven Design προέκυψε αρχικά από τον Eric Evans με την έκδοση του βιβλίου του, “Domain Driven Design : Tackling Complexity in the Heart of Software”. Η συγκεκριμένη αρχιτεκτονική έχει ως στόχο τη διάσπαση του κώδικα, και αντίστοιχα την ανάπτυξή του, σε domain και infrastructure.

Με τον όρο domain στο συγκεκριμένο πρότυπο εννοείται το κομμάτι κώδικα που αφορά την επιχειρησιακή λογική, δηλαδή το τι κάνει η εφαρμογή ενώ αντίστοιχα με την έννοια του infrastructure (υποδομή) αφορά τον τρόπο υλοποίησης. Το domain δεν θα πρέπει να γνωρίζει λεπτομέρειες που αφορούν τον τρόπο υλοποίησης, συνεπώς θα πρέπει να παραμένει άθικτο σε τυχόν αλλαγές που αφορούν το κομμάτι του infrastructure.

6.2 Δομικά στοιχεία αρχιτεκτονικού προτύπου

Ένα από τα πακέτα που περιέχεται στο domain είναι το πακέτο Model. Σε αυτό το πακέτο περιέχονται τα Repositories, τα Entities και τα Aggregates.

Entities αποτελούν οι κλάσεις οι οποίες περιέχουν έναν συνδυασμό από δεδομένα και συμπεριφορά, όπως για παράδειγμα η κλάση User. Έχουν ταυτότητα αλλά αντιπροσωπεύουν δεδομένα με συμπεριφορά.

Υπάρχουν αντικείμενα τα οποία έχουν χαρακτηριστικά, ωστόσο δεν γίνεται να υπάρχουν από μόνα τους. Προκειμένου να δομηθούν και να γίνει ορθή διαχείρισή τους χρησιμοποιείται το Aggregate. Με τον όρο Aggregates εννοείται μία συλλογή από οντότητες και τιμές που σχετίζονται με την τήρηση των επιχειρησιακών κανόνων. Η διαχείρισή τους γίνεται μέσω ενός aggregate root μέσω του οποίου γίνεται αναφορά από αντικείμενα εκτός του aggregate. Τέτοιο παράδειγμα αποτελεί η κλάση Account και η κλάση Attraction.

Τα Repositories αποτελούν interfaces και χρησιμοποιούνται για τον χειρισμό των aggregate κατά την διάρκεια του κύκλου ζωής του και την διατήρησή του. Παράδειγμα ενός repository είναι το UserRepository και AttractionRepository.

Στο πακέτο Application περιέχονται με τη σειρά τους τα πακέτα Queries και Commands καθώς και οι κλάσεις των application services.

Με τον όρο command εννοείται η κλάση η οποία περιγράφει μία εντολή που θα φέρει αλλαγές στην κατάσταση της εφαρμογής και συνήθως θα επηρεάσει τη βάση δεδομένων είτε με κάποια εγγραφή σε αυτήν ή διαγραφή ενός στοιχείου από αυτήν. Παράδειγμα command είναι οι κλάσεις CreateUserCommand και AddAttractionCommand.

Τα queries αντίθετα αφορούν τις κλάσεις οι οποίες περιγράφουν παραμέτρους για τα prepared statements, με των οποίων την εκτέλεση δεν υπάρχει αλλοίωση στη βάση δεδομένων και απλά λαμβάνουμε πληροφορίες από αυτήν. Κλάσεις που ανήκουν σε αυτό το πακέτο είναι οι κλάσεις GetUserByIdQuery και GetSavedAttractionsQuery.

Τα application services, από την άλλη, είναι υπεύθυνα για την εκτέλεση των queries και commands. Αναλυτικότερα, υπάρχει η δυνατότητα να συνδυαστούν commands και queries ασύγχρονα

προκειμένου να μην χρειαστεί να γίνουν επιπλέον προσπελάσεις στην βάση δεδομένων ή να προσθέτουν επιπλέον κανόνες που θα οδηγήσουν σε μείωση της αποδοτικότητας. Αξίζει να σημειωθεί ότι παρόλο που βρίσκονται στο πακέτο domain δεν έχουν γνώση από τους επιχειρησιακούς κανόνες καθώς αυτό το κομμάτι το αναλαμβάνεται από τα Aggregate. Κάποια από τα services που υπάρχουν στο API είναι τα AttractionApplicationService και το UserApplicationService.

Το πακέτο Infrastructure περιέχει τις κλάσεις που προσδιορίζουν τον τρόπο υλοποίησης των επιχειρησιακών κανόνων. Εδώ περιέχονται οι κλάσεις των HttpServerVerticle και HttpProxyVerticle εφόσον ο τρόπος υλοποίησης έγινε μέσω του Eclipse Vertx, αλλά και κλάσεις όπως η PostgressAttractionRepository κάνοντας έτσι αντιληπτή την ανάπτυξη του αντίστοιχου Repository μέσω της χρήσης PostgreSQL.

Στην κλάση main, προκειμένου να χρησιμοποιηθούν όλα τα προαναφερθέντα στοιχεία, γίνεται αρχικοποίηση των Repositories και των application services βάσει αυτών και ύστερα ενός Vertx instance προκειμένου να γίνει deploy των verticle.

Στην περίπτωση τυχόν αλλαγής, όπως για παράδειγμα αλλαγή στη βάση δεδομένων από PostgreSQL σε MongoDB, θα έπρεπε να υλοποιηθούν οι αντίστοιχες κλάσεις και ύστερα να αλλάξει η αρχικοποίηση των Repositories στην main. Έτσι το κομμάτι του domain θα έμενε ανέπαφο μειώνοντας τις πιθανότητες να προκύψει κάποιο σφάλμα ή η ανάγκη για refactor.

6.3 Επίλογος

Σε αυτό το κεφάλαιο περιγράφτηκε το back end σχολιάζοντας την αρχιτεκτονική Domain Driven Design καθώς και τα πλεονεκτήματα που προσφέρει εξηγώντας σημαντικές έννοιες, όπως τα Aggregates, τα Application Services, το domain και το infrastructure.

Κεφάλαιο 7ο: Angular

7.1 Εισαγωγή

Οι δυνατότητες που προσφέρουν οι μοντέρνες διαδικτυακές εφαρμογές , μέσω της διεπαφής του χρήστη, καθιστούν ιδιαίτερα πολύπλοκη την δομή τους. Αυτό καθιστά την ανάπτυξη και την συντήρηση του κώδικα της εφαρμογής δύσκολη. Για αποφυγή αυτού του φαινομένου η ανάπτυξη των σημερινών εφαρμογών γίνονται με την χρήση ενός framework. Με αυτό τον τρόπο ο κώδικας που προκύπτει είναι πιο εύκολα συντηρήσιμος έχοντας καλύτερη δομή. Το framework που επιλέχθηκε στην προκειμένη περίπτωση ήταν αυτό της Angular.

Η Angular αποτελεί ένα ολοκληρωμένο web application framework βασισμένο στην Typescript. Όπως τα αντίστοιχα front end framework ReactJs και VueJs, έτσι και η Angular αποτελεί ένα component based framewok. Αναπτύχθηκε από την όμαδα της Google το 2016, αποτελώντας εξέλιξη της AngularJS.

7.2 Components

Με την έννοια component περιγράφεται ένα κομμάτι κώδικα που περιγράφει ένα τμήμα της εφαρμογής όντας σχεδιασμένο έτσι ώστε να μπορεί να επαναχρησιμοποιηθεί και να εκτελεί ένα σκοπό όπως για παράδειγμα ένα navigation bar.

Τα components της Angular αποτελούνται από ένα HTML template που καθορίζει το τι γίνεται render στην οθόνη του χρήστη, ένα αρχείο TypeScript που καθορίζει την συμπεριφορά του component και ένα CSS selector που καθορίζει τη μορφοποίηση του template.

Ο κύκλος ζωής του component ξεκινάει με την αρχικοποίηση της κλάσης του όταν γίνεται render στην οθόνη μαζί με τα παιδιά του (εμφωλευμένα components). Συνεχίζεται με τυχόν αλλαγές που μπορεί να προκύψουν (change detection) καθώς γίνεται έλεγχος για τις τιμές που αλλάζουν και ενημερώνει αντίστοιχα τα components και το τι βλέπει ο χρήστης. Τελειώνει με την διαγραφή του instance του όταν σταματά να γίνεται rendered. Σημαντικά εργαλεία που βοηθάνε στην διαχείριση του component κατά τον κύκλο ζωής του αποτελούν τα lifecycle hooks όπως για παράδειγμα το onInit που αφορά την αρχικοποίηση του component και το onDestroy που αφορά την καταστροφή του.

7.3 Templates

Τα templates στην Angular ενισχύονται με επιπλέον δυνατότητες. Ένα παράδειγμα αποτελεί το text interpolation. Με τον όρο αυτό εννοείται η προσθήκη εκφράσεων και μεταβλητών σε HTML με τη χρήση διπλών άγκιστρων {{ }}. Παρέχει λειτουργικότητα και ευελιξία καθώς μπορεί να αλλάζει δυναμικά αλλάζοντας το τι βλέπει ο χρήστης.

Ένα άλλον παράδειγμα αποτελεί το binding. Η έννοια αυτή αφορά την δημιουργία μία συσχέτισης μεταξύ του UI που δημιουργήθηκε από το template, όπως ένα DOM element, directive ή component, και του μοντέλου, δηλαδή ενός component instance το οποίο ανήκει στο αντίστοιχο template. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας αγκύλες και μέσα σε αυτές να γίνει εισαγωγή του επιθυμητού χαρακτηριστικού, πραγματοποιώντας έτσι δυναμική δήλωση του style και της κλάσης ενός αντικειμένου πετυχαίνοντας έτσι attribute binding.

Το άλλο είδος binding είναι το event binding. Αντίθετα από το attribute binding στο event binding χρησιμοποιούνται παρενθέσεις μέσα στις οποίες εισάγεται το είδος του event που γίνεται έλεγχος. Με αυτό τον τρόπο εισάγεται ένα ή παραπάνω event handler που η λειτουργία του είναι όμοια με αυτή της μεθόδου addEventListener.

Τέλος, υπάρχει η δυνατότητα συνδυασμού των δύο προηγούμενων τύπων binding και λέγεται η two-way binding. Ο συνδυασμός αυτός είναι ιδιαίτερα χρήσιμος στην περίπτωση που υπάρχει επικοινωνία μεταξύ ενός parent component και του child-component του. Αναλυτικότερα, σε αυτό το σενάριο κάνοντας χρήση τα decorators @Input και @Output, η λειτουργία των οποίων αφορά εισαγωγή δεδομένων και εξαγωγή δεδομένων αντίστοιχα, σε συνδυασμό με τις παρενθέσεις και τις αγκύλες μπορούν να ανταλλάσσουν δεδομένα δύο components. Στην ουσία είναι ένας πιο σύντομος τρόπος γραφής καθώς δεν χρειάζεται να γίνει αναφορά στο event όπως φαίνεται στο παράδειγμα που ακολουθεί.

7.4 Directives

Τα Directives αποτελούν κλάσεις οι οποίες παρέχουν ορισμένες λειτουργίες στα στοιχεία της Angular. Το πιο συνηθισμένο είδος Directive αποτελεί το component, το οποίο περιγράφτηκε προηγουμένως και είναι το μόνο directive που έχει template.

