ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και υλοποίηση Συστήματος διαχείρισης διπλωματικών εργασιών

Βεργίνης Ηλίας
Α.Μ. 171019

Εισηγητής: Χρήστος Τρούσσας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και υλοποίηση Συστήματος διαχείρισης διπλωματικών εργασιών

Βεργίνης Ηλίας
Α.Μ. 171019

Εισηγητής:

Χρήστος Τρούσσας

Εξεταστική Επιτροπή:

Χρήστος Τρούσσας
Ακριβή Κρούσκα
Παναγιώτα Τσελέντη

Christos Troussas
Digitally signed by Christos Troussas
Date: 2024.11.04 15:00:48 +02'00'

Akrivi Krouska
Digitally signed by Akrivi Krouska
Date: 2024.11.04 16:37:56 +02'00'

Panagiota Tselenti
Digitally signed by Panagiota Tselenti
Date: 2024.11.05 11:06:30 +02'00'

Ημερομηνία εξέτασης: Σεπτέμβρης 2024

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Ηλίας Βεργίνης του Παναγιώτη, με αριθμό μητρώου 171019 φοιτητής του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Βεβαιώνω ότι είμαι συγγραφέας αυτής της Διπλωματικής εργασίας και 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».

«Βεβαιώνω ότι είμαι συγγραφέας της παρούσας διπλωματικής εργασίας και ότι έχω αναφέρει ή παραπέμψει σε αυτή, ρητά και συγκεκριμένα, όλες τις πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, προτάσεων ή λέξεων, είτε αυτές μεταφέρονται επακριβώς (στο πρωτότυπο ή μεταφρασμένες) είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για την συγκεκριμένη διπλωματική εργασία»

Ο Δηλών
Ηλίας Βεργίνης

ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτα από όλα, το μεγαλύτερο ευχαριστώ πρέπει να δοθεί στην οικογένεια μου για την υποστήριξη που μου παρείχαν κατά την διάρκεια των σπουδών μου, τόσο σε ηθικό όσο και σε ψυχολογικό κομμάτι.

Επιπλέον, θα ήθελα να ευχαριστήσω τους φίλους μου, που μου παρείχαν σημαντική βοήθεια στην εκπόνηση της εργασίας ανιδιοτελώς, μέσω της τεχνογνωσίας τους.

Τέλος, σημαντική ήταν η συμβουλή και η καθοδήγηση από τον επιβλέποντα καθηγητή κ. Τρούσσα κατά την διάρκεια της υλοποίησης του συστήματος, καθώς και της συγγραφής της εργασίας.

Περίληψη

Αναγνωρίζοντας την δυσκολία που αντιμετωπίζουν στην ανάληψη και την εκπόνηση των διπλωματικών εργασιών, τόσο οι φοιτητές όσο και οι καθηγητές, αποφασίσαμε να δημιουργήσουμε μία διαδικτυακή εφαρμογή για την διευκόλυνση αυτής της διαδικασίας. Η παρούσα διπλωματική εργασία, λοιπόν, αφορά τον σχεδιασμό και την υλοποίηση ενός συστήματος διαχείρισης διπλωματικών εργασιών. Βασικός στόχος-σκοπός την εφαρμογής αυτής είναι η διευκόλυνση τόσο των φοιτητών όσο και τον καθηγητών κατά την εκπόνηση της εκάστοτε διπλωματικής εργασίας, τόσο σε λειτουργικό όσο σε επικοινωνιακό επίπεδο, αποτελώντας μία πλήρη λύση διαχείρισης και παρακολούθησης σε ακαδημαϊκό επίπεδο.

Παρακάτω θα αναλυθούν τα βήματα που ακολουθήθηκαν για την επίτευξη αυτού του στόχου. Πιο αναλυτικά θα αναλυθεί η έρευνα που έγινε σε διάφορα συστήματα διαχείρησης σε εκπαιδευτικό επίπεδο κυρίως, προκειμένου να επιτευχθεί η σωστή δομή στο συγκεκριμένο σύστημα. Επιπλέον, θα γίνει εκτενής περιγραφή της υλοποίησης του συστήματος σε θεωρητικό επίπεδο και στις δυσκολίες που αντιμετωπίστηκαν.

Τέλος, θα γίνει συζήτηση σχετικά με την επιλογή των τεχνολογιών ReactJS για τον πελάτη και NodeJS για τον διακομιστή, παρέχοντας υψηλή απόδοση, ευελιξία και αξιοπιστία στην εφαρμογή. Επιπροσθέτων, θα γίνει ανάλυση για τους λόγους επιλογής αυτών των τεχνολογιών έναντι διάφορων τεχνολογιών, που είναι ευρέως γνωστές.

