ΤΜΗΜΑ ΑΡΧΕΙΟΝΟΜΙΑΣ, ΒΙΒΛΙΟΘΗΚΟΝΟΜΙΑΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΗΣΗΣ
ΣΧΟΛΗ ΔΙΟΙΚΗΤΙΚΩΝ, ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΚΟΙΝΩΝΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

DEPARTMENT OF ARCHIVAL, LIBRARY AND INFORMATION STUDIES
SCHOOL OF MANAGEMENT, ECONOMICS AND SOCIAL SCIENCES

Πτυχιακή Εργασία

Ανάλυση, Σχεδιασμός και Ανάπτυξης μιας Progressive Web App: «Προσωπικό Εβδομαδιαίο Πρόγραμμα Μαθημάτων»

Συγγραφέας

Δημήτρης Ζαραχάνης (ΑΜ: 14024)

Επιβλέπων: Ιωάννης Τριανταφύλλου

Αθήνα, Μάρτιος 2021

Επιτροπή Εξέτασης

1. Ονοματεπώνυμο

Τριανταφύλλου Ιωάννης

2. Ονοματεπώνυμο

Δενδρινός Μάρκος

3. Ονοματεπώνυμο

Κουής Δημήτριος

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ/ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

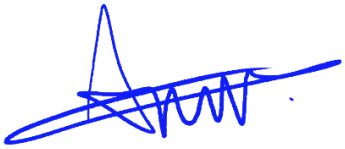
Ο κάτωθι υπογεγραμμένος, Δημήτριος Ζαραχάνης, με αριθμό μητρώου 14024 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Διοικητικών, Οικονομικών και Κοινωνικών Επιστημών του Τμήματος Αρχειονομίας, Βιβλιοθηκονομίας και Συστημάτων Πληροφόρησης, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

Δημήτριος Ζαραχάνης



Περίληψη στα ελληνικά

Στην παρούσα εργασία παρουσιάζεται η ανάλυση, ο σχεδιασμός και η υλοποίηση μιας διαδικτυακής εφαρμογής. Η εφαρμογή αναπτύχθηκε σε 3 κύρια επίπεδα. Το Back-End που βασίζεται στην υπηρεσία Firestore της Google Cloud Platform, το Middle Layer που έχει υλοποιηθεί με το NodeJS και την βιβλιοθήκη ExpressJS, και το Front-End που αναπτύχθηκε με βάση την βιβλιοθήκη React.

Τα τρία αυτά επίπεδα λειτουργούν συνεργατικά ώστε να παρέχεται στους φοιτητές μια διαδραστική εφαρμογή μέσω browser όπου μπορούν διαμορφώσουν το εβδομαδιαίο πρόγραμμα των μαθημάτων τους. Επιπλέον, με τη χρήση των νεότερων δυνατοτήτων που προσφέρουν οι web browsers, συγκεκριμένα με τις δυνατότητες των PWA, η εφαρμογή γίνεται να εγκατασταθεί σε συσκευές απευθείας από τον browser, χωρίς να χρειάζεται να μεσολαβεί κάποιο αποθετήριο εφαρμογών.

Η εργασία χωρίζεται σε τρία μέρη, ανάλυση, σχεδιασμός, υλοποίηση και σε κάθε μέρος αναλύεται το αντίστοιχο μέρος της ανάπτυξης της εφαρμογής, από τη σύλληψη μέχρι την υλοποίηση.

Περεταίρω στόχοι της εργασίας ήταν η εξερεύνηση και χρήση των πιο σύγχρονων τεχνολογιών που είναι διαθέσιμες στο χώρο του web-development και η πρακτική εφαρμογή εννοιών και γνώσεων που προσφέρονται από μαθήματα της τμήματος, όπως η Ανάλυση και Σχεδιασμός Συστημάτων και οι Εφαρμογές στον Παγκόσμιο Ιστό.

Λέξεις Κλειδιά: Λογισμικό, React, PWA, Firestore, Nodejs, Ανάπτυξη Εφαρμογών στον Παγκόσμιο Ιστό

Περίληψη στα αγγλικά

The focus of this thesis is the analysis, the design, and the implementation of a web-based application. The application is implemented in three layers. The Back-End layers is based on the Firestore service, offered by the Google Cloud Platform, the Middle Layers is implemented with the NodeJS runtime along with the ExpressJS web framework. Finally, the Front-End of the application is implemented with the ReactJS library.

The combination of these three layers provides the students/users of the app with an interactive web-based application where they can plan their weekly lessons' schedule. What is more, using the latest features offered by modern web browsers, namely the ability to distribute the app as a Progressive Web Application, the app can be directly installed on a user's device, without using an app store.

The thesis is divided into three parts, analysis, design, and implementation. In each part the corresponding part of the application's development is described in detail, from the application's conception to the development of the source code.

Some additional goals of this thesis were the exploration and usage of some of the latest web-development technologies and the practical application of the knowledge offered by some of the Department's Courses.

Keywords: Software, React, PWA (Progressive Web Application), Firestore, Nodejs, Web-development

Πίνακας περιεχομένων

Πίνακας Σχημάτων

Πίνακας Πινάκων

Κεφάλαιο 1. Εισαγωγή - Φάση Ανάλυσης

1.1 Μελέτη Σκοπιμότητας

Κάθε εξάμηνο οι σχολές εκδίδουν και μοιράζουν τα εβδομαδιαία προγράμματα με τα μαθήματα που παρακολουθούν οι μαθητές μέσα στην εβδομάδα. Ωστόσο, το πρόγραμμα παρουσιάζεται σε έγγραφα με πίνακες που δείχνουν όλα τα μαθήματα της εκάστοτε περιόδου και όλες τις επιλογές των εργαστηριακών ωρών. Αυτό καθιστά ελαφρώς δυσανάγνωστο το πρόγραμμα από τους φοιτητές, ειδικά όσους παρακολουθούν μαθήματα από άλλα εξάμηνα και συχνά έχουν επικαλύψεις στις ώρες διεξαγωγής των μαθημάτων.

Όντας φοιτητής του τμήματος αντιμετώπισα και εγώ αυτό το πρόβλημα και μετά από συζητήσεις με συμφοιτητές μου κατέληξα ότι όντως η παρουσίαση του προγράμματος είναι κάπως προβληματική και θα μπορούσε να βελτιωθεί με τη χρήση νέων τεχνολογιών.

Η λύση του προβλήματος είναι μια εφαρμογή για κινητά αλλά και σταθερούς υπολογιστές/laptops. Στη βάση της είναι μια Single Page Application (SPA) που σερβίρεται από έναν server, με τη διαφορά ότι με χρήση νεότερων δυνατοτήτων που προσφέρουν οι browsers μπορεί ουσιαστικά να εγκατασταθεί στην αρχική οθόνη των κινητών και να λειτουργήσει ακόμα και χωρίς σύνδεση στο Internet, φυσικά με αρκετά περιορισμένες δυνατότητες όσο είναι offline. Μπορεί να ανοίξει και να δείξει κάποια δεδομένα που έχει αποθηκεύσει τοπικά στη συσκευή, μέχρι να συνδεθεί πάλι στο Internet για να συγχρονιστεί με τα υπόλοιπα συστήματα.

Η εφαρμογή έχει σχεδιαστεί έτσι ώστε ο χρήστης να χρειάζεται να κάνει τις ελάχιστες δυνατές ρυθμίσεις και επιλογές για να διαμορφώσει το εβδομαδιαίο του πρόγραμμα. Η επιλογή αυτής της προσέγγισης έγινε στο πνεύμα της όσο το δυνατόν περισσότερης απλοποίησης της εμπειρίας χρήσης της εφαρμογής.

Ο φοιτητής συνδέεται γρήγορα στην εφαρμογή, μπορεί σε δευτερόλεπτα να διαμορφώσει ή και να δει το πρόγραμμά του και στη συνέχεια κλείσει την εφαρμογή ώστε να κάνει τις

υπόλοιπες δραστηριότητές του, μέχρι να χρειαστεί να την συμβουλευτεί εκ νέου. Όλα αυτά γίνονται είτε μέσω κινητού είτε μέσω κάποιας μεγαλύτερης οθόνης.

1.2 Περιορισμοί

Λόγω της φύσης της συγκεκριμένης εργασίας, υπάρχει πάρα πολλή ορολογία πληροφορικής και πιο συγκεκριμένα ορολογία από τον χώρο της Ανάπτυξης Λογισμικού η οποία είναι κατά κύριο στα Αγγλικά. Επειδή η εύρεση αντίστοιχων ελληνικών όρων δεν είναι εύκολη και σε κάποιες περιπτώσεις περισσότερο θα μπερδέψει τον αναγνώστη που πιθανόν θα ξέρει κάποιος όρους στα Αγγλικά αλλά όχι στα Ελληνικά, προτιμήθηκε να μεταφράζονται μόνο λέξεις που έχουν κάπως πιο καθιερωμένες ελληνικές αποδόσεις, όπως ο όρος “λογισμικό” ή “εφαρμογή”. Λέξεις που προέρχονται από πιο εξειδικευμένα πεδια όπως το “version control” ή “branch” θα παραμένουν στον αγγλικό τους τύπο.

1.3 Minimum Viable Product

Η εφαρμογή αναπτύχθηκε με τη λογική του Minimum Viable Product (MVP). Δηλαδή, δόθηκε έμφαση στην ταχεία ανάπτυξη ενός λειτουργικού και χρηστικού λογισμικού που παρέχει αξία στον χρήστη μέσω μιας φιλικής, γραφικής διεπαφής χρήστη (GUI).

Θεωρήθηκε απαραίτητο η εφαρμογή να προσφέρει περιορισμένο αριθμό δυνατοτήτων, αλλά οι δυνατότητες που προσφέρει να μπορούν να αξιοποιηθούν πλήρως, έτσι ώστε μέσω της ανατροφοδότησης από τους τελικούς χρήστες να βρεθούν ευκαιρίες ανάπτυξης επιπλέον δυνατοτήτων που έχουν μεγαλύτερο αντίκτυπο και ανταποκρίνονται στις ανάγκες των χρηστών. Σε αντίθετη περίπτωση υπάρχει κίνδυνος να επενδυθεί χρόνος για την ανάπτυξη δυνατοτήτων που τελικά δεν προσφέρουν την αναμενόμενη αξία στον τελικό χρήση (Agile Alliance , 2021).

Κεφάλαιο 2. Φάση Σχεδίασης

Στα παρακάτω κεφάλαιο θα παρουσιαστεί και θα τεκμηριωθεί η σκοπιμότητα χρήσης των επιμέρους τεχνολογιών με τη χρήση των οποίων αναπτύχθηκαν τα επίπεδα της εφαρμογής.

2.1 Επισκόπηση της δομής της Εφαρμογής

Η εφαρμογή χωρίζεται σε 3 επίπεδα. Στο πρώτο επίπεδο, το επίπεδο του client υπάρχει μια εφαρμογή βασισμένη στη βιβλιοθήκη ReactJS. Η συγκεκριμένη εφαρμογή συνδυάζει και χαρακτηριστικά μιας PWA με αποτέλεσμα να μπορεί να εγκατασταθεί στην οθόνη των κινητών συσκευών και λειτουργήσει ως αυτόνομη εφαρμογή, ακόμη και χωρίς σύνδεση στο διαδίκτυο.

Το τρίτο επίπεδο της εφαρμογής είναι ένα Back-End βασισμένο στην υπηρεσία Firestore που προσφέρεται μέσω του Google Cloud Platform. Συγκεκριμένα είναι μια NoSQL βάση δεδομένων, στην οποία αποθηκεύονται οι χρήστες μαζί με τις επιλογές τους καθώς και οι λίστες σχολών και χειμερινών και εαρινών περιόδων κάθε σχολής.

Η Firestore προσφέρει APIs σε πολλές γλώσσες προγραμματισμού, το οποίο επιτρέπει την προγραμματιστική πρόσβαση στα δεδομένα της βάσης, ανάγνωση και προσθήκη ή επεξεργασία. Για την παρούσα εφαρμογή έγινε χρήση του API για NodeJS από το Middle layer της εφαρμογής.

Το Middle Layer αποτελεί ουσιαστικά τον διαμεσολαβητή μεταξύ των αιτημάτων του client και του Back-End. Τα δύο κύρια μέρη του είναι ένα ExpressJS Rest-API και μια συλλογή με συναρτήσεις οι οποίες απλοποιούν το API της Firestore (Firestore-wrapper). Το REST-API δέχεται τις κλήσεις από του client με τη χρήση συγκεκριμένων endpoints και με τα δεδομένα που περιέχουν τα πακέτα καλεί συναρτήσεις του Firestore-wrapper, οι οποίες ανακτούν ή γράφουν δεδομένα στη βάση. Ακόμη ο Firestore-wrapper αναλαμβάνει την αυθεντικοποίηση των χρηστών.

Μια τυπική ροή δεδομένων προς και από τη βάση είναι η εξής:

Ο client στέλνει ένα http request με ή χωρίς data προς τον ExpressJS server. Ο server αναλόγως το endpoint, χειρίζεται διαφορετικά τα συνοδευτικά data και καλώντας τις μεθόδους του Firestore-wrapper κάνει τις κλήσεις προς το Firestore και επιστρέφει αντίστροφα την απάντηση από το Firestore στον client.

2.2 Ροή Χρηστών

Ο χρήστης μόλις φορτώσει την εφαρμογή καλείται να εγγραφεί στην εφαρμογή μέσω μιας φόρμας, είτε να εισέλθει στην εφαρμογή με τα στοιχεία που χρησιμοποίησε ήδη για να εγγραφεί. Μόλις ο χρήστης εγγραφεί/εισέλθει στην εφαρμογή βλέπει την αρχική οθόνη.

Η αρχική οθόνη χωρίζεται σε 3 μέρη. Αριστερά βρίσκεται ένας βοηθητικός χώρος με τα στοιχεία του χρήστη και στο υπόλοιπο μέρος της οθόνης βρίσκονται ο πίνακας όπου διαμορφώνεται το πρόγραμμα μαθημάτων, ενώ πάνω από την πίνακα βρίσκονται σε μορφή κουμπιών τα ονόματα των μαθημάτων που έχει επιλέξει ο χρήστης. Η επιλογή αυτή γίνεται σε ξεχωριστή σελίδα, όπου με checkboxes επιλέγονται τα μαθήματα στα οποία έχει εγγραφεί ο χρήστης για το τρέχον εξάμηνο, από τη λίστα όλων των μαθημάτων για το εαρινό ή το χειμερινό εξάμηνο, αναλόγως την περίοδο.