Υπάρχουν τα attribute directives τα οποία αλλάζουν την συμπεριφορά ή την όψη ενός άλλου directive, component ή διαφορετικού στοιχείου της Angular. Τέτοιου είδους directive χρησιμοποιείται στην εφαρμογή για να χειριστούμε το style του button ανάλογα με το εάν ένα σημείο ενδιαφέροντος είναι αποθηκευμένο ή όχι.

Τέλος υπάρχουν τα structural directives τα οποία μορφοποιούν το DOM προσθέτοντας ή αφαιρώντας στοιχεία. Χαρακτηριστικά structural directives αποτελούν τα ngIf που βοηθούν στο conditional rendering ανάλογα με το αν ισχύει ή όχι η συνθήκη που ελέγχεται και το ngFor το οποίο επαναλαμβάνει ενέργειες για το κάθε στοιχείο ενός συνόλου.

7.5 Dependency Injection

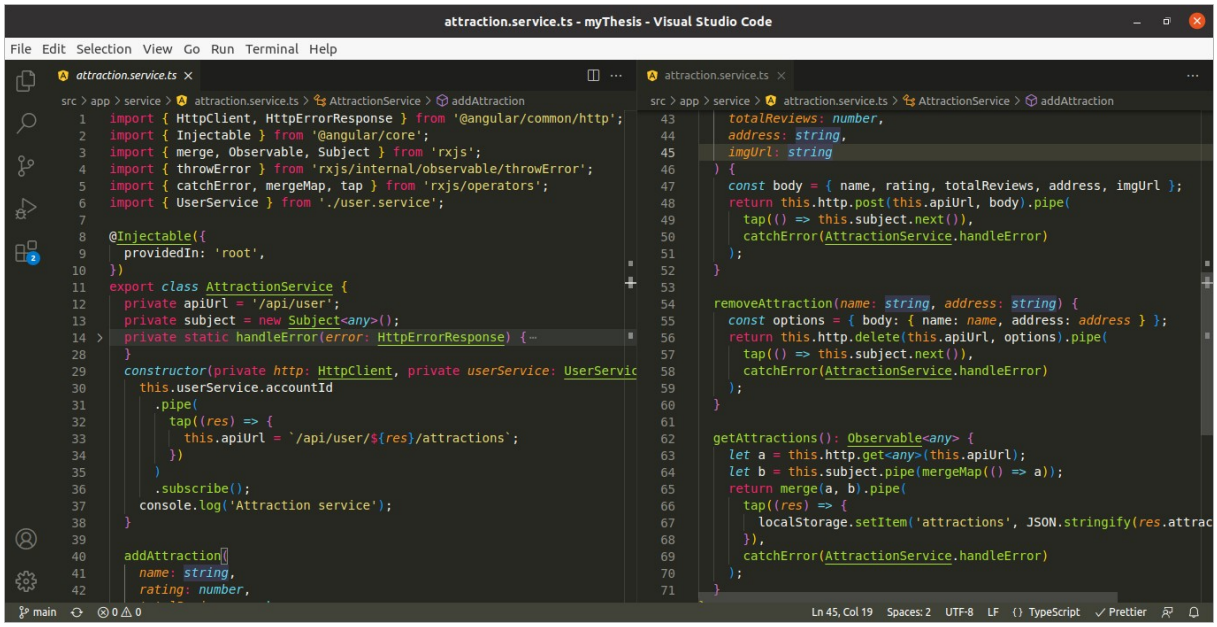
Το Dependency Injection (DI) αναφέρεται σε έναν μηχανισμό της Angular με τον οποίο δίνεται η δυνατότητα να εισάγουμε κομμάτια της εφαρμογής σε άλλα αντίστοιχα που τα χρειάζονται βελτιώνοντας την ευελιξία και τη δομή της εφαρμογής.

Συγκεκριμένα, υπάρχουν δύο ρόλοι στο σύστημα του DI, ο dependency comsumer και ο dependency producer. Η σχέση τους καθορίζεται με τον Injector ο οποίος αποτελεί ένα αντικείμενο που βρίσκει στην cache τα ζητούμενα dependecies ή να τα δημιουργήσει με την βοήθεια του provider, ενός εργαλείου που περιγράφει πως θα γίνει η λήψη του DI σχετικά με ένα DI token. Παραδείγματος χάρη, όταν ζητείται ένα dependency ο injector ελέγχει στις καταχωρήσεις του εάν υπάρχει ήδη κάποιο instance και εάν δεν υπάρχει δημιουργείται και αποθηκεύεται. Η Angular δημιουργεί έναν injector για όλη την εφαρμογή και ονομάζεται root injector καθώς και άλλους injectors που χρειάζονται.

```
11  @Component({
12    selector: 'app-home',
13    templateUrl: './home.component.html',
14    styleUrls: ['./home.component.scss'],
15    providers: [{ provide: HomePresenter, useClass: HomePresenter }],
16  })
```

Εικόνα 7.1 Παράδειγμα provider

Το πιο χαρακτηριστικό injectable στην angular αποτελεί το service. Με αυτήν την έννοια περιγράφεται η κλάση που έχει ένα συγκεκριμένο σκοπό, διαφέρει από την έννοια του component το οποίο είναι υπεύθυνο στην ουσία για το user experience, προβάλλοντας δεδομένα μέσω data binding. Με αυτή την διαφοροποίηση βελτιώνεται η δομή και η ευελιξία της εφαρμογής. Τα components θα πρέπει να λαμβάνουν δεδομένα και να στέλνουν τις ενέργειές του μέσω των services όπως για παράδειγμα η λήψη τιμών από τον server. Υπάρχοντα service στην εφαρμογή για παράδειγα, είναι το UserService που αναλαμβάνει την δημιουργία νέων χρηστών, καθώς και την σύνδεση στον λογαριασμό τους.



Εικόνα 7.2 Παράδειγμα service

7.6 Επίλογος

Η Angular αποτελεί ένα ολοκληρωμένο framework που εστιάζει στην σχεδίαση User Interface και χρησιμοποιείται για front end development. Είναι component based, δηλαδή δίνει έμφαση στην σχεδίαση με την χρήση των component τα οποία αποτελούν κύριο δομικό χαρακτηριστικό της. Σημαντικά στοιχεία αποτελούν τα directives και το dependency injection.

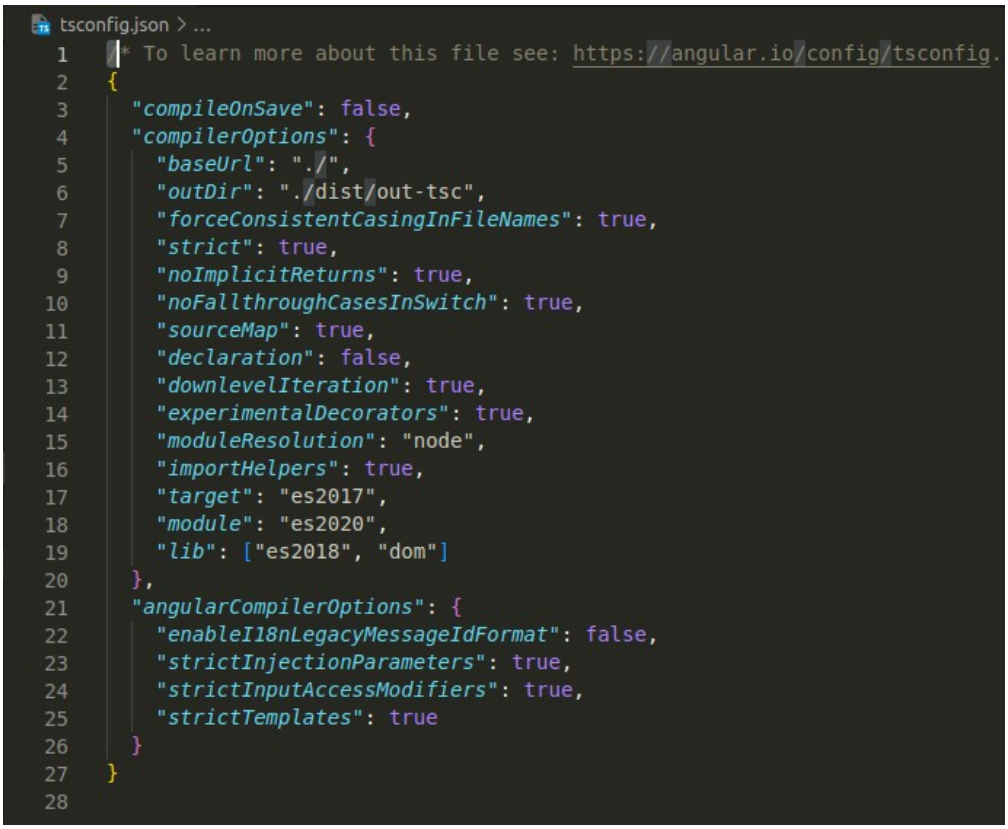
Κεφάλαιο 8ο: TypeScript

8.1 Εισαγωγή

Η TypeScript αποτελεί μία γλώσσα προγραμματισμού, που αναπτύχθηκε από την Microsoft το 2012 και αποτελεί υπερσύνολο της JavaScript. Συγκεκριμένα, σε αντίθεση με την JavaScript η οποία είναι μια loose typed γλώσσα, η TypeScript αποτελεί μία strongly typed γλώσσα προγραμματισμού προσθέτοντας το χαρακτηριστικό static typing. Με αυτή την προσθήκη μειώνονται σημαντικά τα πιθανά λάθη που μπορούν να προκύψουν δίνοντας τη δυνατότητα δήλωσης μεταβλητών ως number, string, void και ούτω κάθε εξής. Αυτό επιτυγχάνεται με το γεγονός ότι δηλώνοντας τύπους για πιθανές συναρτήσεις και μεταβλητές τους έχουμε καλύτερη σαφήνεια στον κώδικα αποφεύγοντας σενάρια που προκύπτει σφάλμα από type error. Επίσης ο κώδικας μεταφράζεται (transpiled) σε JavaScript και πριν γίνει αυτή η μετάβαση γίνεται έλεγχος για το εάν υπάρχει κάποιο σφάλμα με τους τύπους που έχουν δηλωθεί, αποφεύγοντας έτσι τα σφάλματα που προκύπτουν κατά την εκτέλεση (Runtime errors). Με transpile του κώδικα δίνεται η δυνατότητα να προκύψει κάποια παλαιότερη έκδοση της JavaScript, συγκεκριμένα από την έκδοση ES3 έως την πιο πρόσφατη.

8.2 Χρήση της TypeScript στην εφαρμογή

Όπως προαναφέρθηκε, η TypeScript είναι η γλώσσα που χρησιμοποιείται από την Angular για να αναπτυχθεί η διεπαφή του χρήστη. Κατά την δημιουργία της εφαρμογής δημιουργείται αυτόματα ένα αρχείο με το όνομα tsconfig.json. Σε αυτό το αρχείο υπάρχει ένα σετ από κανόνες που αφορούν τις ενέργειες που θα εκτελεστούν κατά την διάρκεια του compile, όπως το implicit type any το οποίο παίρνει τιμές true ή false και περιγράφει την καταχώρηση μιας μεταβλητής σε any όταν δεν μπορεί να το αντιστοιχίσει με κάποιο άλλο τύπο βάση της χρήσης του.