Abstract

Recognizing the difficulty faced by both students and professors in undertaking and completing diploma theses, we decided to create a web application to facilitate this process. This thesis, therefore, focuses on the design and implementation of a diploma thesis management system. The primary goal of this application is to facilitate both students and professors in the completion of each respective diploma thesis, both functionally and communicatively, providing a comprehensive solution for management and monitoring at an academic level.

Below, the steps taken to achieve this goal will be analyzed. More specifically, the research conducted on various management systems at the educational level will be discussed, mainly to achieve the correct structure in the specific system. Additionally, there will be an extensive description of the system's implementation at a theoretical level and the challenges faced.

Finally, there will be a discussion regarding the choice of technologies, namely ReactJS for the client and NodeJS for the server, providing high performance, flexibility, and reliability to the application. Furthermore, an analysis will be conducted on the reasons for choosing these technologies over various others that are widely known.

Λέξεις Κλειδιά: Σύστημα Διαχείρισης, Διπλωματική Εργασία, ReactJS, NodeJS, MySQL.

Keywords: Management System, Diploma Thesis, ReactJS, NodeJS, MySQL

Περιεχόμενα

1.Εισαγωγή 14

1.1 Εξέλιξη τεχνολογίας 14

1.2 Σκοπός και στόχος υλοποίησης 14

1.3 Δομή διπλωματικής εργασίας 15

2. Θεωρητικό υπόβαθρο 16

2.1 Τεχνολογίες Διαδικτύου 16

2.1.1 CSS Frameworks 16
2.1.2 Javascript Frameworks/Libraries 18
2.1.3 Rest API (Representational State Transfer) 21
2.1.4 Δομή Αιτήματος HTTP 22
2.1.5 Έρευνα Framework για Rest API 23

2.2 Παρόμοια Συστήματα 25

2.3 Σύνοψη Κεφαλαίου 26

3. Σχεδιασμός Εφαρμογής 27

3.1 Αλληλεπίδραση Ανθρώπου-Υπολογιστή 27

3.1.1 Βασικές αρχές UI/UX 27
3.1.2 Διαδικασία δημιουργίας UI 28
3.1.3 Τελικό UI 29

3.2 Βάση Δεδομένων 39

3.2.1 Είδη Βάσεων 39
3.2.2 Διαφορές μεταξύ SQL και NoSQL 39
3.2.3 Βασικές αρχές 40

3.3 Υλοποίηση Βάσης 41

3.3.1 Αρχικοποίηση βάσης 41
3.3.2 Μοντέλο Χρήστη 42
3.3.3 Μοντέλο Ρόλων 43
3.3.4 Μοντέλο Διπλωματικής Εργασίας 44
4.3.5 Μοντέλο Commit 45

3.3.6 Μοντέλο Σχολίων 46

3.3.7 Μοντέλο Αρχείων 47

3.4 Σχεσιακά μοντέλα 47

3.5 Τελικό αποτέλεσμα MySQL 50

3.6 Σύνοψη Κεφαλαίου 52

4. Λειτουργίες και υλοποίηση 53

4.1 Περιγραφή απαιτήσεων συστήματος 53

4.2 Αρχιτεκτονική 54

4.3 Τεχνολογία και υλοποίηση Backend 54

4.3.1 Βιβλιοθήκες Εξυπηρετητή 54
4.3.2 Δομή αρχείων Εξυπηρετητή 55
4.3.3 Middlewares 57
4.3.4 Routes 59
4.3.5 Αυθεντικοποίηση στον εξυπηρετητή 65
4.3.6 Δημιουργία Διπλωματικής και Επεξεργασία 68

4.3.7 Δημιουργία Commit 70
4.3.8 Δημιουργία Email 71
4.3.9 Δημιουργία Σχολίου 73
4.3.10 Ασφάλεια Εξυπηρετητή 74
4.4 Τεχνολογία και υλοποίηση Frontend 75
4.4.1 Βιβλιοθήκες Πελάτη 75
4.4.2 Δομή αρχείων Πελάτη 76
4.4.3 Φόρμες και Αίτηματα Πελάτη 78
4.4.4 Αυθεντικόποιηση Πελάτη 84
4.4.5 State Management 85
4.5 Λειτουργία ζωντανής συνομιλίας 88
4.6 Ανάρτηση Αρχείων 92
4.7 Σύνοψη Κεφαλαίου 95
5. Επίλογος 96
5.1 Σύνοψη εφαρμογής 96
5.2 Αδυναμίες Συστήματος και Μελλοντικές βελτιώσεις 96
6.Βιβλιογραφία 98