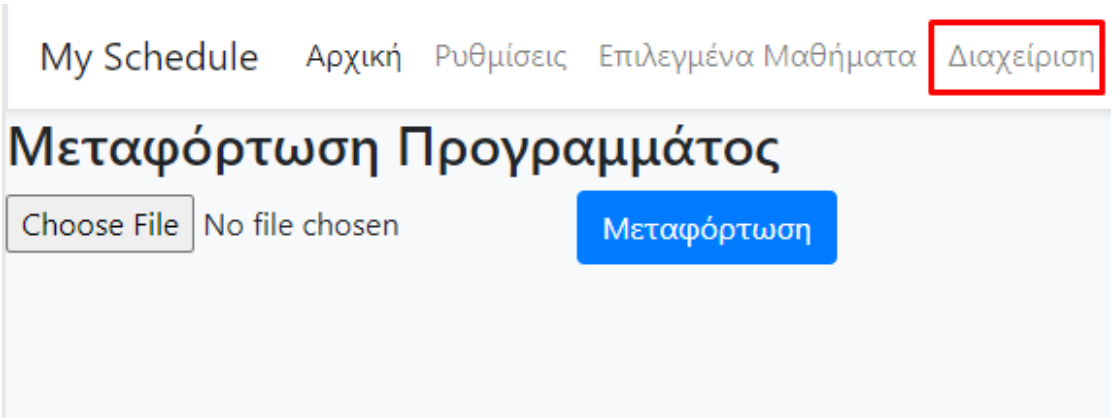
2.3 Ροή Διαχειριστή/Administrator

Η ροή και οι οθόνες που βλέπει ένα διαχειριστής είναι παρόμοιες με του χρήστη, αλλά με κάποιες σημαντικές διαφορές:

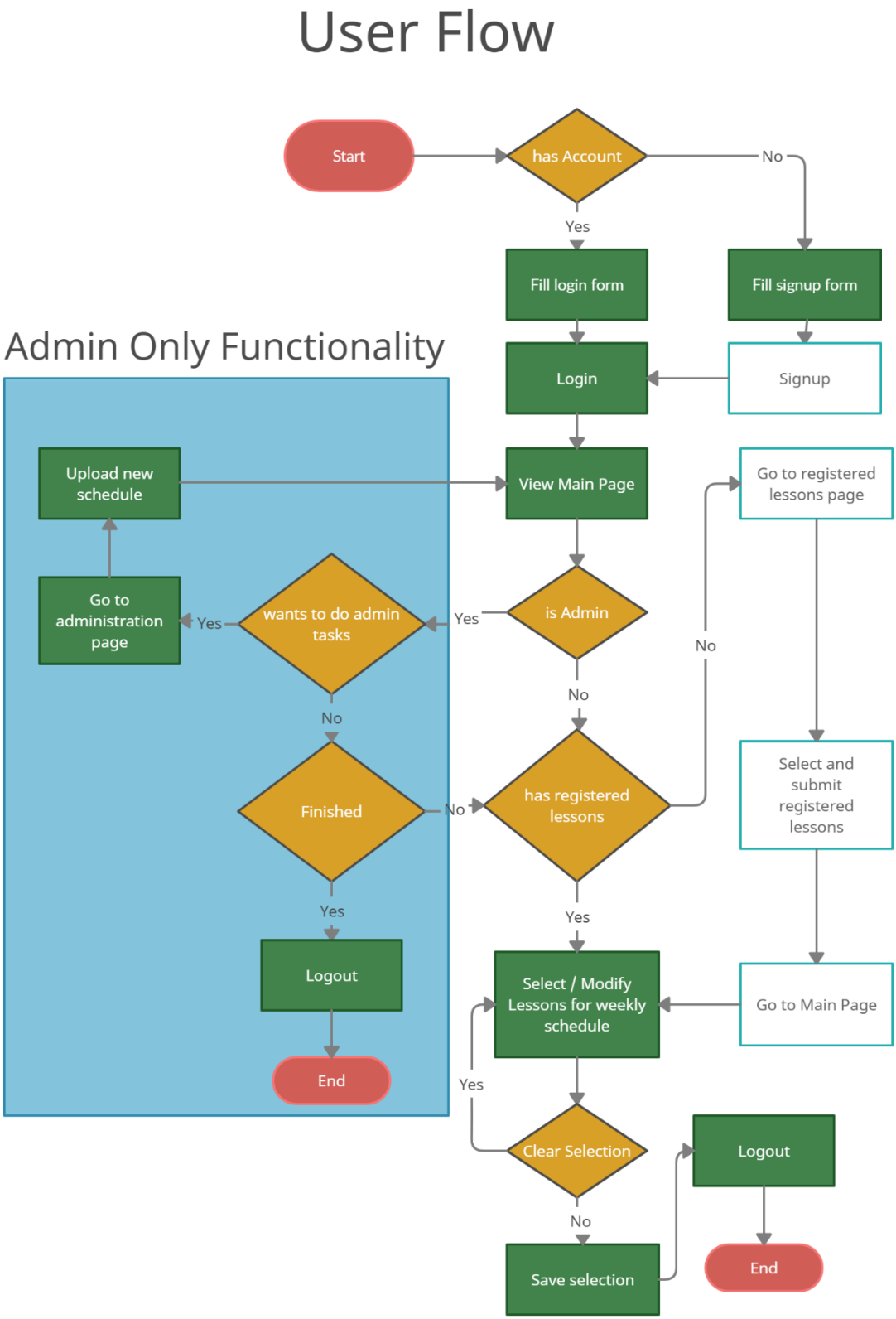
Στην βάση υπάρχουν δύο ρόλοι χρηστών, μαθητής και διαχειριστής. Μέσω της εγγραφής από την φόρμα της εφαρμογή όλοι οι χρήστες εγγράφονται ως φοιτητές. Δυνατότητα δημιουργίας χρήστη με τύπο διαχειριστή υπάρχει μόνο απευθείας στην βάση μέσω του GUI που παρέχει η πλατφόρμα, για λόγους ασφαλείας.

Μετά την είσοδο, ο διαχειριστής βλέπει ακριβώς την ίδια εφαρμογή με τον χρήστη, με την διαφορά όμως ότι έχει πρόσβαση σε μια ακόμη σελίδα με το όνομα “Διαχείριση”. Μέσω

αυτής της σελίδας μπορεί να ανεβάσει στη βάση ένα πρόγραμμα εξαμήνου σε μορφή JSON, το οποίο θα έχει προμηθευτεί από τρίτο.



Εικόνα 1 - Οθόνη αποστολής νέου προγράμματος



Εικόνα 2 - Γενικό διάγραμμα ροής χρήστη, μαζί με τη ροή διαχειριστή

2.4 Front-end - React & Redux

2.4.1 React - Redux

Για τις ανάγκες της εφαρμογής, αναπτύχθηκε ένας client με τη χρήση της βιβλιοθήκης React μαζί με μερικές ακόμα βοηθητικές βιβλιοθήκες που την εμπλουτίζουν. Η συγκεκριμένη βιβλιοθήκη επιλέχθηκε διότι με τα χαρακτηριστικά που προσφέρει μειώνει τον χρόνο που θα χρειαζόταν σε άλλη περίπτωση για να αναπτυχθεί ο κώδικας, καθώς το μεγαλύτερο μέρος της λειτουργικότητας της εφαρμογής θα υλοποιηθεί στον client.

Η συγκεκριμένη βιβλιοθήκη, μαζί με τις συμπληρωματικές της, παρέχει όλα τα απαραίτητα εργαλεία για να αναπτυχθεί η γραφική διεπαφή χρήσης της εφαρμογής, χωρίς να αφιερωθεί χρόνος στην υλοποίηση διαδικασιών όπως η ενημέρωση της οθόνης σε κάθε διάδραση του χρήστη, καθώς και η οργάνωση και διαχείριση του state της εφαρμογής, που δεν αφορούν άλλωστε τον σκοπό την εφαρμογής.

Αξίζει να αναφερθεί, ότι εφόσον ολόκληρη η εφαρμογή φορτώνεται με την πρώτη κλήση προς τον server οποιαδήποτε αλλαγή στην μπάρα διευθύνσεων (url address) τη διαχειρίζεται η βιβλιοθήκη React-Router, αλλάζοντας στην κατάλληλη “σελίδα/οθόνη” αναλόγως τη διεύθυνση. Ο server αποκρίνεται μόνο σε κλήσεις http με δεδομένα που βρίσκει από ή στέλνει προς τη βάση.

Τη διαχείριση του state της εφαρμογής την αναλαμβάνει αποκλειστικά η βιβλιοθήκη Redux, η οποία λειτουργεί συνεργατικά με τη React. Πιο συγκεκριμένα, όλα τα δεδομένα από τα οποία επηρεάζεται η λειτουργία της εφαρμογής ή και τι πρέπει να δείξει στην οθόνη φυλάσσονται στο Redux Store, το οποίο κάθε φορά που τροποποιείται, ειδοποιεί τη βιβλιοθήκη React ώστε να ενημερώσει την οθόνη με τα νέα δεδομένα. Επιπλέον, μέσω του Redux Store γίνονται και οι κλήσεις στον server.

Αν και η React είναι μια πιο απλή βιβλιοθήκη σε σχέση με τους “ανταγωνιστές” της όπως Vue και Angular, είναι αρκετά πολύπλοκη και ως εκ τούτου, μια πλήρης ανάλυση της λειτουργίας είναι εκτός του σκοπού της παρούσας εργασίας, μιας και θα μπορούσε να αποτελεί μια ολόκληρη εργασία από μόνη της.

Ακόμη, για να επιτευχθεί ένα ωραίο οπτικό αποτέλεσμα, χωρίς να αφιερωθεί πολύς χρόνος στην μελέτη και υλοποίηση ενός συστήματος σχεδιασμού αναφορικά με τον οπτικό σχεδιασμό και τη διαμόρφωση στο χώρο των γραφικών στοιχείων της εφαρμογής, επιλέχθηκε η βιβλιοθήκη Bootstrap 4, η οποία μέσω έτοιμων και πολυδοκιμασμένων CSS κανόνων παρέχει ένα έτοιμο σύστημα σχεδιασμού προς αξιοποίηση.

2.4.2 Δυνατότητες PWA

Η συγκεκριμένη εφαρμογή σχεδιάστηκε για να παρέχεται στους χρήστες και ως Progressive Web Application (PWA). Η Google ορίζει μια PWA ως μια διαδικτυακή εφαρμογή που τη διακρίνουν τρία βασικά χαρακτηριστικά. Η PWA πρέπει να είναι εγκαταστάσιμη (installable), αξιόπιστη (reliable) και να προσφέρει προηγμένες λειτουργικότητες (capable).

Το να είναι μια PWA εγκαταστάσιμη σημαίνει ότι η εφαρμογή μπορεί να εγκατασταθεί σε μια συσκευή και να λειτουργεί εκτός κάποιου web browser. Να εμφανίζεται σε αποκλειστικά δικό της “παράθυρο” και να δίνει την εικόνα ότι αποτελεί μια από τις διάφορες εγκατεστημένες εφαρμογής της συσκευής. Τέλος η εφαρμογή, ενδέχεται να προσφέρει τη δυνατότητα διεπαφής με τα υπόλοιπα συστήματα και λειτουργίες της συσκευής.

Μια αξιόπιστη PWA σημαίνει ότι η εφαρμογή λειτουργεί σε δυσμενείς συνθήκες σύνδεσης στο διαδίκτυο ή ακόμα και προσωρινά χωρίς καθόλου σύνδεση στο διαδίκτυο, καθώς και να φορτώνεται και να λειτουργεί γρήγορα. Αν υπάρχει περιορισμένη ή καθόλου σύνδεση στο διαδίκτυο η PWA πρέπει να μπορεί να δείξει στο χρήστη την τελευταία οθόνη και δεδομένα που έβλεπε και ενδεχομένως να προσφέρει κάποιες λειτουργίες αναξάρτητα από τη σύνδεση. Ακόμη είναι χρήσιμο σε δύσκολες συνθήκες η εφαρμογή να ενημερώνει με ελεγχόμενο τρόπο τον χρήστη, όπως κάποιο pop-up, για την κατάσταση παρά να βγάζει τεχνικά μηνύματα λάθους και οθόνες αποτυχίας.

Ως προηγμένες λειτουργικότητες η Google ορίζει δυνατότητες όπως οι βιντεοκλήσεις και η εικονική πραγματικότητα, οι οποίες πλέον έχουν γίνει προσβάσιμες στις τεχνολογίες του διαδικτύου μέσω νέων APIs, όπως το WebRTC και το WebVR. Τα παραπάνω ενισχύει η εμφάνιση της Web Assembly, η οποία επιτρέπει πλέον σε γλώσσες όπως η C και η C++ να παράξουν binaries που μπορούν να λειτουργήσουν σε περιβάλλον browser, συμπληρώνοντας ή αντικαθιστώντας ενίοτε τη JavaScript.

Στην εφαρμογή της εργασίας, οι δυνατότητες PWA προσφέρονται άμεσα από το περιβάλλον Create React App που χρησιμοποιήθηκε για να αναπτυχθεί η Front-End εφαρμογή. Η υποστήριξη δυνατοτήτων PWA γίνεται με τη χρήση της βιβλιοθήκης WorkBox, η οποία αναλαμβάνει να θέσει την υποδομή και την απαραίτητα παραμετροποίηση, έτσι ώστε να χρειάζεται πολύ λίγη παρέμβαση από τον προγραμματιστή για να μετατραπεί η εφαρμογή σε PWA. Σε αυτή τη φάση της ανάπτυξής της η εφαρμογή υποστηρίζει την λειτουργία χωρίς σύνδεση και τη δυνατότητα εγκατάστασης σε συσκευές (Google, 2020).

2.4.3 Μηχανή Αναπαράστασης Προγράμματος

Το κεντρικό χαρακτηριστικό της εφαρμογής είναι το διαδραστικό εβδομαδιαίο πρόγραμμα, το οποίο ο χρήστης μπορεί να διαμορφώσει όπως τον εξυπηρετεί καλύτερα.

Στο πάνω μέρος της οθόνες υπάρχουν τα μαθήματα στα οποία έχει εγγραφεί ο φοιτητής για το συγκεκριμένο εξάμηνο σε μορφή κουμπιών και στη μέση της οθόνης είναι ο πίνακας με τις μέρες και τις ώρες.

Όταν πατηθεί ένα πλήκτρο εμφανίζεται στον πίνακα στην αντίστοιχη μέρα και ώρες το μάθημα. Αν το μάθημα διαρκεί από τις 11:00 έως τις 14:00 την Τρίτη, στη γραμμή της Τρίτης θα γεμίσουν τρία κελιά που αντιστοιχούν στις ώρες 11:00, 12:00, 13:00. Τα κελιά παρουσιάζουν το όνομα του μαθήματος που περιέχουν και είναι χρωματισμένα είτε πράσινα είτε κίτρινα είτε κόκκινα.

Πράσινα χρωματίζονται τα κελιά των οποίων τα μαθήματα δε συμπίπτουν με κανένα άλλο μάθημα σε καμία από τις ώρες που διεξάγονται. Αν υπάρχει έστω και μια σύμπτωση ωρών, τότε τα κελιά στα οποία βρίσκεται η σύμπτωση γίνονται κόκκινα για την επισημάνουν, ενώ τα κελιά που μοιράζονται τα ίδια μαθήματα με αυτά που κοκκινίζουν γίνονται κίτρινα, για να υποδείξουν ότι υπάρχει σύμπτωση ωρών, αλλά όχι στο συγκεκριμένο κελί.

2.4.4 Παραδείγματα Χρήσης της Μηχανής Αναπαράστασης του Προγράμματος:

Τετάρτη		Διοίκηση Μονάδων Πληροφόρησης	Διοίκηση Μονάδων Πληροφόρησης, Βάσεις Δεδομένων	Βάσεις Δεδομένων	Βάσεις Δεδομένων
Πέμπτη			Διαχείριση Πολιτιστικών Αγαθών	Διαχείριση Πολιτιστικών Αγαθών	Διαχείριση Πολιτιστικών Αγαθών

Στην παραπάνω εικόνα φαίνονται δύο περιπτώσεις. Την Τετάρτη παρατηρούμε ότι συμπίπτουν τα μαθήματα “Διοίκηση Μονάδων Πληροφόρησης” με τις “Βάσεις Δεδομένων”. Η ώρα που υπάρχει η σύμπτωση εμφανίζεται κόκκινη για να φανεί ότι εκείνη την ώρα υπάρχει σύμπτωση μαθημάτων, ενώ οι άλλες ώρες και των δύο μαθημάτων εμφανίζονται κίτρινες για ενημερωθεί ο χρήστης ότι επηρεάζονται και τα δύο μαθήματα. Τέλος με πράσινο χρώμα εμφανίζεται η “Διαχείριση Πολιτιστικών Αγαθών” της Πέμπτης, διότι δεν υπάρχει καμία σύμπτωση με άλλο μάθημα τις ώρες που διεξάγεται.

14:00	15:00	16:00	17:00
Ιστορία και Φιλοσοφία των Επιστημών	Ιστορία και Φιλοσοφία των Επιστημών	Εκπαίδευση και Βιβλιοθήκες, Θεματικά Μουσεία	Εκπαίδευση και Βιβλιοθήκες, Θεματικά Μουσεία

Σε αυτή την περίπτωση το μάθημα “Ιστορία και Φιλοσοφία των Επιστημών” δεν συμπίπτει με κανένα άλλο μάθημα και εμφανίζεται με πράσινο χρώμα, ενώ τα μαθήματα “Εκπαίδευση και Βιβλιοθήκες” και “Θεματικά Μουσεία” εμφανίζονται μόνο κόκκινα, διότι οι ώρες διεξαγωγής του συμπίπτουν επακριβώς, οπότε ο φοιτητής σε αυτή την περίπτωση θα πρέπει να ξανασκεφτεί πως θα προσαρμόσει το πρόγραμμά του ώστε να μπορεί να παρακολουθήσει και τα δύο μαθήματα.

2.5 Middle Layer - Nodejs

Το Middle Layer (Middleware) της εφαρμογής αναλαμβάνει τη μεταφορά δεδομένων από τη βάση δεδομένων στον client και από τον client στη βάση δεδομένων αντίστοιχα. Ακούει σε HTTP κλήσεις σε διάφορα endpoints και με βάση τα στοιχεία του αιτήματος, γράφει δεδομένα στη βάση είτε ανακτά δεδομένα και τα στέλνει στον client.

Η εφαρμογή του Middleware έχει υλοποιηθεί σε περιβάλλον NodeJS μαζί με την βιβλιοθήκη ExpressJS, η οποία ουσιαστικά αποτελεί ένα επίπεδο abstraction πάνω από τις υπάρχουσες δικτυακές δυνατότητες του NodeJS.

Η διεπαφή με τη βάση (Firestore) επιτυγχάνεται μέσω του API που προσφέρει το ίδιο το Firestore μέσω βιβλιοθήκης για NodeJS. Ωστόσο, επειδή πολλές από τις εργασίες στη βάση βασίζονται σε παρόμοια “queries”, αναπτύχθηκε για τη συγκεκριμένη εφαρμογή ένα βοηθητικό επίπεδο abstraction με επαναχρησιμοποιούμενων συναρτήσεις, το οποίο απλοποιεί τις κλήσεις στο Firestore.