Εικόνα 8.1 Αρχείο tsconfig.json

Κεφάλαιο 9ο: RxJs

9.1 Εισαγωγή

Όπως και η RxJava έτσι και η RxJs αποτελεί τμήμα της συλλογής ReactiveX ωστόσο αφορά την JavaScipt. Αυτό έχει ως συνέπεια να παρέχεται η ίδια δυνατότητα υλοποίησης ασύγχρονων και event-driven εφαρμογών και από την μεριά του client.

9.2 Observable

Συγκριτικά με την βιβλιοθήκη που αφορά την Java, η συγκεκριμένη έχει ορισμένες διαφορές καθώς απουσιάζουν τύποι όπως είναι το Flowable, το Single, Maybe και το Completable. Στην RxJs η έννοια του Observable περιγράφει μία ροή δεδομένων που παραπέμπει σε μια λίστα που χρησιμοποιεί το Push πρωτόκολλο.

9.2.1 Push – Pull πρωτόκολλο

Στην JavaScript υπάρχουν δύο είδη πρωτόκολλων, τα Push και τα Pull που περιγράφουν πως ένας Consumer θα επικοινωνεί με τον Producer. Στο Pull πρωτόκολλο ο Consumer καθορίζει πότε θα λάβει μία νέα τιμή από τον Producer και ο Producer δεν γνωρίζει πότε θα παραδοθεί η τιμή αυτή στον Consumer. Όλες οι συναρτήσεις στην JavaScript αποτελούν παράδειγμα αυτού του πρωτοκόλλου καθώς η συνάρτηση αποτελεί τον Producer ο οποίος παράγει μία τιμή όταν καλείται κάνοντας pull την τιμή αυτή ο κώδικας την κάνει consume. Στην περίπτωση των Pull πρωτόκολλων ο Producer καθορίζει το πότε θα παράξει μία τιμή (push) ενώ ο Consumer δεν ξέρει πότε θα την λάβει. Αυτή η έννοια περιγράφεται στην JavaScript από τα Promises. Ένα Promise, αποτελώντας τον Producer, παραδίδει μία τιμή σε ένα callback που αποτελεί τον Consumer. Αντίθετα με τις συναρτήσεις το Promise καθορίζει πότε θα στέλνεται η τιμή στα callbacks.

9.2.2 Σύγκριση Observable – Promises

Τόσο τα Promises όσο και τα Observables χρησιμοποιούνται για την διαχείριση ασύγχρονων ενεργειών, ωστόσο παρουσιάζουν πολλές διαφορές. Συγκεκριμένα τα Promises είναι αποκλειστικά ασύγχρονα ενώ τα Observables μπορούν να εκτελέσουν ενέργειες συγχρονισμένα. Τα Observables δεν εκπέμπουν τιμές εκτός εάν κάποιος (ο Observer) κάνει subscribe σε αυτά. Τότε θα μπορεί να λαμβάνει τιμές κάθε φορά που εκπέμπονται από το stream. Αυτό το γεγονός καθιστά τα Observables lazy και ικανά να εκπέμψουν πολλαπλές τιμές σε αντίθεση με τα Promises που επιστρέφουν μόνο μία τιμή και είναι eager, δηλαδή θα εκτελεστούν τη στιγμή που δηλώνονται εκπέμποντας τιμή ανεξάρτητα από το εάν κάποιος θα χειριστεί την τιμή αυτή ή όχι. Τέλος τα observable έχουν τη δυνατότητα να ακυρωθούν και να σταματήσουν να παράγουν τιμές ωστόσο τα Promises δεν μπορούν.

9.3 Subject

Το Subject αποτελεί ένα είδος Observable το οποίο έχει τη δυνατότητα να είναι ταυτόχρονα και Observer. Χαρακτηριστική διαφορά σε σχέση με το Observable αποτελεί το γεγονός ότι το Subject μπορεί να μεταδόσει τις τιμές του σε πολλούς Observers μαζικά ενώ το απλό Observable διατηρεί ξεχωριστό instance για κάθε Observer που έχει κάνει subscribe σε αυτό. Επιπλέον στην ανάπτυξη της

εφαρμογής βοήθησε στην αποφυγή ενός κακού σχεδιαστικού προτύπου κατά το οποίο υπάρχουν εμφωλευμένα Subscriptions το ένα μέσα στο άλλο.

9.3.1 Είδη Subject

Πέρα από τον γενικό τύπο του Subject υπάρχουν επιπλέον τύποι βάσει της τιμής που αποστέλλεται από αυτό. Οι τύποι αυτοί είναι οι εξής:

- BehaviourSubject : Αποθηκεύει την τρέχουσα τιμή που εκπέμπεται και κάθε φορά που προστίθεται ένας νέος Observer λαμβάνει την τιμή αυτή. Απαιτείται αρχικοποίησή του. Στην εφαρμογή χρησιμοποιείται αυτό το είδος προκειμένου να αποκτηθούν πληροφορίες για την κατάσταση του UI.
- ReplaySubject : Απαιτείται αρχικοποίηση με έναν ακέραιο που δείχνει τον αριθμό των τελευταίων τιμών που έχουν εκπεμφθεί. Όταν ένας νέος Observer κάνει Subscribe θα λάβει τις N τελευταίες τιμές που έχουν εκπεμφθεί (όπου N ο αριθμός με τον οποίο αρχικοποιήθηκε το ReplaySubject)
- AsyncSubject : Εκπέμπει την τελευταία τιμή που παράγει το Observable αλλά μόνο μετά την ολοκλήρωσή του.

9.4 Memory leak

Ο κύκλος ζωής ενός Observable ξεκινάει με την δημιουργία του καλώντας το new Observable, ύστερα γίνεται Subscribe από έναν Observer, δημιουργώντας ένα Subscription και μπορεί να ξεκινήσει να λαμβάνει τιμές με τους τελεστές next. Η ζωή του Observable τελειώνει όταν προκύψει κάποιο exception μέσω του τελεστή error ή μέσω του τελεστή complete που σηματοδοτεί το πέρας λήψης τιμών. Κάθε Subscription που γίνεται αποθηκεύεται στη μνήμη και καταλαμβάνει χώρο. Εάν δεν γίνει σωστή διαχείριση των Subscriptions ή συνδυαστεί αυτό το πρόβλημα με κακό προγραμματισμό της εφαρμογή μπορεί να προκληθούν memory leaks. Αυτό έχει ως αποτέλεσμα να μεγαλώνει το μέγεθος που καταλαμβάνουν τα Subscriptions στην μνήμη προκαλώντας επίσης προβλήματα στην απόδοση του συστήματος .

Η RxJs παρέχει ορισμένους operatos που μπορούν να αντιμετωπίσουν το πρόβλημα αυτό, όπως για παράδειγμα οι take, first, last, οι οποίοι σηματοδοτούν το πότε θα σταματήσει να είναι ενεργό το Subscription. Ωστόσο και η Angular παρέχει ορισμένα εργαλεία όπως το async pipe το οποίο αυτοματοποιεί τη διαχείριση των Subscription, και το onDestroy hook στο οποίο μπορεί να γίνει χειροκίνητα unsubscribe κατά την καταστροφή του component.

9.5 Επίλογος

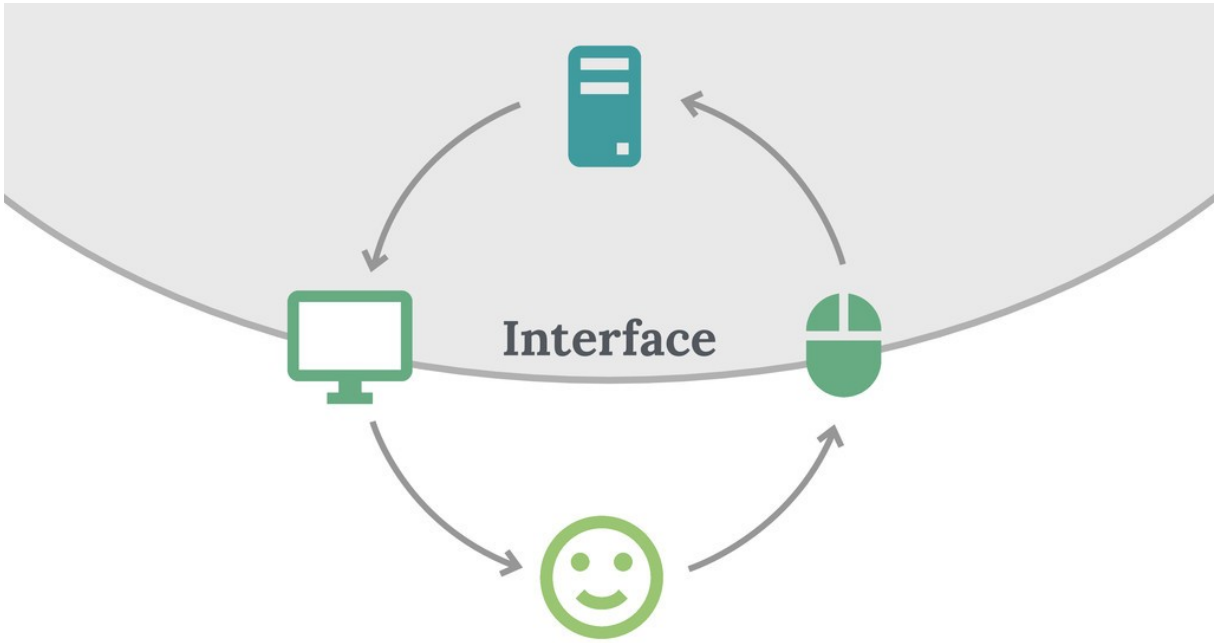
Στο κεφάλαιο αυτό περιγράφηκε η συλλογή της RxJs της βιβλιοθήκης της ReactiveX για την JavaScript. Παρουσιάστηκε η έννοια του Observable και το πως ερμηνεύεται στην JavaScript μέσω σύντομης περιγραφής των πρωτόκολλων push και pull. Επίσης συγκρίνοντας τα Observable με τα Promises προκύπτει το γεγονός ότι τα Observable δίνουν παραπάνω επιλογές και ευελιξία όσο αναφορά τον χειρισμό των δεδομένων. Αναφέρθηκε η έννοια του Subject και το πως η δυνατότητά του να λειτουργεί ως Observable και Observer ταυτόχρονα διαφοροποιείται από το απλό Observable. Επιπρόσθετα εξηγήθηκαν οι διαφορετικοί τύποι Subject. Τέλος έγινε λόγος για τα προβλήματα που μπορούν να προκύψουν στην κατανάλωση πόρων και μνήμης όσο αναφορά τον κακό χειρισμό των Subscriptions .