Καταλογος Εικόνων

Εικόνα 1: HTTP αίτημα 22
Εικόνα 2: Λόγοι κακής διαδικτυακής εφαρμογής 28
Εικόνα 3: Εργαλεία ανάπτυξης UI/UX 29
Εικόνα 4: Σελίδα Σύνδεσης 29
Εικόνα 5: Σελίδα εγγραφής χρήστη 30
Εικόνα 6: Αρχική σελίδα εφαρμογής 31
Εικόνα 7: Αρχική σελίδα χωρίς δεδομένα 32
Εικόνα 8: Φόρμα δημιουργία διπλωματικής εργασίας 32
Εικόνα 9: Ειδοποίηση θετικού αποτελέσματος 33
Εικόνα 10: Λίστα commits 33
Εικόνα 11: Πίνακας Κινητών 34
Εικόνα 12: Πίνακας κινητών επεκτείνομενος 34
Εικόνα 13: Φόρμα δημιουργίας commit 35
Εικόνα 14: Συμπληρωμένη φόρμα commit 35
Εικόνα 15: Θετική ειδοποίηση δημιουργίας commit 36
Εικόνα 16: Ζωντανή επικοινωνία 36
Εικόνα 17: Δημιουργία ηλεκτρονικού μηνύματος 37
Εικόνα 18: Σελίδα ρυθμίσεων 37
Εικόνα 19: Σελίδα διαχειριστή, επεξεργασία χρηστών 38
Εικόνα 20: Προτιμήσεις βάσεων 39
Εικόνα 21: Many-to-Many σχέση 40
Εικόνα 22: Μεταβλητές Περιβάλλοντος 41
Εικόνα 23: Αρχικοποίηση δεδομένων βάσης 41
Εικόνα 24: Αρχικοποίηση βιβλιοθήκης sequelize 42
Εικόνα 25: Μοντέλο Χρήστη 42
Εικόνα 26: Μοντέλο Ρόλων 43
Εικόνα 27: Μοντέλο διπλωματικής Εργασίας 44
Εικόνα 28: Μοντέλο Commit 45
Εικόνα 29: Μοντέλο σχολίων 46
Εικόνα 30: Μοντέλο αρχείων 47
Εικόνα 31: Σχεσιακά μοντέλα 48
Εικόνα 32: Σχεσιακά μοντέλα 2 49
Εικόνα 33: Ρόλοι συστήματος 53
Εικόνα 34: Αρχιτεκτονική Πελάτη-Εξυπηρετητή 54
Εικόνα 35: Package.json εξυπηρετητή 55
Εικόνα 36: Δομή αρχείων εξυπηρετητή 56
Εικόνα 37: Middleware αυθεντικοποίησης 57
Εικόνα 38: Middleware ρόλου διαχειριστή 58
Εικόνα 39: Middleware ρόλου καθηγητή 58
Εικόνα 40: Έλεγχος διπλότυπου email 59
Εικόνα 41: Middleware ύπαρξης ρόλου 59
Εικόνα 42: Route εγγραφή, μέσω διαχειριστή 60
Εικόνα 43: Route εγγραφής 60
Εικόνα 44: Route σύνδεσης 60
Εικόνα 45: Route επιστροφής χρηστών 61
Εικόνα 46: Route αλλαγής κωδικού 61

Εικόνα 47: Route αλλαγής email....61
Εικόνα 48: Route διαγραφής χρήστη....61
Εικόνα 49: Route ενημέρωση χρήστη....61
Εικόνα 50: Route ενημέρωσης προσωπικών στοιχείων χρήστη....62
Εικόνα 51: Route δημιουργίας email....62
Εικόνα 52: Route δημιουργίας διπλωματικής....62
Εικόνα 53: Route ενημέρωσης διπλωματικής....62
Εικόνα 54: Route επιστροφής διπλωματικής....63
Εικόνα 55: Route διδάσκοντα διπλωματικής....63
Εικόνα 56: Route δημιουργίας commit....63
Εικόνα 57: Route διαγραφής commit....63
Εικόνα 58: Route επιστροφής commit, με βάση τον χρήστη....63
Εικόνα 59: Route επιστροφής όλων των commit....64
Εικόνα 60: Route επιστροφής αρχείων....64
Εικόνα 61: Route δημιουργίας σχολίου....64
Εικόνα 62: Route επιστροφής σχολίων, βάση χρήστη....64
Εικόνα 63: Εγγραφή....65
Εικόνα 64: Εγγραφή στην Βάση....65
Εικόνα 65: Εγγραφή μέσω Google....66
Εικόνα 66: Σύνδεση Χρήστη....67
Εικόνα 67: Δημιουργία Διπλωματικής....68
Εικόνα 68: Ενημέρωση Διπλωματικής....69
Εικόνα 69: Ενημέρωση Διπλωματικής 2....69
Εικόνα 70: Δημιουργία commit 1....70
Εικόνα 71: Δημιουργία commit 2....71
Εικόνα 72: Transporter αντικείμενο....71
Εικόνα 73: Δημιουργία Email....72
Εικόνα 74: Δημιουργία Σχολίου 1....73
Εικόνα 75: Δημιουργία Σχολίου 2....74
Εικόνα 76: Fetch Comment....75
Εικόνα 77: .env εξυπηρετητή....76
Εικόνα 78: package.json πελάτη....77
Εικόνα 79: Δομή αρχείων πελάτη....78
Εικόνα 80: CustomInput....79
Εικόνα 81: FormLayout....80
Εικόνα 82: FormLayout 2....80
Εικόνα 83: CreateCommit....81
Εικόνα 84: CreateCommit 2....81
Εικόνα 85: CreateCommit3....82
Εικόνα 86: Axios instance....82
Εικόνα 87: Enum endpoints....83
Εικόνα 88: createCommit ασύγχρονη συνάρτηση....83
Εικόνα 89: Δεδομένα αιτήματος....84
Εικόνα 90: Απάντηση αιτήματος....84
Εικόνα 91: Redux toolkit....86
Εικόνα 92: Δημιουργία redux store....87
Εικόνα 93: Redux provider....87