Ένας δεύτερος σκοπός που εξυπηρετεί το Middleware είναι η υλοποίηση της αυθεντικοποίησης του χρήστη. Παρότι η πλατφόρμα που φιλοξενεί το Firestore προσφέρει και δυνατότητες αυθεντικοποίησης, επιλέχθηκε η ανάπτυξη μιας διαδικασίας από την αρχή για να υπάρχει μεγαλύτερη ευελιξία και έλεγχος στην υλοποίηση της αυθεντικοποίησης.

Όταν γίνεται αίτηση για σύνδεση, το Middleware συγκρίνει τα διαπιστευτήρια που έχει υποβάλει ο χρήστης έναντι των εγγραφών του Firestore και αν βρεθεί ταίριασμα αποστέλλεται στον client το πακέτο των δεδομένων του χρήστη, ώστε να φορτωθεί η κεντρική οθόνη και να εμφανιστούν οι υπάρχουσες επιλογές του χρήστη.

Αντίστοιχα, όταν γίνει αίτημα για εγγραφή, το Middleware δημιουργεί στη βάση μια νέα εγγραφή χρήστη με τα διαπιστευτήρια που υποβλήθηκαν και αυτόματα συνδέει τον χρήστη με τα στοιχεία που υπέβαλε.

2.5.1.1 Μια σημείωση για τον όρο Middleware.

- Είναι γεγονός ότι ο όρος “Middleware” εμφανίζεται σε πολλά σημεία του κειμένου, ωστόσο δεν έχει την ίδια σημασία κάθε φορά. Στην περίπτωση των **middlewares** στο Redux Store του Front-End, που αναλαμβάνει τις ασύγχρονες κλήσεις στον Node/Express server, νοούνται οι συναρτήσεις που πραγματοποιούν τις κλήσεις αυτές.
- Το **Middleware ή Middle Layer** είναι ο ίδιος ο Node/ExpressJS server, ο οποίος λειτουργεί ως διαμεσολαβητής μεταξύ του Front-End και του Firestore Back-End.
- Τέλος μέσα στο REST API του Middle Layer, χρησιμοποιούνται κάποια **middlewares** τα οποία εκτελούν εργασίες πάνω στα εισερχόμενα πακέτα δεδομένων από τον client, για παράδειγμα ο χειρισμός αρχείων JSON.

2.5.2 Γιατί επιλέχθηκαν το NodeJS & Express

Το NodeJS μαζί με την βιβλιοθήκη ExpressJS ως πλατφόρμα ανάπτυξης του Middleware επιλέχθηκαν για τους εξής λόγους.

Το NodeJS είναι περιβάλλον ανάπτυξης σε JavaScript οπότε η συνολική διαδικασία ανάπτυξης της εφαρμογής επιταχύνεται, διότι πραγματοποιείται σε μία μόνο γλώσσα προγραμματισμού. Ακόμη για να ετοιμαστεί μια εφαρμογή σε NodeJS και Express απαιτείται πολύ λίγος κώδικας και παραμετροποίηση συγκριτικά με άλλες τεχνολογίες όπως .Net ή Spring Boot(Java). Μιας και η συγκεκριμένη εφαρμογή έχει μεγαλύτερο βάρος στον client θεωρήθηκε χρήσιμο να υλοποιηθεί πιο διεξοδικά το Front-End κομμάτι αρχικά και το Middleware να έχει πιο περιορισμένες δυνατότητες, οι οποίες απλώς εξυπηρετούν τις ανάγκες επικοινωνίας με τη βάση δεδομένων.

2.6 Back-end - Google Firestore

2.6.1 NoSQL Βάσεις Δεδομένων

Σύμφωνα με τον ιστότοπο της MongoDB, ο όρος NoSQL αναφέρεται σε οποιαδήποτε μη σχεσιακή βάση δεδομένων, δηλαδή βάσεις δεδομένων που δεν αποθηκεύουν δεδομένα σε μορφή σχεσιακών πινάκων. Παρόλα αυτά, δε χάνεται η ικανότητα της διασύνδεσης των δεδομένων, απλά αλλάζει τρόπος που δημιουργούνται οι σχέσεις. Η μη ανάγκη χωρισμού των σχετιζόμενων δεδομένων σε πίνακες, συχνά αποτελεί πλεονέκτημα καθώς απλοποιεί και καθιστά πολύ ποιο ευέλικτη τη μοντελοποίηση των οντοτήτων, αλλά και τη συντήρηση της βάσης.

Υπάρχουν τέσσερα είδη NoSQL βάσεων δεδομένων:

- Οι Document databases Αποθηκεύουν τα δεδομένα στα λεγόμενα “documents”, τα οποία έχουν παρόμοια μορφή με τα JSON αντικείμενα. Κάθε document περιέχει ζεύγη από key-values, όπου κάθε key αποτελεί το αναγνωριστικό ενός value. Τα values μπορούν να πάρουν ποικίλες μορφές, όπως strings, numbers, booleans, λίστες ή αντικείμενα και μπορούν να ταυτιστούν ακριβώς με τα αντικείμενα που χρησιμοποιούν οι προγραμματιστές στον κώδικά τους.
- Οι Key-value databases είναι πιο απλές βάσεις, παρόμοιες με τις Document. Οργανώνουν το περιεχόμενό τους απλά σε key-value ζεύγη. Η διαφορά με τις Document είναι ότι οι δεύτερες έχουν τα key-value pairs μέσα σε documents. Η ανάκτηση δεδομένων γίνεται μόνο με τη χρήση του key του. Θεωρούνται ιδανικές για την αποθήκευση μεγάλου όγκου δεδομένων και για χρήσεις που δεν απαιτούνται περίπλοκες ερωτήσεις για την ανάκτηση. Μια συνήθης χρήση των key-value βάσεων είναι το caching.
- Οι Wide-column databases αποθηκεύουν τα δεδομένα σε πίνακες, γραμμές και στήλες, με τη διαφορά από τις σχεσιακές βάσεις ότι δεν απαιτούν κάθε γραμμή να περιέχει ακριβώς τις ίδιες στήλες με τις άλλες γραμμές. Είναι χρήσιμες σε ανάγκες αποθήκευσης πολλών δεδομένων και ενώ υπάρχει ταυτόχρονα η δυνατότητα να προβλεφθούν τα μοτίβα των ερωτήσεων στο μέλλον.

- Οι Graph databases αποθηκεύουν δεδομένα σε κόμβους και ακμές. Στους κόμβους αποθηκεύονται πληροφορίες σχετικές με άτομα, μέρη ή αντικείμενα και οι ακμές περιγράφουν τις σχέσεις μεταξύ τους. Έχουν εφαρμογή σε περιπτώσεις αναζήτησης μοτίβων σε κοινωνικά δίκτυα ή σε μηχανές δημιουργίας προτάσεων, όπως διαφημίσεις (MongoDB, 2020).

2.6.2 Γιατί επιλέχθηκε η λύση του Firestore

Η βάση δεδομένων Firestore είναι μια υπηρεσία που προσφέρει η Google μέσω της πλατφόρμας Google Cloud Platform. Αποτελεί μια μη σχεσιακή βάση δεδομένων του τύπου Document, αν και τα documents της τα οργανώνει μέσα σε collections. Ακόμη, κάθε document δύναται να περιέχει υπό-συλλογές, ο οποίες περιέχουν δικά τους documents και ούτω καθεξής.

Η δυνατότητα να υπάρχει ευελιξία στα σχήματα των μοντέλων κατά την ανάπτυξη, η δωρεάν φιλοξενία της βάσης από την πλατφόρμα μέχρι ενός ορίου χρήσης, αλλά και η δυνατότητα εύκολης και άμεσης μελλοντικής επέκτασης ήταν τα βασικά χαρακτηριστικά που οδήγησαν στη χρήση της συγκεκριμένης λύσης ως βάση δεδομένων της εφαρμογής. Επιπλέον, το γεγονός ότι η συντήρηση της βάσης δεν είναι απόλυτα ευθύνη του προγραμματιστή, απελευθερώνει χρόνο, ο οποίος μπορεί να επενδυθεί στην ανάπτυξη της κυρίως λειτουργικότητας της εφαρμογής.

Πράγματι, χρειάστηκε αρκετές φορές να τροποποιηθούν τα σχήματα των μοντέλων κατά την ανάπτυξη και η ευελιξία του Firestore ήταν σημαντική βοήθεια.

Κεφάλαιο 3. Φάση Υλοποίησης

Στο παρακάτω κεφάλαιο θα αναλυθεί λεπτομερώς η χρήση των επιλεγμένων τεχνολογιών ανάπτυξης της εφαρμογής. Θα περιγραφεί ο τρόπος λειτουργίας των επιμέρους βιβλιοθηκών/frameworks και θα παρουσιαστούν κομμάτια από τον πηγαίο κώδικα.

3.1 Front-end

3.1.1 Περιβάλλον Ανάπτυξης

Για την υλοποίηση του front-end της εφαρμογής επιλέχθηκε η λύση React-Redux μέσω του Create React App (CRA). Αντί να διαμορφωθεί από το μηδέν το περιβάλλον ανάπτυξης, κάτι που θα κόστιζε σε χρόνο και θα ήταν ιδιαίτερα επιρρεπές σε λάθος παραμετροποιήσεις, προτιμήθηκε το περιβάλλον ανάπτυξης να παραχθεί μέσω του CRA.

Το CRA είναι μια εφαρμογή που λειτουργεί μέσω της γραμμής εντολών, η οποία παράγει ένα έτοιμο περιβάλλον ανάπτυξης για React και παρέχει την δυνατότητα να περιληφθούν και άλλες βιβλιοθήκες όπως η Redux. Το αποτέλεσμα της CRA είναι ένα περιβάλλον βασισμένο στο webpack το οποίο παρέχει δυνατότητες όπως έναν τοπικό development-server, hot-module reloading και άλλα, χαρακτηριστικά που καθιστούν πολύ πιο ταχεία και ευχάριστη την ανάπτυξη, επιτρέποντας στον προγραμματιστή να ασχοληθεί μόνο με το κομμάτι της ανάπτυξης της εφαρμογής και όχι με τη διαμόρφωση τις λεπτομέρειες του περιβάλλοντος ανάπτυξης.

Η ανάπτυξη μια εφαρμογής με βάση το CRA δεν είναι απλά συγγραφή html, css και javascript αρχείων που τα φορτώνονται μετά μέσω κάποιου web server. Στο περιβάλλον ανάπτυξης λειτουργεί ένας τοπικός NodeJs server και εκτελεί χρέη ενός προσωρινού web server, ο οποίος παρακολουθεί το φάκελο που βρίσκεται μέσα η εφαρμογή και σε κάθε αλλαγή σε κάποιο αρχείο, φορτώνει εκ νέου την εφαρμογή στον browser.

Μία από τις παραδοσιακές αδυναμίες της JavaScript είναι η περιορισμένη δυνατότητα που παρέχει στο χωρισμό του κώδικα σε modules. Το πρόβλημα αυτό λύνει το CRA με τη χρήση του Webpack. Το Webpack είναι ένα πρόγραμμα δέσμης αρχείων, το οποίο ενώνει τα διάφορα Javascript αρχεία της εφαρμογής σε ένα τελικό αρχείο που μπορεί εύκολα να χρησιμοποιήσει ο browser και αφού κάνει και κάποιες βελτιστοποιήσεις στον κώδικα, παράγει κάποια τελικά αρχεία, μη εύκολα αναγνώσιμα από άνθρωπο, τα οποία μπορούν να δοθούν τελικά σε οποιονδήποτε web server για να τα στέλνει στους χρήστες.

Το βήμα αυτό θα μπορούσε να χαρακτηριστεί καταχρηστικά ως “compilation”, αν και επί της ουσίας πρόκειται για βελτιστοποίηση ή και εν γένει transpilation, δηλαδή μετάφραση, της JavaScript από μια νέα έκδοση σε παλαιότερες εκδόσεις ή ακόμη και από TypeScript σε JavaScript για λόγους συμβατότητας με παλαιότερους περιηγητές. Αυτό αποτελεί και άλλο ένα από τα βασικά πλεονεκτήματα της χρήσης του Webpack, ότι δε χρειάζεται η ομάδα ανάπτυξης να ασχοληθεί εκτεταμένα με την υποστήριξη των παλαιότερων περιηγητών, καθώς η νεότερη JavaScript μεταφράζεται αυτόματα σε παλαιότερο συντακτικό διατηρώντας ακριβώς την ίδια λειτουργικότητα.

Με αυτόν τον τρόπο, αφενός παράγονται βελτιστοποιημένα από πλευράς επιδόσεων αρχεία JavaScript, ενώ ταυτόχρονα είναι και αρκετά δύσκολα να αναγνωστούν από κακόβουλες χρήστες χωρίς της χρήση εξειδικευμένων εργαλείων για να χρησιμοποιηθούν στην παραγωγή. Αφετέρου δεν υπάρχει ανάγκη διατήρησης κάποιου ειδικού web server κατά την ανάπτυξη, ο κώδικας μπορεί να χωριστεί σε όσα modules κρίνεται αναγκαίο, οι αλλαγές στον κώδικα να αποδίδονται στιγμιαία στον browser κατά την ανάπτυξη, ενώ μπορεί να γίνεται χρήση της νεότερης έκδοσης της JavaScript χωρίς το φόβο περιορισμένης συμβατότητας με παλιούς browsers.

3.1.2 Εργαλεία Ανάπτυξης

Το βασικό εργαλείο ανάπτυξης ήταν το VS Code της Microsoft. Το VS Code είναι ένας δωρεάν και ανοιχτού κώδικα επεξεργαστής κειμένου κατά βάση, αλλά οι δυνατότητες του μπορούν να ενισχυθούν σε πολύ μεγάλο βαθμό με πρόσθετα (Microsoft, 2016). Με τα σωστά

πρόσθετα παρέχονται δυνατότητες εφάμιλλες με αυτές που παρέχουν τα Integrated Development Environments όπως το WebStorm, τα οποία δεν παρέχονται δωρεάν.

Για τη συγκεκριμένη εφαρμογή, εκτός από τα προεγκατεστημένα πρόσθετα, χρησιμοποιήθηκαν κάποια επιπλέον που βοηθούν στην ανάπτυξη εφαρμογών με τη React.

Για το Version Control της εφαρμογής χρησιμοποιήθηκε το σύστημα Git και ο κώδικας βρίσκεται στο Github. Επειδή η ανάπτυξη έγινε από ένα άτομο, δε χρησιμοποιήθηκε κάποια πολύπλοκη διαδικασία, αλλά επιλέχθηκε να υπάρχει ένα master branch, στο οποίο προωθούνται αλλαγές ανά τακτά διαστήματα ανάπτυξης. Το συγκεκριμένο branch αντίστοιχα μεταφέρεται στο απομακρυσμένο αποθετήριο στο Github.

3.1.3 Δομή Πηγαίου Κώδικα

Τη δομή του πηγαίου κώδικα την διαμορφώνει στο μεγαλύτερο μέρος της το ίδιο το CRA. Στο ανώτερο επίπεδο βρίσκονται διάφορα configuration αρχεία, όπως το package.json το οποίο περιγράφει κάποια μεταδεδομένα της εφαρμογής, καθώς και τις βιβλιοθήκες που χρειάζεται η εφαρμογή για να δουλέψει. Ακόμη στο συγκεκριμένο αρχείο ορίζονται εντολές που εκτελούν κάποιες χρήσιμες λειτουργίες, όπως η εντολή για να τρέξει ο τοπικός server ανάπτυξης και η εντολή για να παραχθούν τα τελικά static files από τον πηγαίο κώδικα.

Σε αυτό το σημείο είναι σημαντικό να διευκρινιστεί ότι τα παραπάνω αφορούν το περιβάλλον που χρησιμοποιούμε για να παράξουμε τα τελικά static files που θα φορτώνονται από τον server και όχι την ίδια την Front-End εφαρμογή, η οποία τελικά αποτελείται από ελάχιστα, μεγάλα, αρχεία JavaScript, CSS, html και τυχόν πολυμέσα όπως εικόνες.

Δίπλα στα configuration files βρίσκεται ο φάκελος με όλα τα node-js dependencies που χρειάζεται το περιβάλλον της CRA για να λειτουργήσει, ένας φάκελος public όπου βρίσκονται static files όπως εικόνες και το index.html, το μόνο html αρχείο της εφαρμογής, πάνω στο οποίο, μέσω Javascript, αναρτάται ολόκληρη η εφαρμογή. Ακόμη, στον φάκελο public βρίσκεται το αρχείο manifest.json στο οποίο περιγράφονται οι απαραίτητες προδιαγραφές ώστε η εφαρμογή να αναγνωρίζεται ως Progressive Web Application (PWA) από τους browsers.

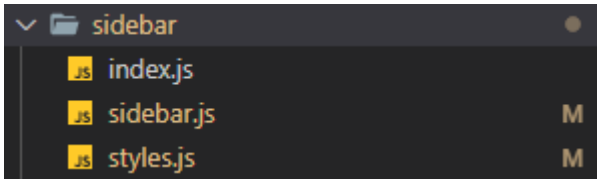
Στο ίδιο επίπεδο, αφού εκτελέσουμε την εντολή να παραχθούν τα τελικά static files, θα εμφανιστεί ο φάκελος build που τα περιέχει και μπορεί να μεταφερθεί ως έχει στον web server που διαμοιράζει την εφαρμογή.