Κεφάλαιο 10ο: Model – View – Intent

10.1 Γενική περιγραφή

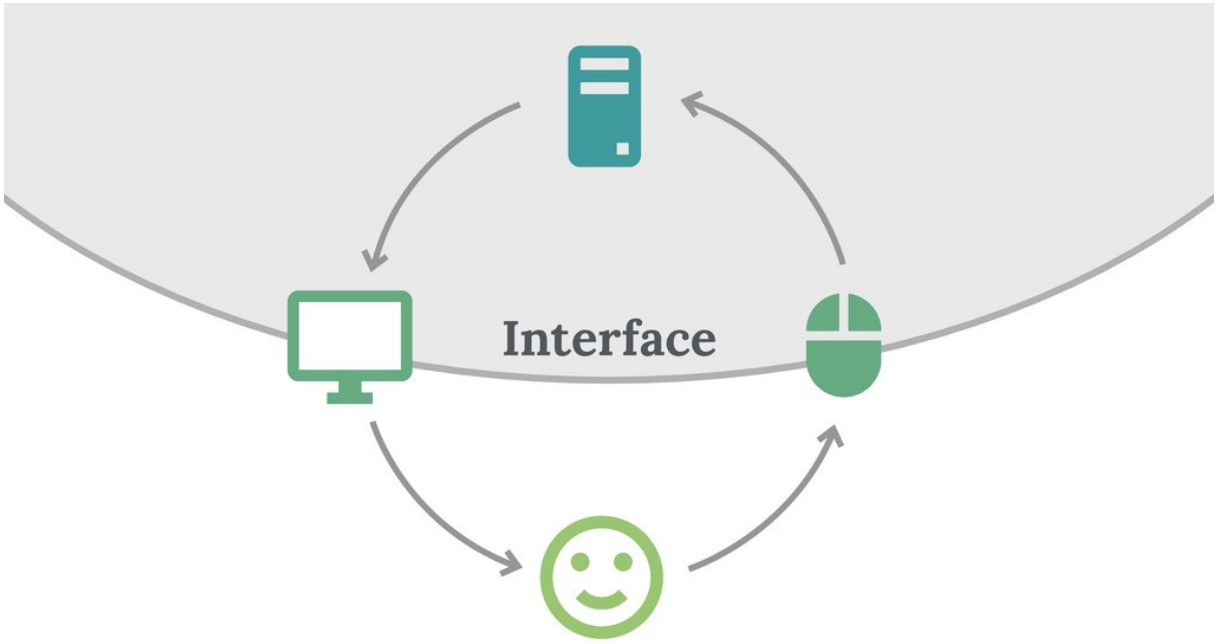
Το αρχιτεκτονικό πρότυπο σχεδίασης που χρησιμοποιήθηκε για την ανάπτυξη της διεπαφής χρήστη είναι το MVI. Το MVI αναφέρθηκε πρώτη φορά το 2015 στο συνέδριο JSConf στη Βουδαπέστη από τον Andre Staltz όταν παρουσίασε το framework που ανέπτυξε ονόματι CycleJs. Στο συνέδριο αυτό ο Staltz εξέφρασε τις εξής παρατηρήσεις

- Η διεπαφή χρήστη μπορεί να εκφραστεί σαν κύκλος



Εικόνα 10.1 Διεπαφή ως κύκλος

- Η διεπαφή χρήστη μπορεί να εκφραστεί ως συνάρτηση

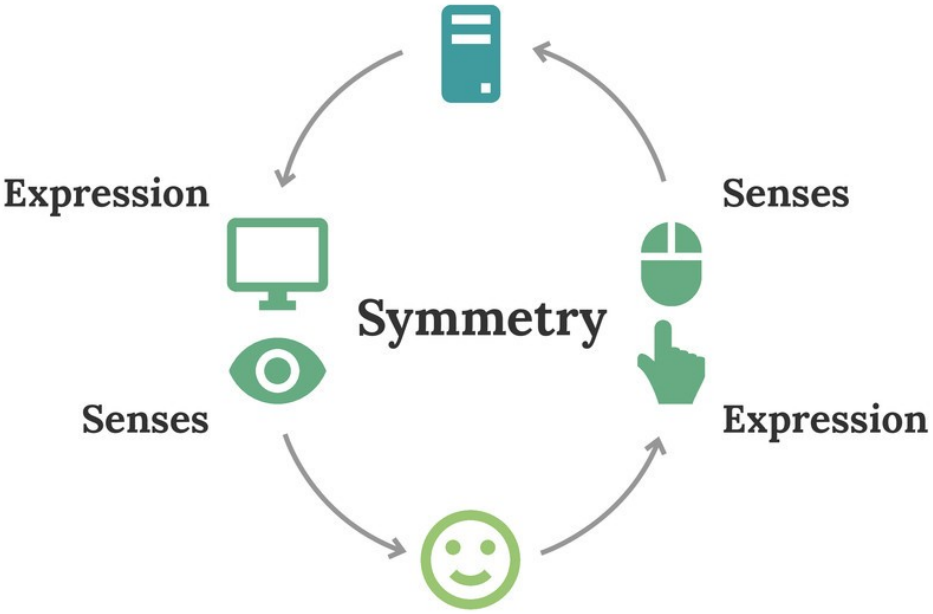


Εικόνα 10.2 Διεπαφή ως συνάρτηση

- Η διεπαφή χρήστη είναι ασύγχρονη

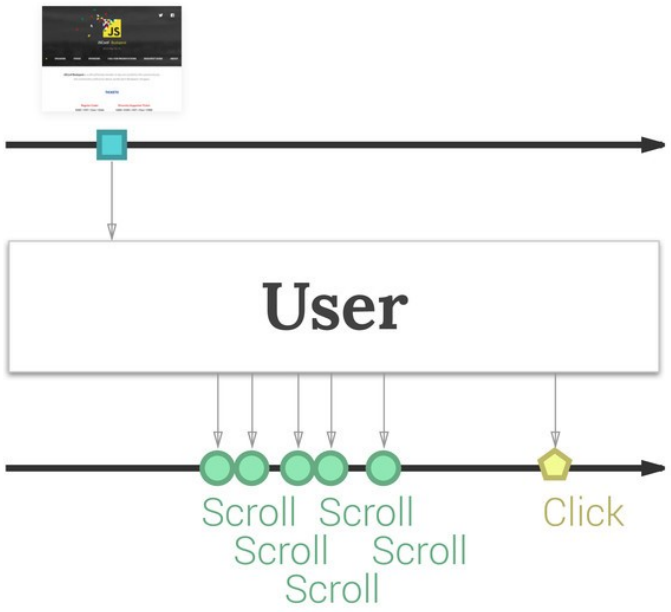
Εικόνα 10.3 Ασύγχρονη διεπαφή

- Η διεπαφή χρήστη είναι συμμετρική



Εικόνα 10.4 Συμμετρική διεπαφή

- Ο χρήστης είναι συνάρτηση



Εικόνα 10.5 Χρήστης ως συνάρτηση

Αναλυτικότερα, εξετάζοντας μία προς μία αυτές τις παρατηρήσεις μπορούμε να δούμε το πως βγήκαν τα συμπεράσματα, Στην πρώτη περίπτωση, αυτό διαπιστώνεται από τον νοητό κύκλο που σχηματίζεται μεταξύ του χρήστη και του υπολογιστή μέσω των συσκευών εισόδου και εξόδου, όπως φαίνεται στην παρακάτω εικόνα.

Επίσης, η διεπαφή χρήστη μπορεί να εκφραστεί ως συνάρτηση εάν θεωρήσουμε ως απόρροια της επεξεργασίας που προέκυψε από την ενέργεια των συσκευών εισόδου και παρουσιάζεται στον χρήστη μέσω των συσκευών εξόδου.

Ταυτόχρονα, θεωρείται και ασύγχρονη εφόσον μπορεί να αλλάζει κατάσταση καθώς αλληλεπιδρά ο χρήστης με αυτήν όπως για παράδειγμα με τη μορφή autocomplete, σε αντίθεση με την περίπτωση των CLI που είναι συνήθως blocking και δεν υπάρχει αλληλεπίδραση μέχρι να εκτελεστεί η εντολή.

Επιπλέον, όπως ο υπολογιστής αποτελείται από συσκευές εισόδου και εξόδου έτσι μπορούμε να κάνουμε την αντίστοιχη αναλογία και για τον άνθρωπο θεωρώντας ως συσκευή εισόδου τα μάτια του για παράδειγμα, ή έστω όποιον άλλο τρόπο χρησιμοποιείται για την αντίληψη και συλλογή πληροφοριών και για συσκευή εισόδου τις εκφράσεις και αλληλεπιδράσεις του. Από αυτό τον συλλογισμό προκύπτει το συμπέρασμα πως η αλληλεπίδραση μεταξύ των δυο παρουσιάζει ομοιότητες καθιστώντας την διεπαφή συμμετρική.

Όσον αναφορά την ύπαρξη του χρήστη ως συνάρτηση, μπορεί να προκύψει το συμπέρασμα, θεωρώντας ως παράμετρο το τι αντιλαμβάνεται ο χρήστης μέσω των αισθήσεών του, όπως παραδείγματος χάρη την όραση και το τι βλέπει στην οθόνη και ως έξοδο τις ενέργειές του όπως είναι κάποιο κλικ ή scroll με το ποντίκι.

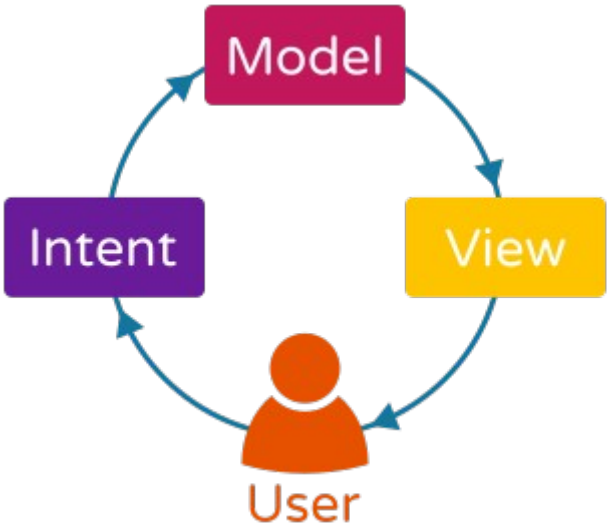
Στο MVI, βάση των παραπάνω παρατηρήσεων, δίνεται ιδιαίτερη έμφαση στην μονόδρομη κυκλική ροή δεδομένων, στην οποία υπάρχουν αμετάβλητα αντικείμενα, και την αντιδραστικότητα.

Αναλύοντας το κάθε στοιχείο του προτύπου προκύπτουν οι εξής παρατηρήσεις. Το Model είναι υπεύθυνο για την επιχειρησιακή λογική και είναι κατά μία έννοια η γέφυρα μεταξύ του backend και του front end καθώς μέσω αυτού μεταφέρονται τα δεδομένα, ωστόσο δεν γνωρίζει τίποτα για το View και συνεπώς για την κατάσταση της εφαρμογής.

Το View είναι υπεύθυνο για το τι βλέπει ο χρήστης στην οθόνη του, δηλαδή την κατάσταση της εφαρμογής (state). Υλοποιείται με τη βοήθεια ενός Presenter ο οποίος δέχεται σαν παράμετρο το Intent και ζωγραφίζει το UI βάσει αυτού δημιουργώντας ένα νέο μοντέλο.