Τελευταίος και σημαντικότερος είναι ο φάκελος src (εκ του source), όπου βρίσκεται όλος ο κώδικας την εφαρμογής, χωρισμένος σε υποφακέλους. Στο πρώτο του επίπεδο βρίσκονται τα αρχεία που χρειάζεται η React για να λειτουργήσει και να παραμετροποιηθεί στις ανάγκες της εκάστοτε εφαρμογής, καθώς και οι φάκελοι με τα components και το Redux Store.

Το Redux Store είναι το κεντρικό σημείο που αποθηκεύονται πληροφορίες όσο η εφαρμογή λειτουργεί. Οι πληροφορίες αφορούν από την κατάσταση της εφαρμογής, όπως αν ο χρήστης έχει συνδεθεί, μέχρι τα προσωπικά στοιχεία του εκάστοτε χρήστη, όταν αυτά φορτωθούν από τον server σε μεταγενέστερο χρόνο. Ακόμη, εκεί βρίσκεται ο κώδικας που ασχολείται με τις κλήσεις προς το Middleware, ώστε να διατηρείται πιο καθαρός ο κώδικας που αφορά το γραφικό/παρουσιαστικό κομμάτι της εφαρμογής.

Τις εργασίες παρουσίασης γραφικών και την πληροφορίας αναλαμβάνουν τα components. Το κεντρικό component βρίσκεται απευθείας μέσα στον src φάκελο και ονομάζεται App.js. Μέσα σε αυτό το component βρίσκονται τα σταθερά components, όπως η μπάρα πλοήγησης, η οποία φαίνεται καθόλη τη λειτουργία της εφαρμογής καθώς και Router Components τα οποία αναλόγως του URL στην γραμμή διευθύνσεων του browser φορτώνουν διαφορετικά components. Για παράδειγμα, αν το URL δείχνει “/” εμφανίζεται η φόρμα για σύνδεση ή εγγραφή. Άμα ο χρήστης συνδεθεί επιτυχώς, το URL αλλάζει σε “/my-schedule” και εμφανίζεται η κεντρική σελίδα διαμόρφωσης προγράμματος. Αυτές οι εργασίες γίνονται αυτοματοποιημένα από τη βιβλιοθήκη React-Router.

Κάθε component που έχει υλοποιηθεί από το μηδέν για την εφαρμογή βρίσκεται σε ένα φάκελο που φέρει το όνομά του. Μέσα στον φάκελο βρίσκονται συνήθως 3 αρχεία JavaScript. Το αρχείο index.js απλοποιεί την εισαγωγή components στο App.js, ενώ το styles.js περιγράφει κάποια styles για την εμφάνιση της component. Για να λειτουργήσει η μετάφραση από JavaScript σε CSS styles χρησιμοποιείται η βιβλιοθήκη JSS. Τέλος, στο αρχείο με το όνομα του component βρίσκεται το ίδιο το React Component με τη μορφή συνάρτησης. Κάθε component όταν κληθεί, ενδέχεται να λάβει κάποια δεδομένα είτε από το πατρικό του component είτε από το Redux Store, και αφού τα επεξεργαστεί παράγει html κώδικα.



Εικόνα 3 - Δομή ενός τυπικού Component της εφαρμογής

Σε κάποιες περιπτώσεις, κάποια components άρχισαν να διογκώνονται, οπότε θεωρήθηκε απαραίτητο κάποια κομμάτια τους να αποτελέσουν νέα components με την ίδια δομή. Τα components που παρουσιάζουν αυτή την ιδιαιτερότητα έχουν ακόμη ένα φάκελο στη δομή τους που περιέχει τα επιμέρους components που χρειάζονται.

3.1.4 React

Παρακάτω θα συνοψιστούν τα βασικά χαρακτηριστικά και λειτουργίες της βιβλιοθήκης, καθώς και κάποιων άλλων βιβλιοθηκών που λειτουργούν συμπληρωματικά στην React.

Η συγκεκριμένη βιβλιοθήκη δίνει πολύ μεγάλο βάρος στην οργάνωση του κώδικα σε “Components”, έτσι ώστε συνδυάζοντας πολλά ανεξάρτητα Components μπορούν να δημιουργηθούν πολύπλοκες εφαρμογές.

Ένα στοιχειώδες Component μπορεί να είναι είτε μια συνάρτηση είτε μια JavaScript κλάση (Class Component) που κληρονομεί από την βασική React.Component Class όλη τη λειτουργικότητα που απαιτείται για να λειτουργεί η βιβλιοθήκη σωστά και επιτρέπει στον προγραμματιστή να αναπτύξει τα διαφορά χαρακτηριστικά της εφαρμογής του με μεθόδους που θα υλοποιήσει μέσα στην κλάση του Component.

Οι μέθοδοι αυτές αναλαμβάνουν χωρίζονται στις μεθόδους διαχείρισης του component κατά τις διάφορες φάσεις του (lifecycle phases) και στις μεθόδους που ορίζει ο προγραμματιστής και μπορούν να εκτελούν οποιαδήποτε λειτουργία.

Οι πρώτες ασχολούνται όπως τι θα γίνει μόλις το component προστεθεί στο DOM, τι θα γίνει όταν τα δεδομένα που λαμβάνει αλλάξουν και τι θα γίνει τελικά όταν διαγραφεί από το DOM.

Να σημειωθεί ότι οι lifecycle μέθοδοι καλούνται αυτόματα από την ίδια τη βιβλιοθήκη όταν το component περνάει από τις διάφορες φάσεις της ζωής του.

Το δεύτερο είδος μεθόδων ορίζεται εξολοκλήρου από τον προγραμματιστή και είτε καλούνται από τις lifecycle είτε καλούνται όταν γίνεται διάδραση του χρήστη με την διεπαφή, πχ ένα click σε κουμπί ή scrolling.

Τελικά μέσω της μεθόδου render επιστρέφεται μια περιγραφική δομή στη γλώσσα JSX, που αποτελεί μίξη της JavaScript με την HTML. Μέσα στη δομή JSX υπάρχει κώδικας HTML με τα οικεία attributes, καθώς και ειδικά attributes της JSX που επιτρέπουν να γραφεί καθαρά πλουσιότερη λειτουργικότητα.

Για παράδειγμα, υπάρχει ένα <button> element και πρέπει στο click event να καλείται μια συγκεκριμένη method που ορίζεται στο class body. Στο <button> θα προστεθεί το attribute “onClick={}” και μέσα στις αγκύλες θα βρίσκεται το όνομα της μεθόδου που πρέπει να κληθεί. Αυτή η λειτουργικότητα θα φαίνεται έτσι:

```
<button
  className="btn btn-warning ml-2"
  onClick={(e) => this.handleSetLogin(e,!isLogin)}>
Switch to {isLogin ? 'Sign Up' : 'Login'}
</button>
```

Εικόνα 4 - Παράδειγμα κουμπιού με event listener

Η άλλη μορφή Component, και πλέον προτεινόμενη από τους δημιουργούς της βιβλιοθήκης, είναι Components ως συναρτήσεις (Functional Components). Κάθε Functional Component αποτελεί μια συνάρτηση, η οποία στο κυρίως σώμα της κάνει τους απαραίτητους υπολογισμούς και τροποποιήσεις σε δεδομένα που πρέπει να φανούν στην οθόνη και τελικά επιστρέφει, όπως και το Class Component, μια δομή JSX.

Η βασική διαφορά μεταξύ Class και Functional Components είναι κυρίως το πόσο πιο καθαρός και σύντομος είναι ο functional κώδικας σε αντίθεση με την προσέγγιση που βασίζεται σε κλάσεις.

Αξίζει να αναφερθεί ότι στις αρχικές εκδόσεις της βιβλιοθήκης, τα functional components, λόγω του γεγονότος ότι είναι συναρτήσεις, δε μπορούσαν να κρατήσουν state οπότε η χρήση τους περιοριζόταν σε components που απλά λάμβαναν δεδομένα και επέστρεφαν JSX με ίσως κάποιες τροποποιήσεις των δεδομένων. Ωστόσο, με την εισαγωγή των λεγόμενων "Hooks" μπορεί να κρατηθεί state μέσα στα functional components, καθώς και να υλοποιηθούν οι αντίστοιχες Lifecycle μέθοδοι, πλέον ως απλές συναρτήσεις. Λειτουργικά οι δύο προσεγγίσεις γραφής των Components παράγουν ακριβώς το ίδιο αποτέλεσμα, αλλά διαφέρουν στην σύνταξη και στο πως λειτουργούν στο παρασκήνιο.

Συνδυάζοντας με ιεραρχικό τρόπο πολλά components, σε ένα κεντρικό component δημιουργείται η τελική εφαρμογή όπου το κάθε component, με πρώτο το κεντρικό "καλεί" τα Children Components του. Ο κανόνας είναι ότι τα Components έχουν ένα εξωτερικό σημείο εισαγωγής δεδομένων input για δεδομένα, τα Component Props.

Τα Component Props ή Props περνιούνται σε κάθε Component από το πατρικό του Component μέσω JSX attributes. Τα συγκεκριμένα δεδομένα που περνάει ένα component στα «παιδιά» του, είτε τα έχει δεχτεί από το αντίστοιχο δικό του πατρικό component είτε τα πήρε από κάποιο κεντρικό Store (Redux). Ωστόσο, δεδομένα από κάποιο Store μπορεί να πάρει απευθείας και το ίδιο το Component Περισσότερα για τα Stores στην ενότητα του Redux και του State Management που έπεται.

Ύστερα, τα δεδομένα που έλαβε το καλούμενο Component μπορεί να τα τροποποιήσει όπως ορίζουν οι ανάγκες της εφαρμογής και είτε να τα δείξει μέσω της JSX είτε να τα περάσει στα δικά του «παιδιά» Components.

Στην ακόλουθη εικόνα φαίνεται το functional component που εμφανίζει την οθόνη από όπου μπορεί ο διαχειριστής να ανεβάσει στη βάση μια νέα περίοδο με μαθήματα.

```jsx
import React, { useState } from 'react';
import { uploadFile } from '../../features/store/stateSlice'

export default function AdminPanel() {

  const [fileToUpload, setFileToUpload] = useState();
  function handleChange(e) {
    setFileToUpload(e.target.files[0]);
  }
  async function handleClick() {
    await uploadFile(fileToUpload);
    alert('File Uploaded');
  }

  return (
    <div>
      <h3>Μεταφόρτωση Προγραμμάτος</h3>
      <input
        type="file"
        name="fileUpload"
        id="fileUpload"
        onChange={handleChange}
      />
      <button className="btn btn-primary" onClick={handleClick}>Μεταφόρτωση</button>
    </div>
  )
}

```

Εικόνα 5 - Παράδειγμα ενός functional component

3.1.5 State Management - Redux

Σε μια διαδραστική εφαρμογή, τις περισσότερες φορές χρειάζεται να διατηρείται κάποιο state για το UI της εφαρμογής, μεταξύ των διαδοχικών επανασχεδιασμών του UI από τον περιηγητή. Μία λύση σε αυτό το πρόβλημα είναι η τήρηση ενός state αντικειμένου ως property σε ένα class component ή με τη χρήση του λεγόμενου “useState hook” στα functional components.

Ουσιαστικά το state είναι μια δομή που μένει αναλλοίωτη από το rendering του UI από τον browser και μπορεί να αλλαχθεί μόνο με τρόπους που έχουν οριστεί από τον προγραμματιστή.

Κάθε Component μπορεί να διατηρεί το δικό του state εσωτερικά. Οποιαδήποτε αλλαγή στο state του Component ή στο Store αυτόματα πυροδοτεί τον επανασχεδιασμό του UI από το σημείο που βρίσκεται στο δέντρο του DOM το Component του οποίο το state άλλαξε και ιεραρχικά προς τα κάτω, δηλαδή και των παιδιών του. Μέσω εγγραφών που βρίσκονται στο state μπορούν να περνιούνται και data στα child components μέσω των props που

αναφέρθηκαν παραπάνω.

Η παραπάνω προσέγγιση, σε μικρές εφαρμογές με μικρή πολυπλοκότητα και λίγα συνολικά components μπορεί να δουλέψει σχετικά καλά. Ωστόσο, όταν η εφαρμογή είναι πλέον μεγάλη, γίνεται γρήγορα κουραστική και δύσκολη η διαχείριση του state.

Για να λυθεί το πρόβλημα όπου κάθε Component έχει δικό του state, από το οποίο τροποποιεί τον εαυτό του και τα παιδιά του στο render, μια πρώτη λύση είναι να κρατηθεί το state όλη της εφαρμογής στο κεντρικό Component. Ό,τι data χρειάζονται τα Components πιο κάτω στην ιεραρχία μπορούν να περνιούνται επίπεδο-επίπεδο μέσω props.

Ωστόσο, και αυτή η προσέγγιση έχει πολλά προβλήματα. Για παράδειγμα, αν ένα Component που βρίσκεται 10 επίπεδα κάτω από το κεντρικό Component χρειάζεται πληροφορίες από το state για να παρουσιάσει τα γραφικά που πρέπει, θα χρειαστεί να περαστεί η συγκεκριμένη πληροφορία από όλα τα ενδιάμεσα επίπεδα πρώτα, με προσοχή στα ονόματα των μεταβλητών που δίνονται ως props. Ακόμα και να γίνει αυτό, άμα χρειαστεί να αλλάξει το όνομα μιας τιμής στο state θα χρειαστεί να αλλάξει και στα ονόματα μεταβλητών σε πάρα πολλά σημεία μέχρι να φτάσει πάλι στο τελευταίο Component που το χρειάζεται, οπότε προκύπτουν σοβαρά προβλήματα συντήρησης της εφαρμογής.

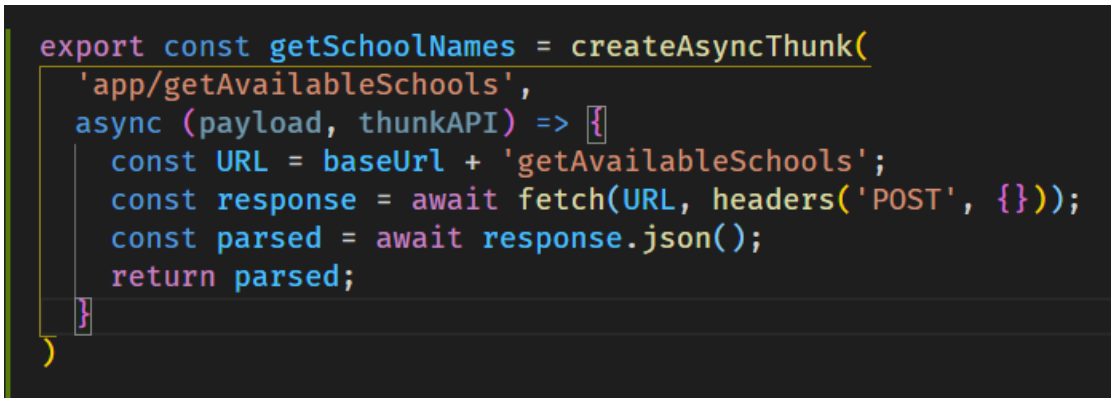
Μια προσέγγιση που λύνει τα παρακάτω προβλήματα είναι η εξαγωγή όλου του state της εφαρμογής σε ένα “store” που περικλείει όλη την εφαρμογή και στο οποίο έχουν πρόσβαση άμεσα όλα τα components στην ιεραρχία. Οπότε αμέσως καθίσταται άμεση και εύκολη η πρόσβαση στα δεδομένα του state από οποιοδήποτε σημείο, καθώς και η τροποποίησή τους. Στη συνέχεια, το συγκεκριμένο store “ειδοποιεί” στο παρασκήνιο όλα τα components τα οποία συνδέονται μαζί του, ότι η κατάσταση της εφαρμογής άλλαξε και άρα πρέπει να επανασχεδιαστούν στην οθόνη με τα νέα δεδομένα.

```
export const stateSlice = createSlice({
  name: 'state',

  initialState: {
    loggedIn: false,
    user: {
      localId: '',
      currentSemester: '',
      name: ' ',
      registryNumber: '',
      sessionExpiresIn: 0,
      email: '',
      isAdmin: false
    },
    registeredLessons: [],
    selectedLessons: [],
    tableValues: genDaysTable(5, 13),
    asyncLessons: [],
    lessonNames: [],
    filesToUpload: [],
    availableSchoolNames: [],
  },
```

Εικόνα 6 - Η αρχική δομή του state της εφαρμογής

Στην παρακάτω εικόνα φαίνεται η συνάρτηση η οποία χρησιμοποιείται για να φορτωθούν στην εφαρμογή από τον server τα ονόματα των διαθέσιμων σχολών για να διαλέξει ένας χρήστης κατά την εγγραφή του.



```
export const getSchoolNames = createAsyncThunk(
  'app/getAvailableSchools',
  async (payload, thunkAPI) => {
    const URL = baseUrl + 'getAvailableSchools';
    const response = await fetch(URL, headers('POST', {}));
    const parsed = await response.json();
    return parsed;
  }
)
```

Εικόνα 7 - Συνάρτηση που ανακτά τα ονόματα των διαθέσιμων σχολών

3.1.6 Bootstrap 4

Όπως κάθε διαδικτυακή εφαρμογή, η εφαρμογή της παρούσας εργασίας δίνει έμφαση στην ευχρηστία και στην καλαισθησία. Σε περιβάλλον Web Browser η διαμόρφωση και η κατανομή των γραφικών στοιχείων, όπως κείμενο, κουμπιά και εικόνες γίνεται κατά βάση με τη γλώσσα CSS. Ορίζονται κάποια κανόνες σε αρχεία CSS ο οποίοι περιγράφουν πως πρέπει να εμφανίζονται τα στοιχεία.