Τα Intent είναι στην ουσία τα γεγονότα που λαμβάνουν χώρα κατά την διάρκεια της αλληλεπίδρασης μεταξύ του χρήστη και την εφαρμογής και αποτελούν παράμετρο των presenters προκειμένου να δημιουργηθεί το νέο μοντέλο βάση του οποίου θα γίνει μία εγγραφή στη βάση δεδομένων ή η νέα δημιουργία της κατάστασης της εφαρμογής.

Η φύση του σχεδιαστικού προτύπου, δηλαδή η μονόδρομη ροή δεδομένων, σε συνδυασμό με αμετάβλητα Models έχει ως αποτέλεσμα σε ευκολότερη συντήρηση και debugging εφόσον ο προγραμματιστής θα πρέπει να έχει στο νου του κατά κύριο λόγο μόνο λογικά λάθη που προκύπτουν κατά την σχεδίαση του View από τον Presenter. Ωστόσο, αρνητικό χαρακτηριστικό σε αυτό το πρότυπο αποτελεί η καμπύλη μάθησης καθώς αποτελεί λίγο μεγαλύτερη σε σχέση με άλλα, καθώς συνηθίζεται να χρησιμοποιείται με επιπλέον απαιτητικές προσθήκες, όπως η χρήση της βιβλιοθήκης RxJs ή συνήθως RxJava. Η χρήση του προτύπου είναι εκτενέστερη στην ανάπτυξη εφαρμογών Android, ή γνώση αντιδραστικού προγραμματισμού και multithreading (παράλληλος προγραμματισμός).



Εικόνα 10.6 Model View Intent

10.2 Χρήση προτύπου στην εφαρμογή

Στην εφαρμογή χρησιμοποιείται το ResultService που λειτουργεί ως Presenter και τον μοιράζονται συγκεκριμένα component των οποίων οι ενέργειες επηρεάζουν το state της εφαρμογής και συνεπώς επηρεάζει το ένα το άλλο όσον αναφορά την κατάστασή τους. Παραδείγματος χάρη, όταν συνδέεται ο χρήστης στο λογαριασμό του πρέπει να ενημερωθεί κατάλληλα το component που θα δείξει τα αποθηκευμένα σημεία ενδιαφέροντος. Ωστόσο, υπάρχουν και components τα οποία έχουν δικό τους αποκλειστικό Presenter όπως είναι το Autocomplete search bar και ο χάρτης καθώς δεν αλληλοεξαρτώνται όσο αναφορά το state τους. Ωστόσο μοιράζονται ένα service που βοηθάει στην υλοποίηση επιχειρησιακής λογικής που αφορά την εύρεση σημείων ενδιαφερόντων και την επίδειξή τους στο χάρτη. Στο Typescript αρχείο κάθε component γίνεται η δημιουργία ενός intent βάσει της επιθυμητής ενέργειας το οποίο αποστέλνεται στον αντίστοιχο Presenter του εκάστοτε component. Το intent περιέχει τις πληροφορίες που θα χρησιμοποιηθούν για τη δημιουργία των νέων Model βάσει των οποίων θα ενημερωθεί το νέο state. Τα παραπάνω υλοποιούνται με την βοήθεια της βιβλιοθήκης RxJs χρησιμοποιώντας Subjects, Observables καθώς και συγκεκριμένους τελεστές προκειμένου να επιτευχθεί ο κατάλληλος χειρισμός των data streams και να προκύψει το κατάλληλο αποτέλεσμα που θα αντικατοπτρίζει το ζητούμενο state με τα σωστά δεδομένα που θα περιέχονται στα Model.

10.3 Επίλογος

Στο κεφάλαιο αυτό περιγράφηκε η αρχιτεκτονική του προτύπου MVI, αναφέρθηκαν οι παρατηρήσεις που οδήγησαν στην δημιουργία του, εξηγήθηκαν τα δομικά του μέρη, δηλαδή το κομμάτι του Model, του View και αντίστοιχα του Intent. Τέλος, παρουσιάστηκε η χρήση του προτύπου στην εφαρμογή αναφέροντας τον τρόπο με τον οποίο λειτουργεί το κάθε δομικό στοιχείο του πρότυπου, εφαρμόζοντάς το σε μία εφαρμογή που είναι υλοποιημένη με την Angular και εξηγώντας την αναλογία των κύριων χαρακτηριστικών του προτύπου MVI και των component της Angular.

Κεφάλαιο 11ο: Διεπαφή χρήστη (User Interface)

11.1 Material Design

Στην εφαρμογή έγινε χρήση του Material Design. Η τεχνολογία αυτή αποτελεί προϊόν της Google και αναπτύχθηκε το 2014 και μπορεί να χαρακτηριστεί ως γλώσσα σχεδιασμού (design language) καθώς παρέχει οδηγίες και συμβουλές για την σχεδίαση διεπαφών. Αντίστοιχες γλώσσες έχουν αναπτυχθεί από την IBM με το Carbon Design και την Microsoft με το Fluent Design. Πέρα από τις οδηγίες που προσφέρει για την ορθή σχεδίαση των διεπαφών παρέχεται και η δυνατότητα χρήσης υλοποιημένων component με ήδη υπάρχουσες λειτουργίες δομημένο έτσι ώστε να χρειαστεί προσθέσει ο καθένας την λειτουργικότητα που χρειάζεται βάσει των αναγκών του. Το Material Design μπορεί να χρησιμοποιηθεί και για την ανάπτυξη mobile εφαρμογών είτε Android για native εφαρμογές είτε με Flutter για cross platform αλλά και για web εφαρμογές. Όσον αναφορά το web κομμάτι τα frameworks, όπως είναι η Angular, χρησιμοποιούν το Material Design μέσω του Angular Material και η React κάνει χρήση του Material UI, μιας βιβλιοθήκης που υλοποιεί το Material Design για τις απαιτήσεις React εφαρμογών.

11.2 Sass

Προκειμένου να δομηθεί η μορφή μιας διαδικτυακής εφαρμογής όσο αναφορά την εμφάνισή της, γίνεται χρήση της CSS (Cascading Style Sheets), ωστόσο η CSS δεν περιέχει κύρια χαρακτηριστικά που παρατηρούνται σε κλασικές προγραμματιστικές δομές, όπως είναι η χρήση μεταβλητών, συναρτήσεων οι οποίες βοηθούν στην επαναχρησιμοποίηση του κώδικα. Αυτό είχε ως αποτέλεσμα την ανάπτυξη προεπεξεργαστικών γλωσσών οι οποίες αποτελούν υπερσύνολο της CSS, όπως για παράδειγμα η σχέση της TypeScript με την JavaScript, που καλούνται να καλύψουν αυτά τα κενά τα οποία εντοπίζονται στην CSS.

Μία από αυτές τις προεπεξεργαστικές γλώσσες είναι η SASS (Syntactically Awesome Style Sheets) που δημιουργήθηκε το 2006 από τον Chris Eppstein και την Natalie Weizenbaum. Αρχικά η σύνταξη της δεν περιείχε άγκιστρα και για κάθε κανόνα γινόταν αλλαγή σειράς κάνοντας χρήση της κατάληξης .sass στα αρχεία. Πλέον έχει υπερισχύσει το νέο συντακτικό το οποίο έχει τις ίδιες δυνατότητες, ωστόσο γίνεται χρήση άγκιστρων και ερωτηματικών (;) στο τέλος του κάθε κανόνα και τα αρχεία έχουν την κατάληξη .scss, το οποίο είναι συντομογραφία του Sassy CSS.

Βασικά χαρακτηριστικά που παρατηρούνται στην γλώσσα αυτή είναι τα ακόλουθα. Πρώτο στοιχείο αποτελούν οι μεταβλητές. Με το σύμβολο $ συμβολίζεται μία μεταβλητή και ακολουθείται είτε από μία συγκεκριμένη τιμή ή από ένα σύνολο τιμών με δομή key – value που μπορεί να χρησιμοποιηθεί σε διάφορα σημεία κάνοντας αναφορά στο όνομα της χωρίς να χρειάζεται η τιμή της. Είναι ιδιαίτερα χρήσιμες όταν πρέπει να γίνει αναφορά σε κάποιο χρώμα που γίνεται αναφορά του σε πολλά διαφορετικά σημεία στην εφαρμογή, όπως είναι για παράδειγμα χρώματα από ένα logo μίας εταιρείας.

Άλλο σημαντικό χαρακτηριστικό που λείπει από την CSS είναι οι εμφωλευμένοι κανόνες. Με την βοήθεια της Sass δίνεται η δυνατότητα εισαγωγής κανόνων για στοιχεία που περιέχονται σε γονικό αντικείμενο οδηγώντας έτσι σε πιο καθαρό και ξεκάθαρο κώδικα. Ωστόσο, καλή πρακτική αποτελεί η αποφυγή πολλών εμφωλευμένων κανόνων καθώς κατά την μετατροπή του scss αρχείου σε css μπορεί

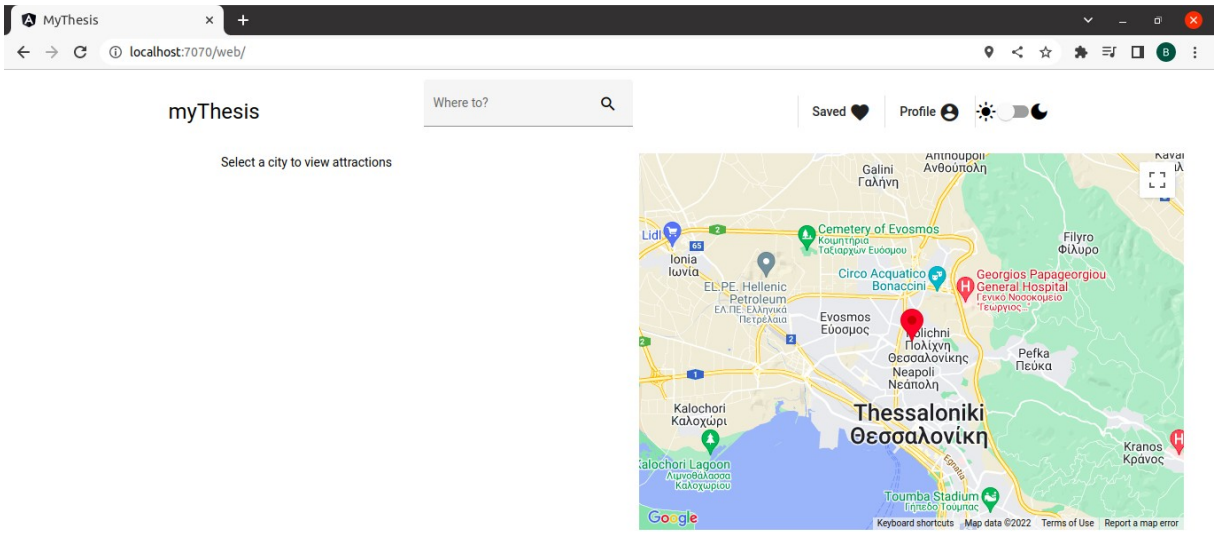
να αυξηθεί η καταναλώσει των πόρων προκειμένου να γίνει render η σελίδα εξαιτίας της έκτασης του css κώδικα.

Ένα από τα κύρια στοιχεία που βοηθάει στην επαναχρησιμοποίηση κώδικα αποτελεί το mixin. Η χρήση του γίνεται μέσω του τελεστή @ ακολουθώντας με το keyword mixin και το όνομά του (@mixin). Για παράδειγμα στην υλοποίηση σκούρου θέματος στην εφαρμογή χρησιμοποιήθηκε ένα mixin το οποίο θα άλλαζε τα χρώματα στην εφαρμογή. Επιπρόσθετα με το @include μπορούμε να χρησιμοποιήσουμε το mixin στα επιθυμητά σημεία.

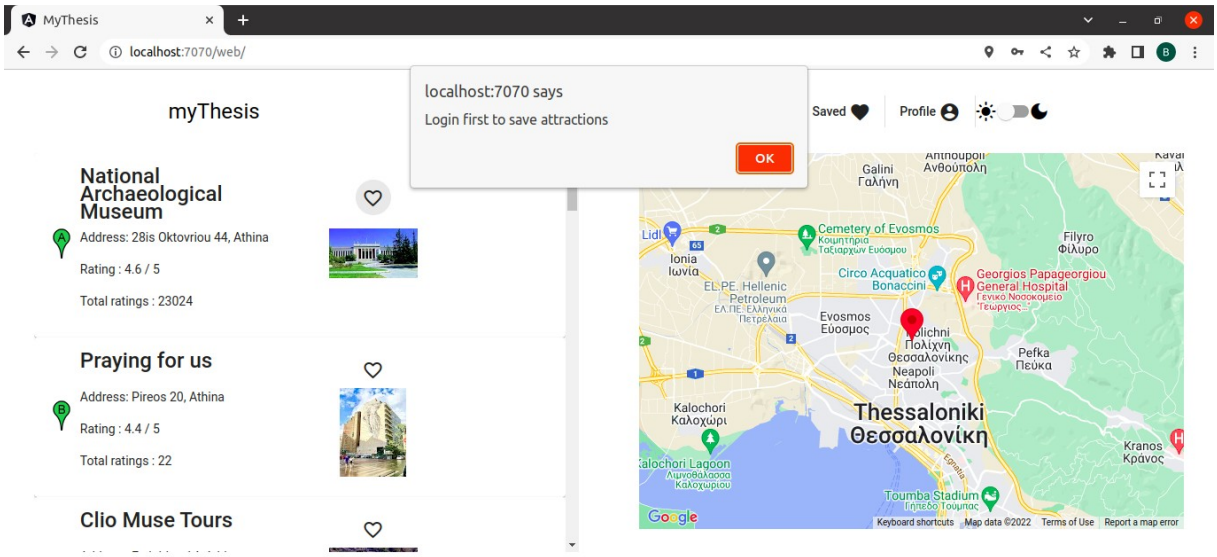
Με την χρήση των εργαλείων που προαναφέρθηκαν συντάχθηκαν και τα media queries. Με αυτόν τον όρο εννοούνται οι κανόνες που εισάγονται σε συγκεκριμένα components προκειμένου να γίνει responsive η εφαρμογή. Δηλαδή να μην αλλοιώνεται η εμφανισή της και να μην υπάρχει απώλεια πληροφοριών όταν γίνεται χρήση της μέσω κινητών ή tablet, εξαιτίας των μικρότερων διαστάσεων που έχουν.

11.3 Προεπισκόπηση εφαρμογής

Αρχικά ο χρήστης επισκέπτοντας την ιστοσελίδα, δεν έχει δημιουργημένο λογαριασμό οπότε ορισμένες λειτουργίες δεν είναι δυνατές. Συγκεκριμένα μπορεί να ψάξει σημεία ενδιαφέροντος από την γραμμή αναζήτησης και να του παρουσιαστούν, ωστόσο δεν είναι δυνατή η αποθήκευσή τους βγάζοντας το αντίστοιχο μήνυμα στην οθόνη του χρήστη.

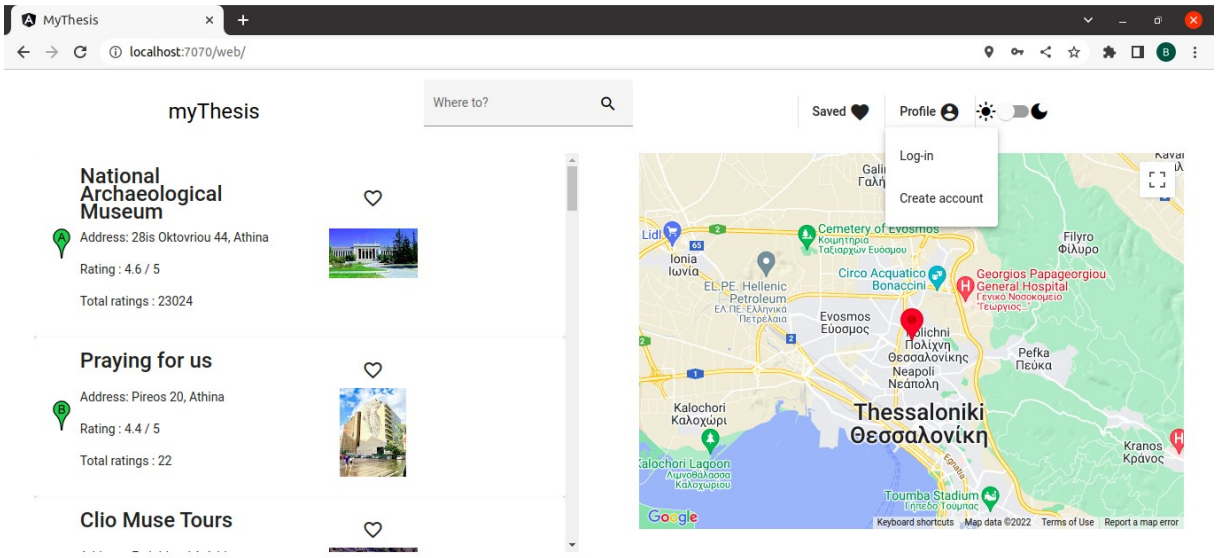


Εικόνα 11.1 Αρχική σελίδα εφαρμογής



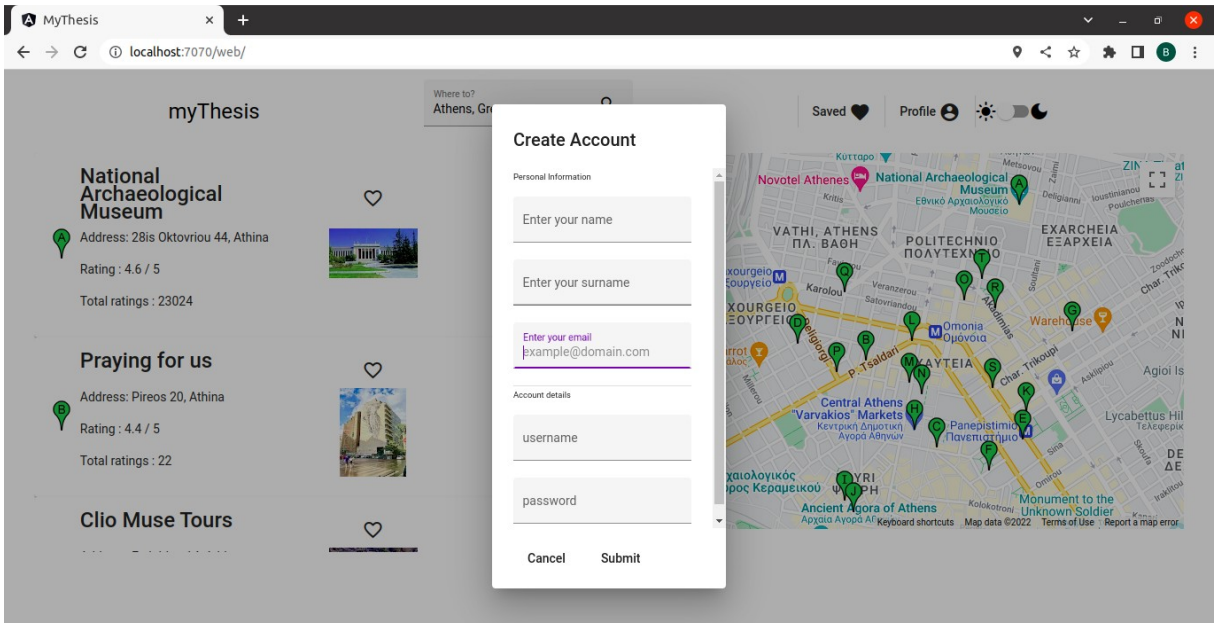
Εικόνα 11.2 Παρότρυνση χρήστη για σύνδεση στον λογαριασμό

Επίσης λόγω απουσίας σύνδεσης σε λογαριασμό οι επιλογές του μενού είναι δομημένες έτσι ώστε να παροτρύνουν τον χρήστη να δημιουργήσει έναν.



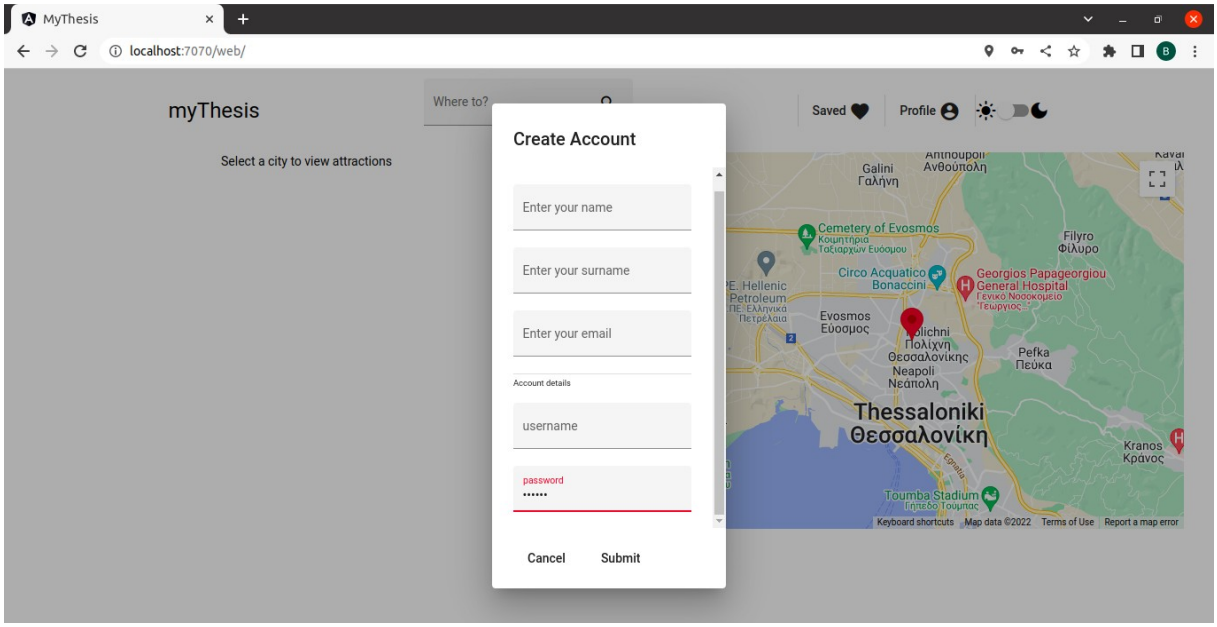
Εικόνα 11.3 Επιλογές χρήστη από το μενού

Στην περίπτωση δημιουργίας λογαριασμού εμφανίζεται η παρακάτω φόρμα, στην οποία ο χρήστης συμπληρώνει τα στοιχεία που ζητούνται προκειμένου να ολοκληρωθεί η διαδικασία δημιουργίας λογαριασμού.