Η συγγραφή τέτοιων αρχείων είναι χρονοβόρα και χρειάζεται εκτεταμένους ελέγχους για να εξασφαλιστεί ότι οι κανόνες είναι σωστοί σε όλες τι περιπτώσεις και περιηγητές. Αυτά τα προβλήματα λύνουν οι βιβλιοθήκες CSS.

Στη συγκεκριμένη εργασία προτιμήθηκε η χρήση της βιβλιοθήκης Bootstrap 4. Η Bootstrap είναι μια καθιερωμένη και πολυδοκιμασένη βιβλιοθήκη, η οποία με κανόνες που μας παρέχει έτοιμους σε μορφή CSS κλάσεων, εξασφαλίζει ότι τα γραφικά στοιχεία της εφαρμογής θα παρουσιάζονται σε όλες τις περιπτώσεις σωστά. Επιπλέον, σχεδιαστικά οι κανόνες της Bootstrap παράγουν ένα αρκετά καλαίσθητο αποτέλεσμα εμπειρίας χρήστη, χωρίς να χρειάζεται ιδιαίτερη αισθητική παρέμβαση από τον προγραμματιστή.

Μια σημαντική σημείωση είναι ότι για να επιτύχει διάφορες διαδράσεις και animations, όπως μενού που κρύβονται, η Bootstrap 4 που χρησιμοποιείται στην εφαρμογή χρειάζεται την βιβλιοθήκη jQuery. Ωστόσο, επειδή η συνύπαρξη της React με την jQuery είναι προβληματική μιας και οι δύο βιβλιοθήκες ασχολούνται εξίσου με το UI, αποφασίστηκε να παραληφθεί η jQuery και να συμπεριληφθούν μόνο τα CSS κομμάτια της Bootstrap.

Το κενό που αφήνει η jQuery, σε όποιο σημείο χρειάστηκε, καλύφθηκε με απλή Javascript και με εργαλεία της React. Έτσι διασφαλίζεται ότι δεν θα υπάρξουν συγκρούσεις στον κώδικα του UI.

3.2 Middleware – Backend

Το Middleware έχει υλοποιηθεί με το Nodejs, ένα Runtime περιβάλλον βασισμένο στη JavaScript. Οι δύο βασικοί λόγοι επιλογής του ήταν ότι προσφέρει δυνατότητες ταχείας ανάπτυξης και διευκολύνει και επιταχύνει περεταίρω τη συνολική ανάπτυξη της εφαρμογής καθώς είναι στην ίδια γλώσσα με το React, Front-End App.

Αν και το Node παρέχει APIs για την ανάπτυξη ενός web server, η βιβλιοθήκη Express αποτελεί τον πλέον διαδεδομένο τρόπο ανάπτυξης web servers σε περιβάλλον Node. Το Express ουσιαστικά αποτελεί ένα abstraction πάνω από τα υπάρχοντα web APIs του Node, το οποίο μειώνει σημαντικά τον απαραίτητο κώδικα, ενώ τον κάνει και πιο ευανάγνωστο. Επιπλέον, με τη χρήση των λεγόμενων “middlewares” μπορούν να προστεθούν νέες δυνατότητες σε διάφορα σημεία που απαιτούνται, όπως ο χειρισμός ενός request που φέρει ένα πακέτο δεδομένων με τη μορφή JSON.

Η Middleware εφαρμογή αποτελείται από ένα Rest API, το οποίο περιμένει κλήσεις από τον client και αναλόγως την κλήση κάνει εργασίες στο Firestore. Οι εργασίες στη βάση πραγματοποιούνται μέσω του επίσημου API που προσφέρει το Firestore. Οι εργασίες στη βάση, για μείωση πολυπλοκότητας και επανάληψης έχουν οριστεί μέσα σε επαναχρησιμοποιούμενες συναρτήσεις, τις οποίες καλούν τα endpoints του Middleware, περνώντας τους κάποιες παραμέτρους από τον client, όπου είναι απαραίτητο.

3.2.1 Περιβάλλον Ανάπτυξης

Το Middleware αναπτύχθηκε με το NodeJS. Η δομή του είναι αρκετά απλή όπως και η λειτουργικότητά του. Η εφαρμογή εκκινεί με την εντολή “node main”, η οποία τρέχει σε περιβάλλον NodeJs το αρχείο main.js, που αποτελεί το entry point του Server.

3.2.2 Εργαλεία Ανάπτυξης

Το εργαλείο ανάπτυξης του Middleware ήταν το VS Code, το οποίο χρησιμοποιήθηκε και για το Front-End App. Για να επιταχύνεται η ανάπτυξη του κώδικα χρησιμοποιήθηκε ένα πρόσθετο πάνω από στο NodeJS, το nodemon, το οποίο παρακολουθεί τα αρχεία του φακέλου της εφαρμογής και μόλις εντοπίσει κάποια αλλαγή, εκτελεί από την αρχή τον server. Η χρήση του nodemon γίνεται με την εντολή “nodemon main”, η οποία ουσιαστικά αντικαθιστά τη “node main”.

3.2.3 Δομή Πηγαίου Κώδικα

Σε αντίθεση με το Front-End η δομή του Middleware δεν παρουσιάζει μεγάλη πολυπλοκότητα. Αποτελείται από ένα φάκελο, στον οποίο περιέχονται τα JavaScript αρχεία που εκτελούν τον Express Web Server καθώς και από κάποιους ακόμη φακέλους.

Συγκεκριμένα, δίπλα στα αρχεία του Web Server βρίσκεται ο φάκελος «node_modules» που περιέχει τις απαραίτητες NodeJs βιβλιοθήκες για να λειτουργεί ο server, καθώς και ο φάκελος build όπου φιλοξενείται το αποτέλεσμα της διαδικασίας build του Front-End, δηλαδή τα στατικά αρχεία που θα στέλνονται στον χρήστη όταν προσπαθεί να δει την εφαρμογή μέσω browser.

Η κυρίως εφαρμογή βρίσκεται μέσα στο αρχείο “main.js”, το οποίο αποτελεί και το entry point της εφαρμογής, το αρχείο δηλαδή που χρησιμοποιεί το Runtime του Nodejs για να εκκινήσει τον web server. Για να απλοποιηθεί η δομή του κώδικα στο main.js κάποιες λειτουργικότητες έχουν αναπτυχθεί σε ξεχωριστά JavaScript αρχεία, από τα οποία εισάγονται συναρτήσεις και κλάσεις στο main.js για να χρησιμοποιηθούν.

Το σημαντικότερο βοηθητικό αρχείο είναι το firestoreClient.js. Στο συγκεκριμένο αρχείο γίνεται μέσω επαναχρησιμοποιούμενων συναρτήσεων χρήση του API που προσφέρει το Firestore, έτσι ώστε να μη χρειάζεται σε κάθε περίπτωση να γράφονται αναλυτικά τα queries προς το Firestore. Ακόμη, υπάρχουν κάποια αρχεία τα οποία φιλοξενούν βοηθητικές λειτουργικότητες, οι οποίες θα φανούν χρήσιμες περισσότερο στην μελλοντική ανάπτυξη της εφαρμογής, αλλά προς το παρόν δε χρησιμοποιούνται εκτενώς, όπως κάποιες συναρτήσεις επαλήθευσης των στοιχείων που υποβάλλονται και περιγραφές των μοντέλων της βάσης.

3.2.4 FireStore Wrapper

Το Firestore προσφέρει ένα χρήσιμο API ως βιβλιοθήκη για πολλές γλώσσες προγραμματισμού. Στη συγκεκριμένη εφαρμογή χρησιμοποιήθηκε η βιβλιοθήκη για Nodejs. Για να χρησιμοποιηθεί το API πρέπει να οριστεί μέσα στον κώδικα του server ένα αντικείμενο που αναλαμβάνει να εκτελεί τις διαδικασίες προς τη βάση.

```
// Configure Firestore
const admin = require('firebase-admin');
admin.initializeApp({
  credential: admin.credential.cert(DBKEY)
});
const db = admin.firestore();
```

Εικόνα 8 - Αρχικοποίηση του API που επικοινωνεί με το Firestore

Το DBKEY είναι απαραίτητο για τη σύνδεση με τη βάση και φυλάσσεται σε ένα JSON αρχείο στον φάκελο του κώδικα της εφαρμογής. Κατά την έναρξη του server το κλειδί αυτό διαβάζεται και το περιεχόμενό του χρησιμοποιείται για την σύνδεση με τη βάση. Να σημειωθεί ότι στο αποθετήριο στο Github κλειδί αυτό δεν είναι διαθέσιμο για λόγους ασφαλείας. Το αντικείμενο που χρησιμοποιείται για τις εργασίες στη βάση ορίζεται ως db στον κώδικα.

Οι δυο συλλογές της βάσης φυλάσσονται σε δύο μεταβλητές ώστε να μη χρειάζεται να πληκτρολογούνται κάθε φορά που πρέπει να γίνει κάποια ενέργεια στη βάση.

```
// Collections References
const USERS_COLL = db.collection('users');
const SCHOOLS_COLL = db.collection('schools');
```

Εικόνα 9 - Οι μεταβλητές των συλλογών της βάσης

Με την χρήση των παραπάνω μεταβλητών μπορούν να γίνουν εγγραφές ή τροποποιήσεις στη βάση, στην αντίστοιχη συλλογή. Για παράδειγμα η εγγραφή ενός χρήστη:

```
const createUserDocument = async (userId, data) => {
  const res = await USERS_COLL.doc(userId).set(data);
  return res;
}
```

Εικόνα 10 - Συνάρτηση που δημιουργεί έναν νέο χρήστη στη βάση

Παραπάνω βλέπουμε μια ασύγχρονα εκτελέσιμη συνάρτηση η οποία δέχεται κάποια δεδομένα ως παραμέτρους και με βάση αυτά ορίζει στη συλλογή users ένα νέο document με key το userId και ως value το αντικείμενο των data. Η λέξη-κλειδί await εξασφαλίζει ότι αφού

ολοκληρωθεί η κλήση θα οριστεί τότε η μεταβλητή res (εκ του response) και θα επιστραφεί στη συνάρτηση η οποία κάλεσε την createUserDocument ώστε να χρησιμοποιηθεί η απάντηση της βάσης όπου αλλού χρειάζεται.

Τα παραπάνω αποσπάσματα από τον κώδικα βρίσκονται στο αρχείο firestoreClient, το οποίο οργανώνει και απλοποιεί όλη τη λειτουργικότητα από και προς τη βάση, έτσι ώστε στον υπόλοιπο κώδικα να καλούνται οι παραπάνω συναρτήσεις και όχι η σύνταξη που της διεπαφής του Firestore, καθώς είναι αρκετά πιο πολύπλοκη.

3.2.5 Rest API

Στη συγκεκριμένη ενότητα θα αναλυθεί ο τρόπος λειτουργίας του REST API, το οποίο αναλαμβάνει να ακούει τις κλήσεις από το Front-End και να μεταφέρει δεδομένα από και προς τη βάση δεδομένων.

Όπως αναφέρεται και παραπάνω, ο http server έχει υλοποιηθεί με τη χρήση της βιβλιοθήκης ExpressJS. Η ExpressJS είναι μια μινιμαλιστική βιβλιοθήκη που απλοποιεί σημαντικά τη διαδικασία δημιουργίας ενός server και ενώ έχει αρκετές δυνατότητες, με τη χρήση προσθέτουν μπορούν να αυξηθούν οι δυνατότητές της. Ένα χαρακτηριστικό τέτοιο πρόσθετο που χρησιμοποιήθηκε είναι το multer, ένα Middleware που χρησιμοποιείται κυρίως για τον χειρισμό αρχείων που αποστέλλονται προς τον server από τον client.

Οι διάφορες ενέργειες που κάνει το Middleware περιγράφονται σε συναρτήσεις που ακούνε σε διάφορα endpoints. Για παράδειγμα, όταν ο client ζητήσει την εγγραφή ενός χρήστη, εκτελείται αυτή τη συνάρτηση:

```
app.post('/requestSignUp', jsonParser, async (req, res) => {
  await signUpUser(req.body);
  const  userDataResponse = await loginUser({email: req.body.email, password: req.body.password})
  const {name, registryNumber, semester, email, uuid} = userDataResponse;

  if (userDataResponse) {
    res.json({ success: true, payload: { name, registryNumber, semester, email, uuid } })
  } else {
    res.json({ success: false, errorMsg: 'Something went wrong :(' })
  }
});
```

Εικόνα 11 - Endpoint που χειρίζεται τις κλήσεις εγγραφής χρήστη

```
const signUpUser = async({ email, password, semester, registryNumber, name, schoolCode }) => {
  const uniqueId = uuid();
  const user = new User(email, password, semester, name, registryNumber, uniqueId, schoolCode)
  const res = await USERS_COLL.doc(registryNumber).set({ ...user });
  return res;
}
```

Εικόνα 12 - Η συνάρτηση που κάνει την εγγραφή του χρήστη στη βάση

```
const loginUser = async ({ email, password }) => {
  const dbRef = USERS_COLL.where('email', '==', email).where('password', '==', password);
  const snapshot = await dbRef.get();
  let user;

  snapshot.forEach(doc => {
    user = doc.data()
  });
  return user;
```

Εικόνα 13 - Η συνάρτηση που χειρίζεται τις κλήσεις για σύνδεση

Πιο αναλυτικά, όταν ο server λάβει μια κλήση τύπου POST με endpoint το ”/requestSignUp”, θα ενεργοποιηθεί η παραπάνω συνάρτηση. Θα κληθεί η συνάρητηση signUpUser από το Firestore Wrapper . Θα πραγματοποιηθεί κλήση στη βάση για την εγγραφή του χρήστη και αμέσως μετά θα πραγματοποιηθεί η αντίστοιχη κλήση για σύνδεση. Αν οι κλήσεις για εγγραφή και σύνδεση ολοκληρωθούν επιτυχώς, αποστέλλεται στον client ένα πακέτο δεδομένων στο οποίο περιέχονται όλα τα δεδομένα του χρήστη τα οποία θα εμφανίσει το Front-End, διαφορετικά αποστέλλεται ένα μήνυμα λάθους.

3.2.6 Δομή των οντοτήτων στη βάση της εφαρμογής

Η βάση δεδομένων στο κορυφαίο επίπεδο διαθέτει δύο συλλογές. Τους χρήστες (users) και τις σχολές (schools).

3.2.7 Συλλογή: Χρήστες

Η συλλογή χρήστες είναι απλή. Διαθέτει ένα επίπεδο με documents, το καθένα εκ των οποίων αντιστοιχεί σε ένα χρήστη της εφαρμογής, φοιτητή ή διαχειριστή.

Ο κάθε χρήστης αποτελείται από ένα ζεύγος key-value. Ως key χρησιμοποιείται ο Αριθμός Μητρώου, αν και κρίθηκε καλή ιδέα μελλοντικά να αντικατασταθεί με ένα uuid (universally unique identifier), για λόγους ασφαλείας κυρίως. Το value αποτελείται από μια λίστα με ιδιότητες.