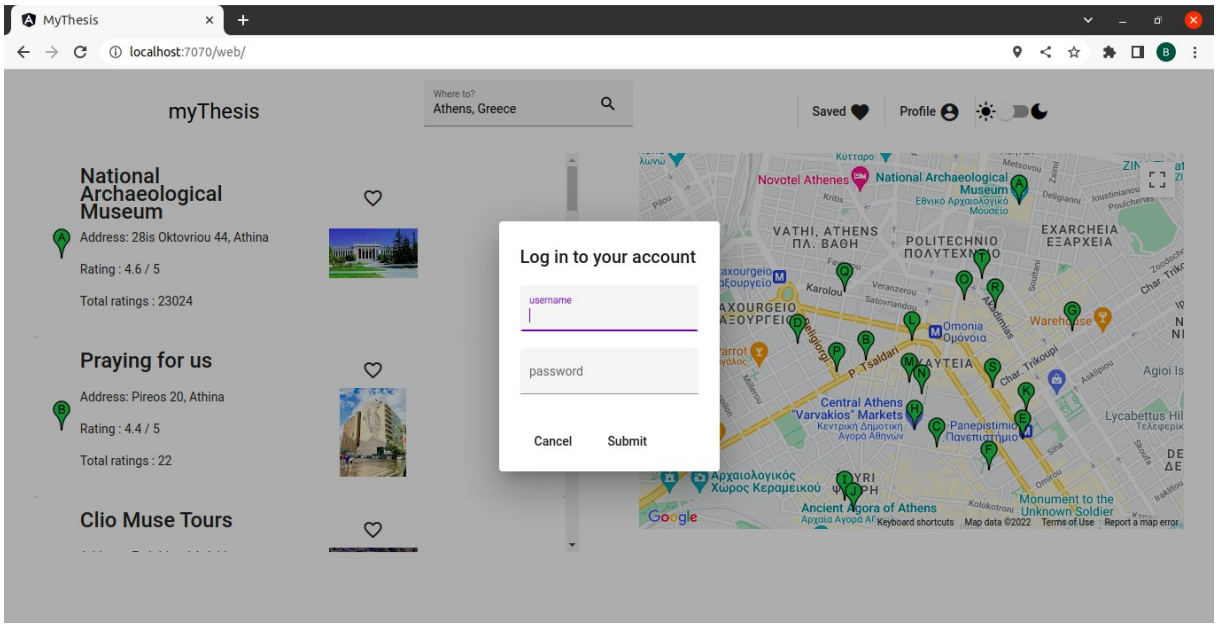
Εικόνα 11.4 Φόρμα δημιουργίας λογαριασμού

Στην περίπτωση που εισάγει κωδικό που δεν πληροί τις προϋποθέσεις, δηλαδή να περιέχει τουλάχιστον ένα μικρό χαρακτήρα, τουλάχιστον ένα κεφαλαίο γράμμα, τουλάχιστον έναν αριθμό και τουλάχιστον έναν ειδικό χαρακτήρα, όπως επίσης και το επιθυμητό μήκος χαρακτήρων, το συγκεκριμένο πλαίσιο γίνεται κόκκινο εξαιτίας της εφαρμογής του form validator του οποίου η λειτουργία είναι η βεβαίωση ότι η εισαγωγή στοιχείων είναι σωστή.



Εικόνα 11.5 Επισήμανση σφάλματος στον χρήστη

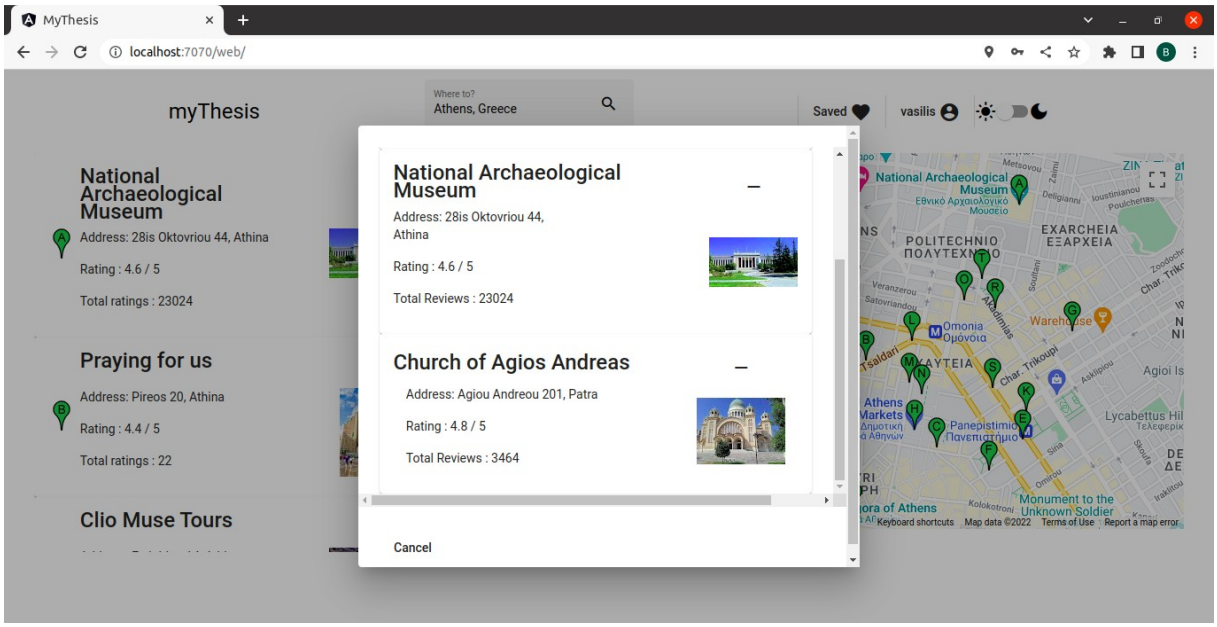
Εάν ο χρήστης είχε ήδη υπάρχον λογαριασμό θα έπρεπε να πατήσει την επιλογή που γράφει log-in και θα εμφανιζόταν το ακόλουθο παράθυρο στο οποίο ο χρήστης θα έπρεπε να εισάγει το username και τον κωδικό του.



Εικόνα 11.6 Φόρμα σύνδεσης στον λογαριασμό

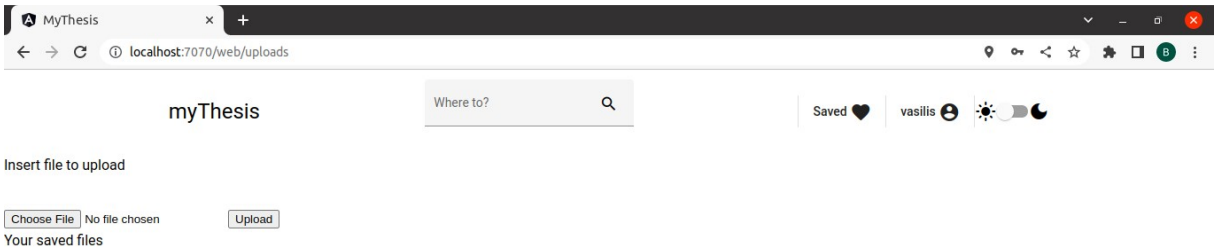
Έχοντας συνδεθεί στον λογαριασμό του παρατηρείται η αλλαγή στο μενού. Το κουμπί που γράφει Profile αλλάζει τιμή στο username του χρήστη και η επιλογές του χρήστη από εκεί αυξάνονται.

Πλέον ο χρήστης μπορεί να αποθηκεύει τα επιθυμητά σημεία ενδιαφέροντος καθώς και να βλέπει τα αποθηκευμένα στοιχεία πατώντας την αντίστοιχοι επιλογή από το μενού.



Εικόνα 10.7 Εμφάνιση αποθηκευμένων αξιοθέατων

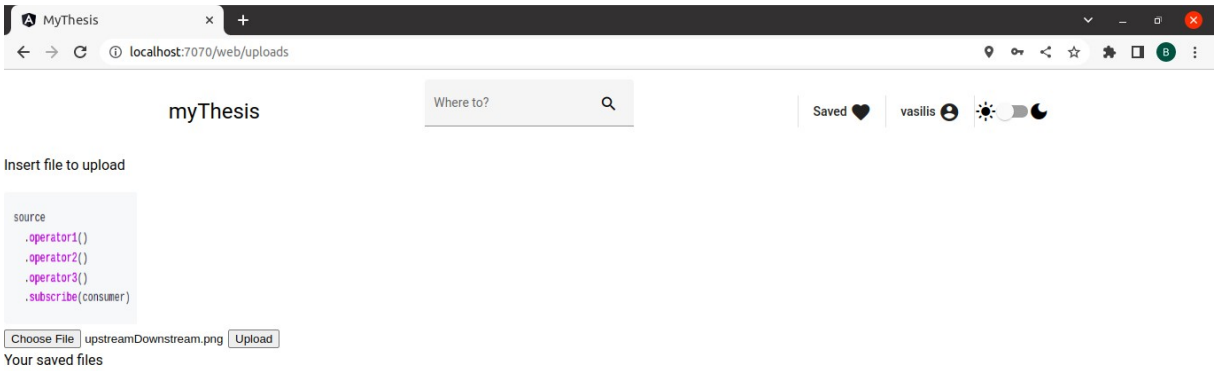
Έχοντας την δυνατότητα να κάνει upload επιθυμητές εικόνες στη βάση δεδομένων, επιλέγοντας να κάνει κλικ ο χρήστης στην συγκεκριμένη επιλογή πηγαίνει στη σελίδα που εκπληρώνει αυτό τον σκοπό. Η ενέργεια αυτή γίνεται με την βοήθεια του Angular Router αλλάζοντας το url. Πηγαίνοντας στην σελίδα οι φωτογραφίες που έχει ανεβάσει ο χρήστης είναι ορατές εκεί.



Εικόνα 10.8 Σελίδα ανεβάσματος αρχείων

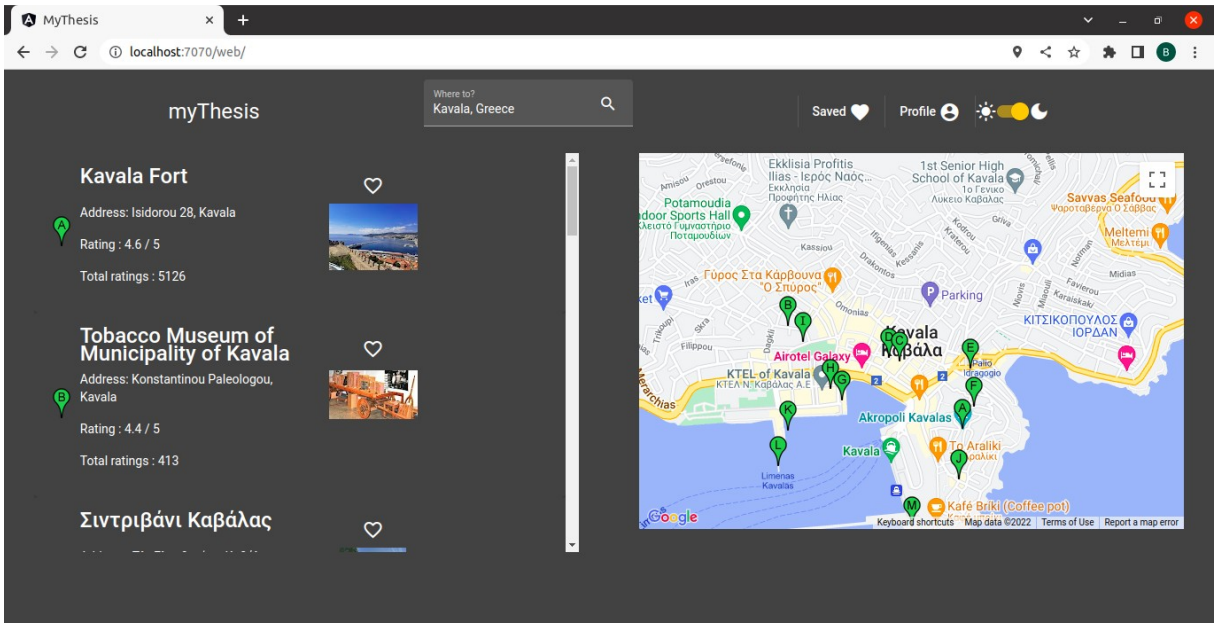
Έχοντας αλλάξει σελίδα ο χρήστης μπορεί να ξαναγυρίσει πίσω στην αρχική κάνοντας κλικ στο logo της εφαρμογής.

Στην περίπτωση επιλογής upload εικόνας, η εικόνα γίνεται render και εμφανίζεται η ονομασία του αρχείου προκειμένου να βεβαιωθεί ο χρήστης πως ανεβάζει την επιθυμητή φωτογραφία.



Εικόνα 10.9 Προεπισκόπηση εικόνας

Επιπλέον χαρακτηριστικό είναι το σκούρο θέμα το οποίο αλλάζει την όψη της εφαρμογής, αλλάζοντας σε πιο σκοτεινά χρώματα για να μην κουράζεται το μάτι του χρήστη.



Εικόνα 10.10 Λειτουργία σκούρου θέματος

11.4 Επίλογος

Σε αυτό το κεφάλαιο αναφέρθηκαν τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της διεπαφής του χρήστη. Συγκεκριμένα, η χρήση του Material Design της Google έδωσε μία μοντέρνα και ταυτόχρονα οικεία αίσθηση κατά την χρήση της εφαρμογής. Η Sass προσφέροντας εργαλεία που βοηθούν στην ευκολότερη συγγραφή CSS κανόνων βοήθησε κυρίως με χρήση των mixins να αναπτυχθεί το σκούρο θέμα και τα media queries κάνοντας την εφαρμογή προσβάσιμη και από κινητά. Τέλος, έγινε μία σύντομη Προεπισκόπηση της εφαρμογής προκειμένου παρουσιαστούν τα κύρια χαρακτηριστικά, οι λειτουργίες και η δομή της.

Κεφάλαιο 12ο: Συμπεράσματα και βελτιώσεις

12.1 Συμπεράσματα

Με την ολοκλήρωση της εφαρμογής προέκυψαν τα ακόλουθα συμπεράσματα όσο αναφορά τις τεχνολογίες που χρησιμοποιήθηκαν. Η επαφή με μοντέρνες τεχνολογίες που επιλέχθηκαν από τον ίδιο τον προγραμματιστή έκαναν την ανάπτυξη της εφαρμογής πιο ευχάριστη και ενδιαφέρουσα διαδικασία. Οι δυνατότητες που παρέχονται από την Angular τόσο στην ανάπτυξη της εφαρμογής όσο και στο γεγονός ότι αποτελεί ολοκληρωμένο framework, οδηγώντας στην μείωση χρήσης εξωτερικών βιβλιοθηκών, έκανε την ανάπτυξη της διεπαφής του χρήστη σχετικά απλή διαδικασία παρόλο την μεγάλη καμπύλη μάθησης του framework. Η χρήση του Docker οδήγησε στην ταχεία και εύχρηστη υλοποίηση της βάσης δεδομένων εξαλείφοντας την ανάγκη εγκατάστασης της βάσης στον υπολογιστή του προγραμματιστή. Επιπλέον, η επιλογή του Vert.x για την δημιουργία του API ήταν ιδιαίτερα ενδιαφέρουσα διότι η επαφή με έννοιες αντιδραστικού προγραμματισμού ήταν κάτι πρωτόγνωρο όπως και η χρήση εργαλείων της συλλογής ReactiveX που εξυπηρετούν αυτόν τον σκοπό.

Όσο αναφορά τα σχεδιαστικά πρότυπα και τις αρχιτεκτονικές που χρησιμοποιήθηκαν, προκύπτουν τα ακόλουθα συμπεράσματα. Γενικά η ύπαρξη ενός πρότυπου όσο αναφορά την σχεδίαση του λογισμικού είναι ιδιαίτερα σημαντική καθώς παρατηρείται σημαντική αύξηση ευκολίας στην συντήρηση και την επεκτασιμότητα του κώδικα. Στην περίπτωση του Domain Driven Design ο διαχωρισμός του κώδικα σε domain και infrastructure καθιστά εύκολη την μετάβαση αλλαγών που αφορούν ήδη υπάρχοντα χαρακτηριστικά, όπως παραδείγματος χάρη αλλαγή σε βάση, εστιάζοντας στο κομμάτι του infrastructure το οποίο αφορά τον τρόπο υλοποίησης της επιχειρησιακής λογικής. Όσο αναφορά το MVI, παρόλο που χρησιμοποιείται περισσότερο στις εφαρμογές Android, η λογική του μπορεί να υλοποιηθεί και στην περίπτωση του web με διάφορους τρόπους οδηγώντας σε ένα δομημένο κώδικα που είναι εύκολο να υλοποιηθούν tests και να αποφευχθούν λογικά λάθη από τους προγραμματιστές χάρης στην μονόδρομη κυκλική ροή των δεδομένων που υιοθετείται.

12.2 Βελτιώσεις

Στο κομμάτι του API μία βελτίωση που μπορεί να γίνει είναι η διάσπαση των Application Services σε microservices. Η τρέχουσα εφαρμογή αποτελεί μία μονολιθική εφαρμογή δηλαδή μία ενοποιημένη εφαρμογή ανεξάρτητη από άλλες. Αυτό έχει ως αποτέλεσμα την ευκολία στην εύρεση λαθών που ίσως προκύψουν, την διεξαγωγή testing και την ανάπτυξη του κώδικα. Ωστόσο, όσο μεγαλώνει σε μέγεθος η εφαρμογή παρουσιάζονται προβλήματα, όπως για παράδειγμα στο deployment, καθώς με μία αλλαγή θα πρέπει να γίνει εκ νέου deploy ολόκληρη η εφαρμογή. Μια λύση σε αυτό αποτελούν τα microservices. Με αυτή την προσέγγιση θα χωριζόντουσαν τα application services σε αυτόνομες ανεξάρτητες εφαρμογές όπου το κάθε ένα θα είχε την δική του βάση δεδομένων λύνοντας το κάθε ένα το δικό του επιχειρησιακό πρόβλημα. Με αυτό τον τρόπο παρέχεται ευελιξία στις ομάδες ανάπτυξης λογισμικού, να αναπτύξει η κάθε μία το δικό της κομμάτι χωρίς να υπάρχουν αναμονές και καθυστερήσεις. Επίσης μεγαλώνει η δυνατότητα του testing και της συντηρησιμότητας και παρέχει δυνατότητα συνεχούς διαθεσιμότητας της εφαρμογής, εφόσον μια αλλαγή σε ένα service θα σήμαινε το redeployment μόνο του εαυτού του και όχι ολόκληρης της εφαρμογής.

Στην μεριά του front end θα μπορούσε να προστεθεί ένα service το οποίο θα ήταν υπεύθυνο για να μετατρέπει τον κωδικό που εισάγει ο χρήστης σε κωδικοποιημένη μορφή βελτιώνοντας την ασφάλεια του συστήματος. Στην προκειμένη περίπτωση μιας και δεν ήταν απαραίτητο στα ζητούμενα της

εφαρμογής, ο κωδικός του χρήστη αποθηκεύεται στην βάση δεδομένων ως απλό κείμενο. Ωστόσο καλή πρακτική είναι να προστίθεται ένα έξτρα τμήμα στον κωδικό που εισάγει ο χρήστης (salt) και ύστερα ακόμη ένα αντίστοιχο αποκαλείται pepper (ή και secret salt) κατόπιν να κατακερματίζεται μέσω ενός αλγόριθμου όπως είναι για παράδειγμα ο SHA2 (secure hashing algorithm) και αποθηκεύεται στην βάση δεδομένων το ενιαίο κατακερματισμένο κείμενο μαζί με το salt. Σημαντική παρατήρηση αποτελεί πως το pepper δεν αποθηκεύεται στην βάση δεδομένων μαζί με το salt, αλλά φυλάσσεται αλλού προκειμένου να προφυλαχτεί σε περίπτωση παραβίασης της βάσης των δεδομένων από κάποια επίθεση. Σε συνδυασμό με αυτό θα μπορούσε να προστεθεί ένα authentication service στην μεριά του back end το οποίο θα παρείχε κάποια token όταν ο χρήστης θα ταυτοποιούσε την είσοδό του στον λογαριασμό του βελτιώνοντας ακόμα περισσότερο την ασφάλεια της εφαρμογής.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[1] Eric Evans, *Domain Driven Design : Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.

Internet Site

[2] Docker Documentation [Online]. Available:https://docs.docker.com .

[3] Docker [Software engineering] | IEEE Journal & Magazine | IEEE Explore [Online]. Available : https://ieeexplore.ieee.org/abstract/document/7093032

[4] Benchmarking Database Performance for Genomic Data [Online]. Available : https://onlinelibrary.wiley.com/doi/epdf/10.1002/jcb.25049?saml_referrer=

[5] Google Maps Platform [Online]. Available : https://developers.google.com/maps/documentation/

[6] Eclipse Vert.X [Online]. Available : https://vertx.io/docs/

[7] Angular [Online]. Available : https://angular.io/docs

[8] ReactiveX, "An API for asynchronous programming," [Online]. Available: https://reactivex.io/.

[9] RxJS, "Reactive Extensions Library for JavaScript," [Online]. Available: https://rxjs.dev/

[10] RxJava, “Reactive Extensions for the JVM”, [Online]. Available : https://github.com/ReactiveX/RxJava

[11] TypeScript, “TypeScript : JavaScript With Syntax For Types” [Online]. Available : https://www.typescriptlang.org/docs/

[12] Material Design [Online]. Available : https://m3.material.io/

[13] Sass, “Sass: Syntactically Awesome Style Sheets” [Online]. Available : https://sass-lang.com/