Πίνακας 1 - Σχήμα Χρήστη

Όνομα ιδιότητας	Τύπος ιδιότητας
Email	string
Name	string
Password	String
RegisteredLessons (Μαθήματα που έχει εγγραφεί για το εξάμηνο ο φοιτητής)	List
RegistryNumber (αριθμός μητρώου τμήματος)	string
SchoolCode (κωδικός σχολής στη βάση)	string
SelectedLessons (μαθήματα στο πρόγραμμά του ο φοιτητή)	List
Semester	String | number (1-8)
UserType	String (student | admin)
uuid	string

Εικόνα 14 - Χρήστης στη βάση δεδομένων

3.2.8 Συλλογή: Σχολές

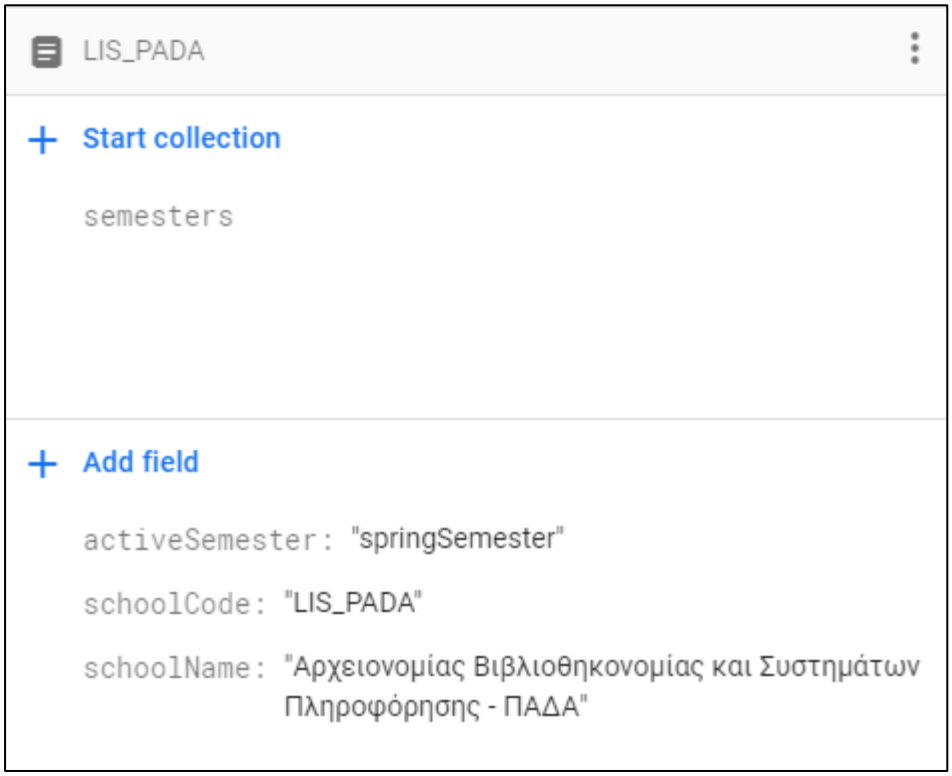
Η συλλογή Σχολές είναι αρκετά πιο σύνθετη από τους Χρήστες. Στο πρώτο της επίπεδο διαθέτει από ένα document για κάθε σχολή που είναι εγγεγραμμένη με αναγνωριστικό τον κωδικό της σχολής, τον ίδιο κωδικό που περιέχουν στις εγγραφές τους και οι χρήστες ώστε να γίνεται η συσχέτιση.

3.2.8.1 Σχήμα Σχολής

Κάθε document-σχολή έχει τρία πεδία. Την ενεργή περίοδο (activeSemester) που μπορεί να είναι είτε “springSemester” είτε “winterSemester”, τον κωδικό της σχολής και το όνομα της σχολής, το οποίο μπορεί να καθορίζεται από τον διαχειριστή. Μαζί με τα πεδία, το κάθε document-σχολή περιέχει μια υπό-συλλογή από documents που αντιστοιχούν στα δύο είδη περιόδων, εαρινή και χειμερινή.

Πίνακας 2 - Σχήμα Σχολής

Όνομα πεδίου	Τύπος
schoolName	string
schoolCode	string
semesters	Object
activeSemester	string



Εικόνα 15 - Σχολή στη βάση δεδομένων

3.2.8.2 Σχήματα Περιόδου και Μαθήματος

Κάθε περίοδος έχει μια λίστα από πεδία, τα οποία περιγράφουν τα μαθήματα του εξαμήνου. Κάθε πεδίο έχει ως αναγνωριστικό του το όνομα του μαθήματος και ως τιμή μια δομή JSON σε μορφή string λόγω τεχνικών περιορισμών. Η συγκεκριμένη δομή όταν μεταφερθεί στο Middle Layer, ώστε να διακινηθεί στο Front-End, μετατρέπεται από string σε JavaScript Object.

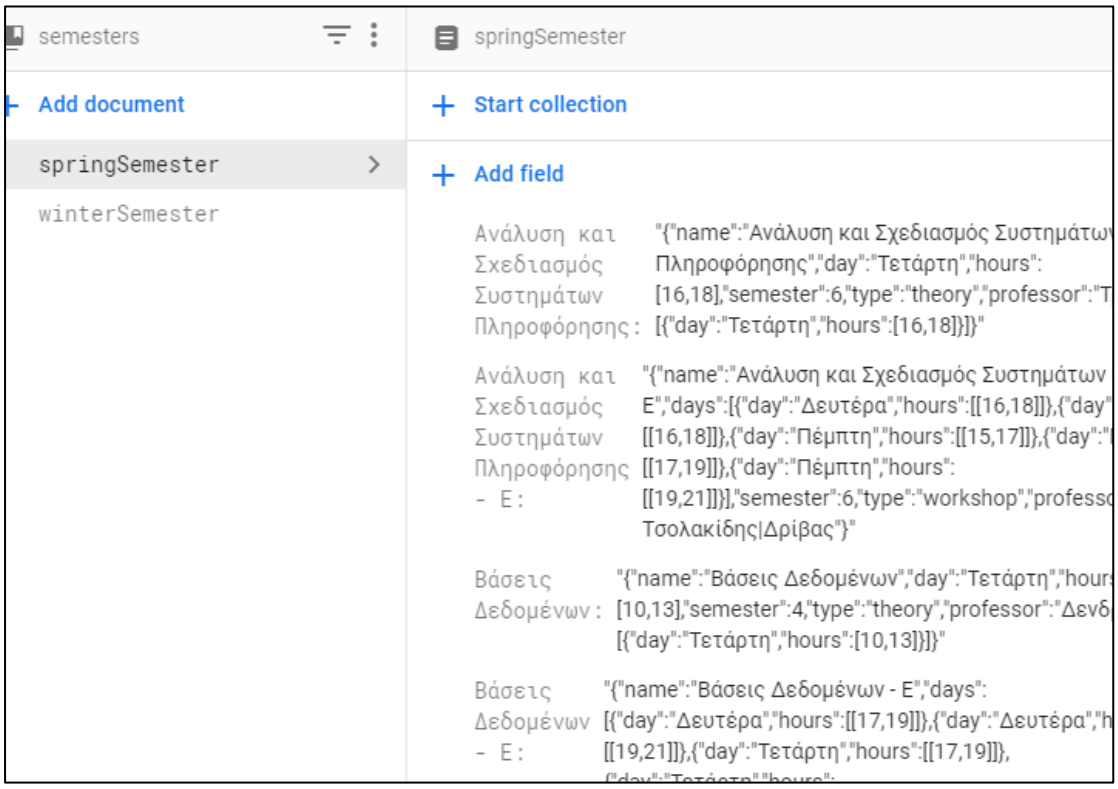
Πίνακας 3 - Σχήμα Περιόδου

Όνομα πεδίου	Τύπος
lessons	list
semesterType	string
schoolCode	string

Πίνακας 4 - Σχήμα Μαθήματος

Όνομα πεδίου	Τύπος
name	string
semester	number
type	string
professor	string
days	List<{ day: string, hours: [number] }>

Στιγμιότυπο οθόνης από το γραφικό περιβάλλον του Firestore, στο οποίο φαίνεται μια αποθηκευμένη εαρινή περίοδος με τα μαθήματα που περιέχει:



Εικόνα 16 - Περίοδος (εαρινή) στη βάση δεδομένων

3.2.9 Σχέσεις οντοτήτων

Μια σχολή περιέχει δύο τύπους περιόδων, εαρινή και χειμερινή. Κάθε περίοδος συνδέεται με τα αντίστοιχα εξάμηνα της, η χειμερινή τα εξάμηνα 1, 3, 5, 7 και η εαρινή τα 2, 4, 6, 8.

Το κάθε εξάμηνο συνδέεται με την αντίστοιχη περίοδο και περιέχει μια σειρά από ιδιότητες. Η σημαντικότερη ιδιότητά του είναι η λίστα με τα μαθήματα.

Το κάθε μάθημα συνδέεται με το αντίστοιχο εξάμηνό του και περιέχει μια σειρά από ιδιότητες. Η βασικότερη, από πλευράς λειτουργικότητας της εφαρμογής είναι η λίστα με τις ώρες και μέρες διεξαγωγής των μαθημάτων.

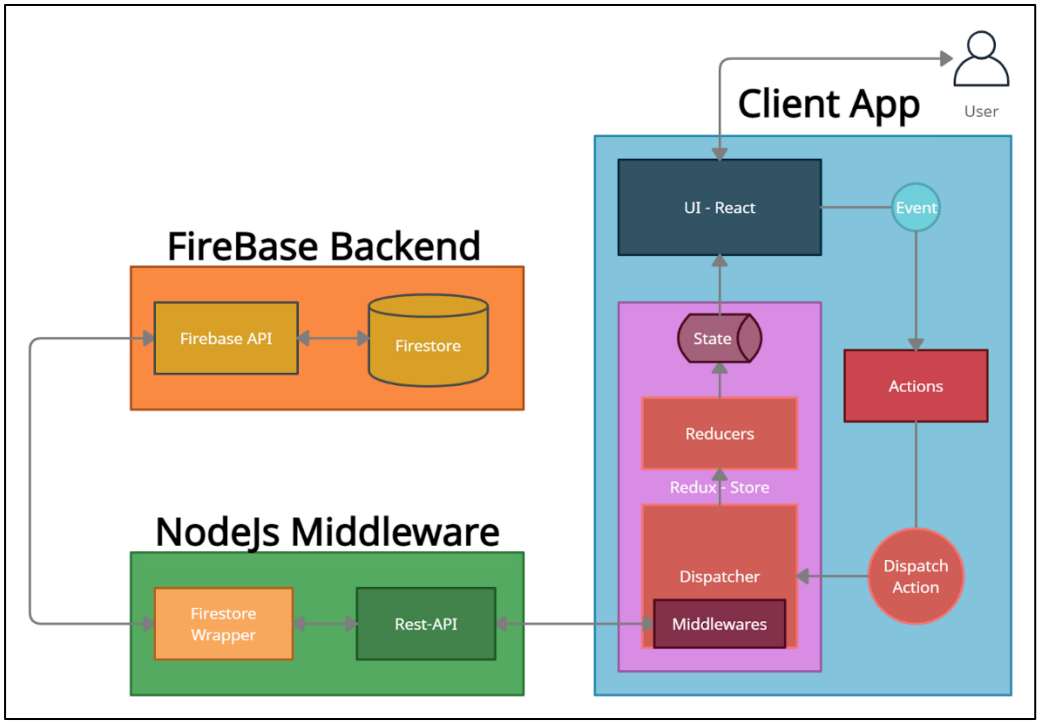
Οι χρήστες χωρίζονται σε φοιτητές και διαχειριστές. Ο διαχειριστής αποτελεί υπερσύνολο του φοιτητή, με κάποιες επιπλέον ιδιότητες που τον καθιστούν ικανό να έχει πρόσβαση στο διαχειριστικό μέρος της εφαρμογής. Είναι σημαντικό να αναφερθεί ότι ένας χρήστης μπορεί να γίνει διαχειριστής μόνο με απευθείας τροποποίηση του user type του στη βάση για λόγους

ασφαλείας.

Οι χρήστης συνδέεται με μια σχολή μέσω της φόρμας εγγραφής και μέσω αυτής της σύνδεσης, αναλόγως το εξάμηνο που είναι ενεργό, λαμβάνει στον client μια λίστα με όλα τα διαθέσιμα μαθήματα του εξαμήνου και κάνοντας τις επιλογές που επιθυμεί, αποτυπώνεται στην εγγραφή του στη βάση μια λίστα με επιλογές του “δηλωμένων” του μαθημάτων.

Στη συνέχεια, μέσω της κύριας οθόνης δημιουργεί το εβδομαδιαίο πρόγραμμα μαθημάτων του, το οποίο επίσης αποτυπώνεται στην εγγραφή του στη βάση σε μια συγκεκριμένη δομή, η οποία την επόμενη φορά που θα συνδεθεί, ανακτάται από τον client και παρουσιάζεται έτοιμη στον πίνακα.

Οι παραπάνω λειτουργίες είναι διαθέσιμες και στους διαχειριστές, με τη διαφορά ότι οι διαχειριστές βλέπουν και την οθόνη «Διαχείριση». Στη συγκεκριμένη οθόνη, για τις ανάγκες του MVP, έχουν τη δυνατότητα να ενημερώσουν τα εξάμηνα της σχολής ανεβάζοντας ένα νέο πρόγραμμα σε μορφή JSON, το οποίο παράγουν οι ίδιες οι σχολές και έχει πολύ συγκεκριμένη δομή.

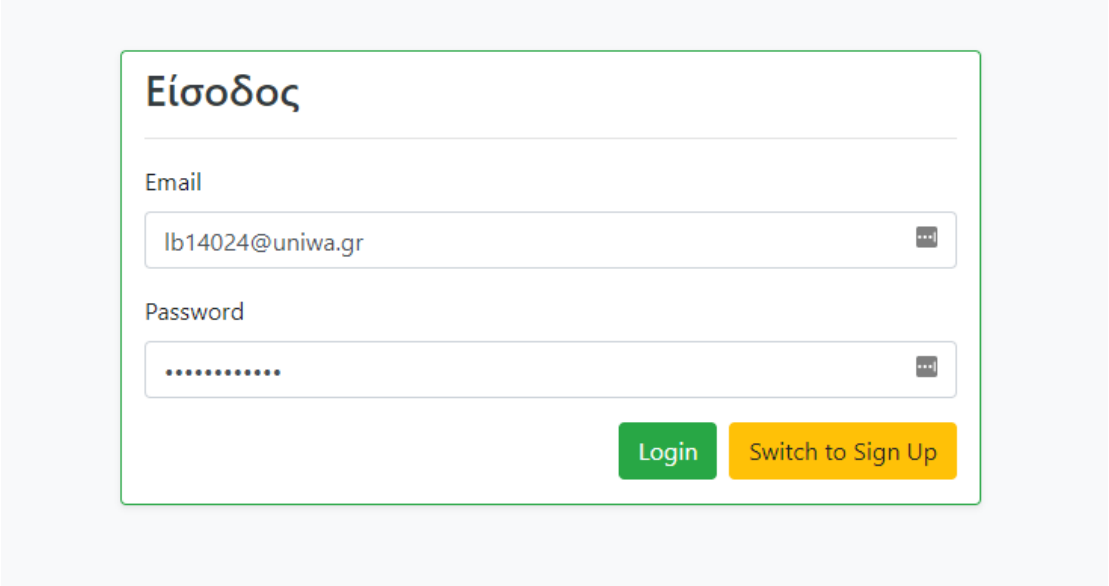


Εικόνα 17 - Γενική Αρχιτεκτονική Εφαρμογής

3.3 Παραδείγματα Χρήσης της Εφαρμογής

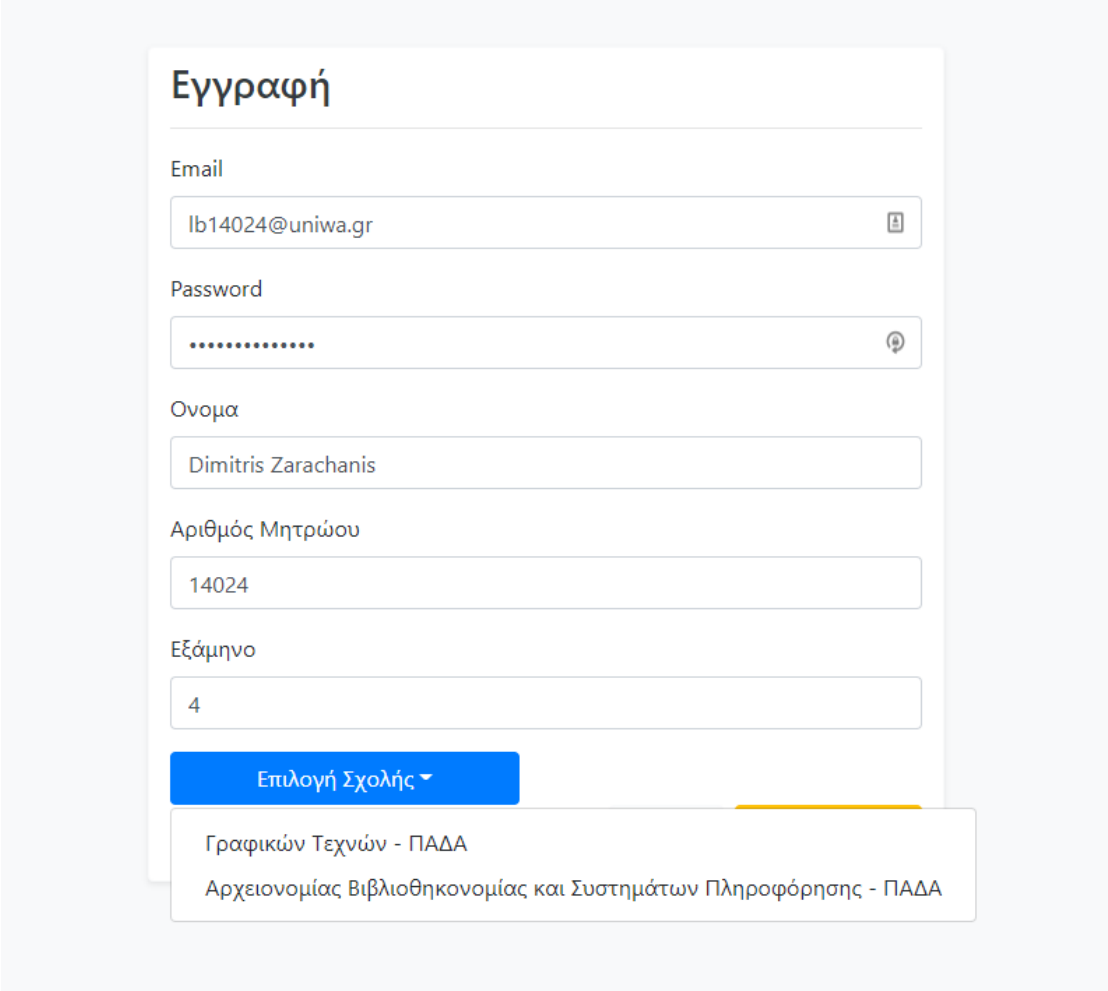
Στην παρακάτω ενότητα παρουσιάζεται μέσω στιγμιότυπων οθόνης μια τυπική ροή χρήστης της εφαρμογής από ένα χρήστη.

Μόλις φορτωθεί η εφαρμογή ο χρήστης βλέπει την οθόνη σύνδεσης ή εγγραφής. Αν έχει δημιουργήσει ήδη λογαριασμό, εισάγει τα στοιχεία του και συνδέεται στην εφαρμογή.



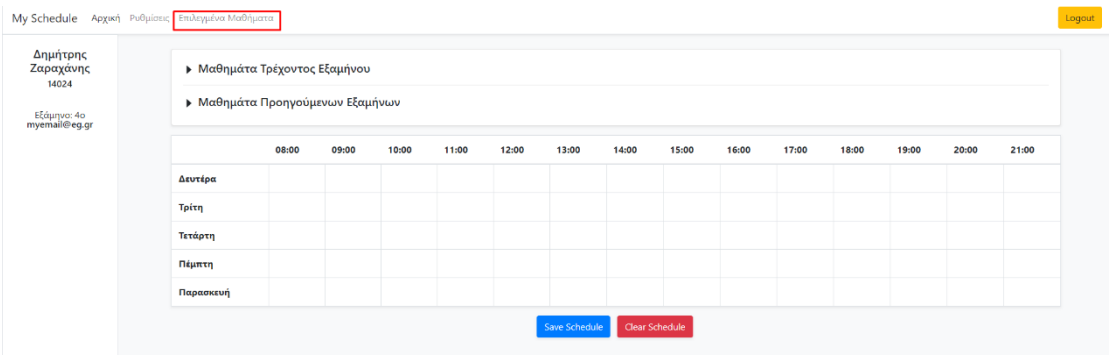
Εικόνα 18 - Φόρμα Σύνδεσης

Αν δεν έχει δημιουργήσει ακόμη λογαριασμό τότε πατώντας το κίτρινο κουμπί μεταβαίνει στη φόρμα εγγραφής. Αφού τη συμπληρώσει, την υποβάλει και εφόσον ολοκληρωθεί επιτυχώς όλη η διαδικασία τότε συνδέεται αυτόματα στην εφαρμογή με τα στοιχεία που κατέθεσε.



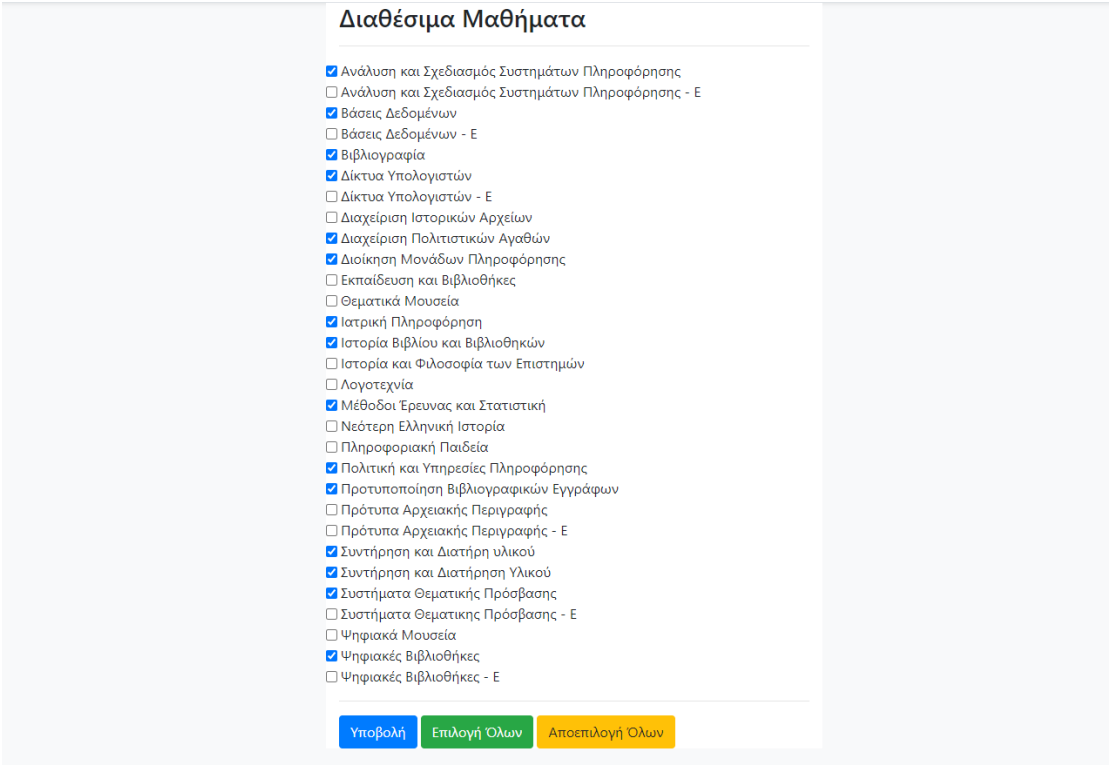
Εικόνα 19 - Φόρμα Εγγραφής

Αν συνδέεται για πρώτη φορά στην εφαρμογή ή δεν έχει επιλέξει ακόμα τα μαθήματα του εξαμήνου που έχει γραφτεί θα πρέπει να πλοηγηθεί στην οθόνη “Επιλεγμένα μαθήματα”.



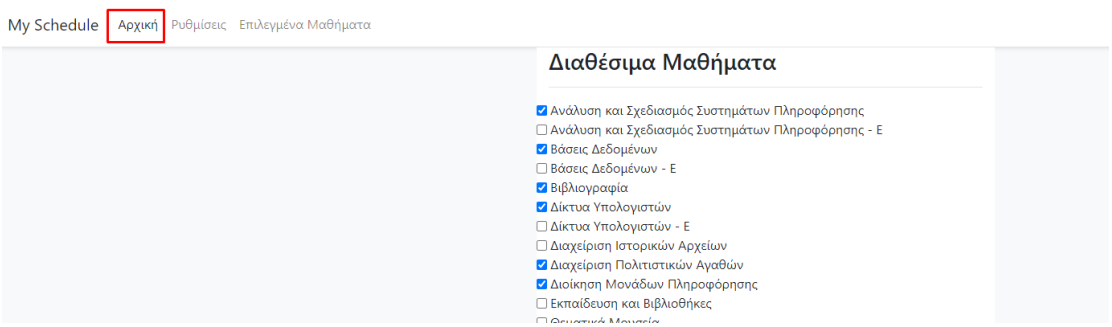
Εικόνα 20- Αρχική Οθόνη Εφαρμογής

Από τη συγκεκριμένη λίστα όλων των μαθημάτων της περιόδου πρέπει να επιλέξει τα μαθήματα στα οποία έχει εγγραφεί.



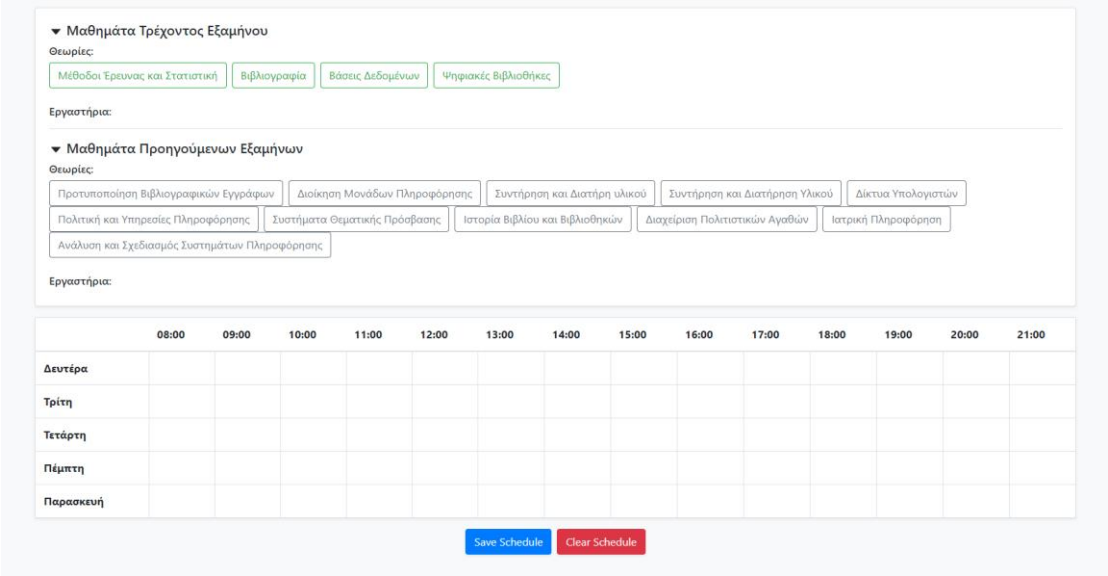
Εικόνα 21 - Οθόνη Επιλογές Εγγεγραμμένων Μαθημάτων

Αφού τα υποβάλει, πρέπει να πλοηγηθεί στην “Αρχική” οθόνη.



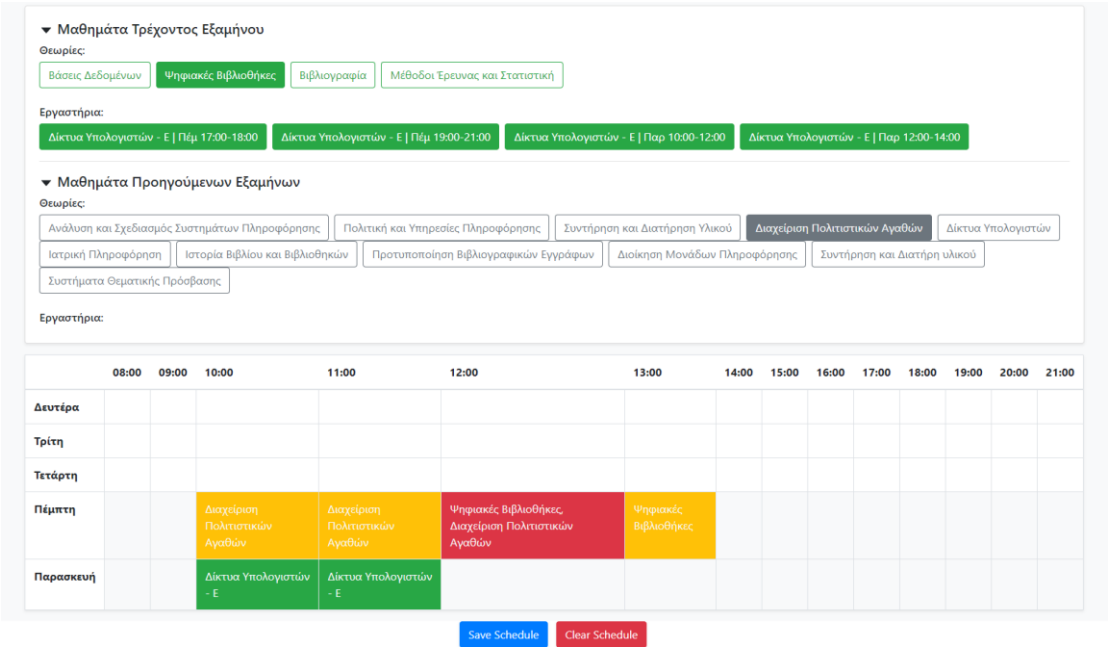
Εικόνα 22 - Επιστροφή στην Αρχική Οθόνη

Εφόσον έχουν επιλεχθεί μαθήματα, εμφανίζονται στην αρχική πλέον οι διάφορες επιλογές σε μορφή κουμπιών. Με πράσινο χρώμα είναι τα μαθήματα του τρέχοντος εξαμήνου και με γκρι των προηγούμενων ή επόμενων εξαμήνων.



Εικόνα 23 - Αρχική Οθόνη με διαθέσιμα μαθήματα

Αφού κάνει τις επιλογές μαθημάτων και ωρών που επιθυμεί στα εργαστήρια, μπορεί να δει άμεσα πως διαμορφώνεται το πρόγραμμά του. Με πράσινο χρώμα εμφανίζονται τα μαθήματα που δεν έχουν συμπτώσεις στις ώρες τους, ενώ με κόκκινο και κίτρινο όσα έχουν συμπτώσεις, έτσι ώστε να μπορεί να τροποποιήσει τις επιλογές του όσο το δυνατόν καλύτερα.



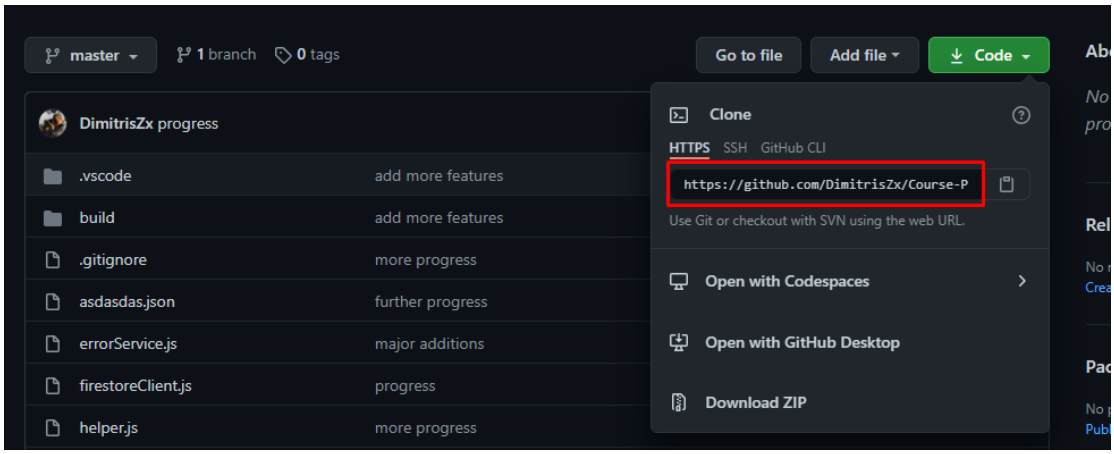
Εικόνα 24 - Συμπληρωμένος Πίνακας Μαθημάτων

Τέλος, όταν το πρόγραμμα έχει διαμορφωθεί ικανοποιητικά, ο χρήστης πατάει το κουμπί “Save Schedule” για να αποθηκεύσει στη βάση τις προτιμήσεις του, έτσι ώστε όταν συνδεθεί στο μέλλον να βρει ακριβώς τις ίδιες επιλογές μαθημάτων και ωρών.

3.4 Εγκατάσταση της εφαρμογής

Για να εγκατασταθεί η εφαρμογή τοπικά είναι απαραίτητο το αποθετήριο του κώδικα του Middle Layer. Ο σύνδεσμος προς το αποθετήριο βρίσκεται στο τέλος της εργασίας.

Εφόσον είναι εγκατεστημένο το Git στον υπολογιστή, αντιγράφεται ο παρακάτω σύνδεσμος από το Github και σε τερματικό με την εντολή “git clone <copied repo name>”, αντιγράφεται ολόκληρος ο κώδικας του Middle Layer της εφαρμογής.



Εικόνα 25 - Github - Αποθετήριο Middle Layer

```
dimit@DESKTOP-PFDCBPO MINGW64 ~
$ git clone https://github.com/DimitrisZx/Course-Planner-Node.git
```

Εικόνα 26 - Αντιγραφή του αποθετηρίου στον υπολογιστή

Για να τρέξει η εφαρμογή απαιτείται να έχει εγκατασταθεί στον υπολογιστή μια πρόσφατη έκδοση του NodeJS. Ένας γρήγορος τρόπος επιβεβαίωσης είναι ο εξής:

```
dimit@DESKTOP-PFDCBPO MINGW64 ~/Desktop/coursePlannerBackend (master)
$ node -v
v12.13.1
```

Εικόνα 27 - Έλεγχος έκδοσης/ύπαρξης NodeJS Runtime

Αφού τα παραπάνω ολοκληρωθούν επιτυχώς, πρέπει να εγκατασταθούν οι απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή. Αφού ο χρήστης πλοηγηθεί μέσω του terminal στον φάκελο που βρίσκεται η εφαρμογή, πρέπει να τρέξει την εξής εντολή:

```
dimit@DESKTOP-PFDCBPO MINGW64 ~/Desktop/coursePlannerBackend (master)
$ npm install
```

Εικόνα 28 - Εγκατάτασταση των dependencies του Middle Layer

Όταν εγκατασταθούν οι απαραίτητες βιβλιοθήκες η εφαρμογή μπορεί να εκκινήσει με την εντολή:

```
dimit@DESKTOP-PFDCBPO MINGW64 ~/Desktop/coursePlannerBackend (master)
$ node main
```

Εικόνα 29 - Έναρξη λειτουργίας Middle Layer

Εναλλακτικά, άμα πρέπει να γίνει κάποια τροποποίηση, καλό θα ήταν να χρησιμοποιηθεί μια παραλλαγή της παραπάνω εντολής, η οποία μόλις δει κάποια τροποποίηση ξανατρέχει τον server με τις νέες αλλαγές.

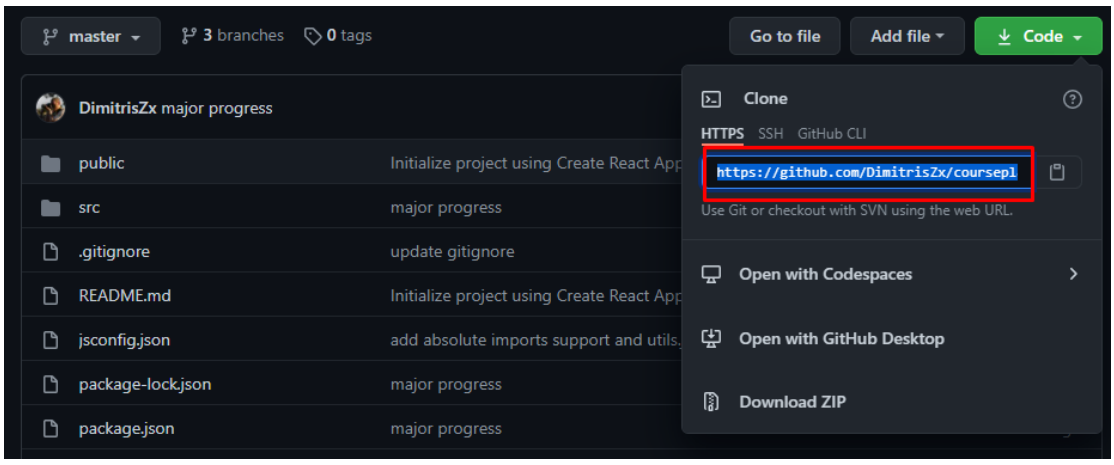
```
dimit@DESKTOP-PFDCBPO MINGW64 ~/Desktop/coursePlannerBackend (master)
$ nodemon main
```

Εικόνα 30 - Έναρξη λειτουργίας Middle Layer σε λειτουργία παρακολούθησης αλλαγών

Ανοίγοντας έναν browser, ο χρήστης αν πλοηγηθεί στη θύρα που έχει οριστεί να απαντάει ο server θα λάβει την Front-End εφαρμογή, ώστε να μπορεί να τη χρησιμοποιήσει.

3.4.1 Ανάπτυξη - Συντήρηση κώδικα για το Front-End

Για να αναπτυχθεί περεταίρω ο κώδικας του Front-End θα πρέπει να ακολουθηθούν τα αντίστοιχα βήματα εγκατάστασης με την server. Από το Github αντιγράφεται το link του αποθετηρίου:

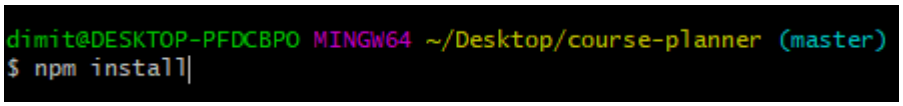


Εικόνα 31 - Github - Αποθετήριο Front-end κώδικα

Και αντιγράφεται τοπικά στον υπολογιστή με την εντολή:

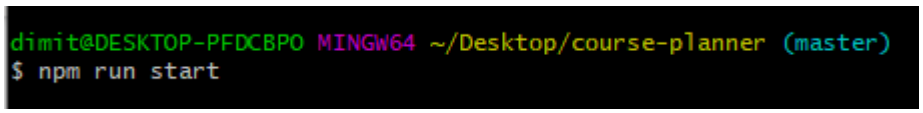
Εικόνα 32 - Αντιγραφή του Front-End περιβάλλοντος στον υπολογιστή

Χρειάζεται να εγκατασταθούν οι απαραίτητες βιβλιοθήκες με την εντολή:



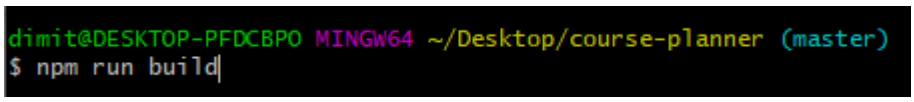
Εικόνα 33 - Εγκατάσταση των dependencies

Και για να αρχίσει να λειτουργεί ο τοπικός server που είναι απαραίτητος για την ανάπτυξη του κώδικα χρειάζεται η εντολή:



Εικόνα 34 - Εκκίνηση του Front-End development server

Εφόσον γίνουν οι απαραίτητες αλλαγές, με την εντολή:



Εικόνα 35 - Διαδικασία παραγωγής static files για διαμοιρασμό

Παράγονται τα static files στον φάκελο build, τα οποία θα μοιράζει ο server στους χρήστες. Το περιεχόμενο του φακέλου build θα πρέπει να μεταφερθεί στον αντίστοιχο φάκελο build που βρίσκεται ExpressJS server.

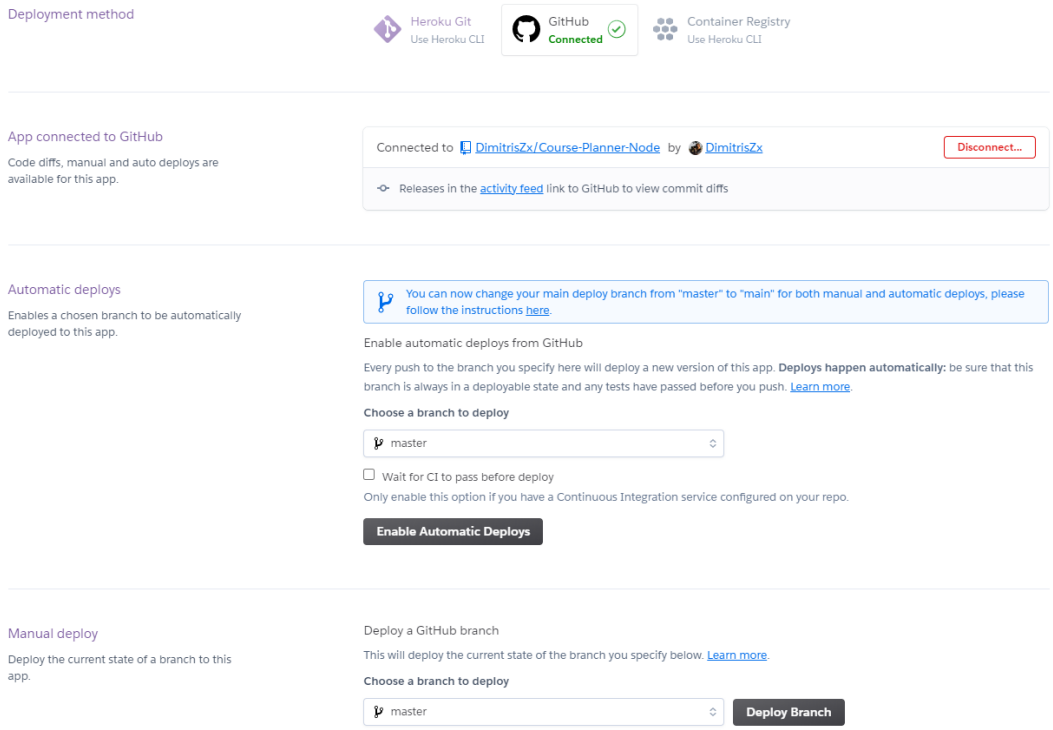
3.5 Φιλοξενία της Εφαρμογής

Η εφαρμογή, κατά τον χρόνο συγγραφής της παρούσας εργασίας, στην πλατφόρμα Heroku.

Η Heroku είναι μια cloud πλατφόρμα η οποία φιλοξενεί διαδικτυακές εφαρμογές. Εκτός από τα πληρωμένα πακέτα που προσφέρει, έχει και τη δυνατότητα δωρεάν φιλοξενίας μιας εφαρμογής, με περιορισμούς, όπως το ότι αν δε χρησιμοποιείται η εφαρμογή για αρκετή ώρα μπαίνει σε κατάσταση αναμονής και θα χρειαστεί περισσότερη ώρα για να ανταποκριθεί την επόμενη φορά που θα κληθεί.

Παρόλα αυτά, κρίθηκε ότι αποτελεί ιδανικό περιβάλλον φιλοξενίας της παρούσας εφαρμογής, καθώς είναι δωρεάν και αρκετά απλή στη διαχείριση. Ο κώδικας της εφαρμογής μεταφέρεται μέσω του Github αποθετηρίου που αναφέρεται παραπάνω. Μέσω του

γραφικού περιβάλλοντος της Heroku, συνδέεται ο αντίστοιχος λογαριασμός του Github που περιέχει το σχετικό αποθετήριο και με ένα κουμπί γίνεται το deployment και η εγκατάσταση. Αμέσως παρέχεται ένας υπερσύνδεσμος με τη διαδικτυακή διεύθυνση της εφαρμογής και υπάρχει άμεση πρόσβαση.



Εικόνα 36 - Στιγμιότυπο από το γραφικό περιβάλλον της Heroku

Κεφάλαιο 4. Συμπεράσματα – Μελλοντικές επεκτάσεις

4.1 Συμπεράσματα

Κατά την εκπόνηση της συγκεκριμένης εργασίες, υπήρξε τριβή με πολλές σύγχρονες τεχνολογίες ανάπτυξης διαδικτυακών εφαρμογών σε όλα τα επίπεδα, από τη βάση δεδομένων μέχρι την γραφική εφαρμογή χρήστη. Λόγω του πλήθους των χαρακτηριστικών που έπρεπε να αναπτυχθούν και του περιορισμένου χρόνου και τεχνογνωσίας αποφασίστηκε να γίνει μια προσπάθεια ανάπτυξης μια εφαρμογής που εκτελεί ένα βασικό σκοπό από την αρχή μέχρι το τέλος, χωρίς να αναπτυχθούν διεξοδικώς όλες οι πιθανές πτυχές της.

Αποκτήθηκε πολλή τεχνογνωσία σε όλο το εύρος της ανάπτυξης λογισμικού που περιγράφεται παραπάνω και μέσω αρκετών λαθών αποκτήθηκε παραπάνω εμπειρία σε θέματα που αφορούν διαχείριση έργων, οργάνωση του κώδικα, σχεδιασμού, ανάλυσης και αρχιτεκτονικής λογισμικού, όπως και περιγραφής του σε κείμενο.

Φυσικά, παράχθηκε ένα λογισμικό που μπορεί να χρησιμοποιηθεί και με τις κατάλληλες επεκτάσεις που περιγράφονται και παρακάτω είναι δυνατόν να αξιοποιηθεί από περισσότερους χρήστες.

4.2 Αξιοποίηση

Το αποτέλεσμα της πτυχιακής εργασίας, η διαδικτυακή εφαρμογή, βρίσκεται σε ένα δημόσιο αποθετήριο στο Github. Έτσι δίνεται η ευκαιρία σε όποιον επιθυμεί να μπορέσει να συνεισφέρει σε αυτή ή να αντιγράψει τον πηγαίο κώδικα και να την επεκτείνει μόνος του. Επιπλέον η εφαρμογή αυτή δύναται να συνεχίσει να αναπτύσσεται και μετά το πέρας της εργασίας αυτής και τελικά να λειτουργήσει επίσημα στο διαδίκτυο.

4.3 Μελλοντικές βελτιώσεις/επεκτάσεις

Λόγω ποικίλων περιορισμών, όπως χρόνου και τεχνικής κατάρτισης, έγιναν κάποιες επιλογές στην αρχή της ανάπτυξης που μπορούν να βελτιωθούν μελλοντικά.

Μια πολύ σημαντική τροποποίηση στο Middleware θα ήταν ο κώδικας να μεταφερθεί σε ένα περισσότερο στιβαρό framework όπως το NestJS, το οποίο είναι επίσης βασισμένο σε Node και Express, αλλά παρέχει μια αυστηρή οργάνωση και κάποιες καθιερωμένες πρακτικές που διευκολύνουν την ανάπτυξη του κώδικα. Επιπλέον, με την υποστήριξη που έχει για την TypeScript, ο κώδικας μπορεί να γραφτεί πιο καθαρά με την εισαγωγή τύπων και περισσότερων εργαλείων αντικειμενοστραφούς προγραμματισμού. Αντίστοιχα θα βελτίωνε την ανάπτυξη και του Front-End App η εισαγωγή της TypeScript.

Ένα σημαντικό πρόβλημα που εκκρεμεί είναι μια πιο σωστή, από πλευράς ασφάλειας και οργάνωσης, υλοποίηση του συστήματος εγγραφής και αυθεντικοποίησης.

Από πλευράς λειτουργικότητας θα ήταν καλή ιδέα η λειτουργικότητα του εβδομαδιαίου προγράμματος μαθημάτων να γενικευθεί σε ένα εβδομαδιαίο πρόγραμμα δραστηριοτήτων, το οποίο μπορεί να συνδυάζει τα μαθήματα με άλλες τυχόν ασχολίες του χρήστη.

Τέλος είναι σημαντικό να βρεθεί ένας αξιόπιστος πάροχος για να φιλοξενήσει τον Express server, ώστε η εφαρμογή να μπορεί να αξιοποιηθεί από το ευρύ κοινό. Σε αυτή τη φάση της ζωής της μπορεί να λειτουργήσει μόνο τοπικά σε κάποιον υπολογιστή.

Βιβλιογραφικές Αναφορές

Agile Alliance. (2021, February 10). What is a Minimum Viable Product (MVP)? Retrieved from https://www.agilealliance.org/glossary/mvp/#q=%7E(infinite%7Efalse%7Efilters%7E(tags%7E(%7E'mvp))%7EsearchTerm%7E'%7Esort%7Efalse%7EsortDirection%7E'asc%7Epage%7E1)

Chart, Diagram & Visual Canvas Software. (2020). Retrieved from https://creately.com/

Cloud Application Platform | Heroku. (2021). Retrieved March 19, 2021, from https://www.heroku.com/

Concepts. (2020). Retrieved from https://webpack.js.org/concepts/

Databases on AWS: The Right Tool for the Right Job. (2020). Retrieved from https://aws.amazon.com/nosql/

Documentation. (2020). Retrieved from https://firebase.google.com/docs

Education, I. C. (2020, November 23). NoSQL Databases. Retrieved from https://www.ibm.com/cloud/learn/nosql-databases

Express - Node.js web application framework. (2020). Retrieved from https://expressjs.com/

Getting Started | Create React App. (2020, July 2). Retrieved from https://create-react-app.dev/docs/getting-started/

Git. (2020). Retrieved from https://git-scm.com/

GitHub: Where the world builds software. (2020). Retrieved from https://github.com/

Microsoft. (2016, April 14). Visual Studio Code - Code Editing. Redefined. Retrieved from https://code.visualstudio.com/

MongoDB. (2020). What is NoSQL? NoSQL Databases Explained. Retrieved from https://www.mongodb.com/nosql-explained

NodeJS. (2020). Retrieved from https://nodejs.org/en/about/

Otto, M. J. T. (2020). Bootstrap. Retrieved from https://getbootstrap.com/

Progressive web apps (PWAs) | MDN. (2020, December 15). Retrieved from https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

React – A JavaScript library for building user interfaces. (2020). Retrieved from https://reactjs.org/

Redux - A predictable state container for JavaScript apps. | Redux. (2020). Retrieved from https://redux.js.org/

Standards - W3C. (2020). Retrieved from https://www.w3.org/standards/

What are Progressive Web Apps? (2020). Retrieved from https://web.dev/what-are-pwas/

Παράρτημα

Σύνδεσμοι προς τα αποθετήρια του κώδικα:

Front-end: https://github.com/DimitrisZx/courseplanner

Middleware: https://github.com/DimitrisZx/Course-Planner-Node