Εικόνα 94: Reducer action 88
Εικόνα 95: Κλήση action 88
Εικόνα 96: Επιλογή δεδομένων redux 88
Εικόνα 97: Μοντέλο επικοινωνίας με socket 89
Εικόνα 98: Αρχικοποίηση Server 90
Εικόνα 99: Socker Listener και συναρτήσεις 90
Εικόνα 100: Υλοποίηση socket στον πελάτη 91
Εικόνα 101: Δημιουργία δωματίου UI 92
Εικόνα 102: Αποστολή μηνύματος UI 92
Εικόνα 103: Ειδοποίηση νέου μηνύματος UI 93
Εικόνα 104: Απάντηση στο μήνυμα 93
Εικόνα 105: Formdata file handling 93
Εικόνα 106: Αίτημα ανάρτηση αρχείων 94
Εικόνα 107: Υλοποιήση εξυπηρετητή αρχείων 94
Εικόνα 108: Προσκόλληση αρχείων στο commit 95
Εικόνα 109: Απάντηση αιτήματος 95
Εικόνα 110: Αποθήκευση αρχείων 96

Κατάλογος Πινάκων

Πίνακας 1 Πίνακας χρηστών50
Πίνακας 2 Πίνακας Ρόλων50
Πίνακας 3 Συνδετικός πίνακας, χρήστη -ρόλου51
Πίνακας 4 Παράδειγμα πίνακα χρήστη-ρόλου51
Πίνακας 5 Πίνακας Διπλωματικής Εργασίας51
Πίνακας 6 Σχεσιακός Πίνακας μαθητή-εργασίας51
Πίνακας 7 Πίνακας Commit52
Πίνακας 8 Πίνακας Σχολίων52
Πίνακας 9 Πίνακας αρχείων52
Πίνακας 10 Ενδιάμεσος Πίνακας σχολίου-παραλήπτη52

1.Εισαγωγή

Σε αυτό το κεφάλαιο θα αναλυθεί το αντικείμενο που πραγματεύεται η παρούσα διπλωματική εργασία. Αρχίζοντας, θα μιλήσουμε για την εξέλιξη της τεχνολογίας και πως αυτή έχει επηρεάσει την ζωή μας, κυρίως σε ακαδημαϊκό επίπεδο. Συνεχίζοντας, θα δούμε τους λόγους που οδήγησαν στην ανάγκη μίας εφαρμογής σαν και αυτή, καθώς και τα προβλήματα που μπορεί να λύσει. Τέλος, θα δοθεί μία περίληψη για την δομή της διπλωματικής εργασίας.

1.1 Εξέλιξη τεχνολογίας

Η τεχνολογία στις μέρες μας εξελίσσεται με ραγδαίους ρυθμούς, απλοποιώντας διαδικασίες στην καθημερινότητα μας, που πριν την εξέλιξη αυτή ήταν αρκετά χρονοβόρες. Χαρακτηριστικό παράδειγμα αποτελεί η ενεργή ένταξη της τεχνολογίας και του διαδικτύου στις δημόσιες υπηρεσίες. Πλέον, ο χρήστης μπορεί με μεγαλύτερη ευκολία και ταχύτητα να βρει οποιοδήποτε κρατικό έγγραφο χρειάζεται, χωρίς να είναι απαραίτητη η φυσική του παρουσία στα Κέντρα Εξυπηρέτησης Πολιτών (ΚΕΠ). Προφανώς, αυτό αποτελεί ένα από τα πολλά παραδείγματα που η τεχνολογία βοήθησε την καθημερινότητα των ανθρώπων. Η εξέλιξη αυτή δεν μπορούσε να αφήσει ανεπηρέαστη την ακαδημαική κοινότητα. Οι αλλαγές που έχουν έρθει με την πάροδο του χρόνου έχουν απλοποιήσει διαδικασίες, οι οποίες πρώτα απαιτούσαν φυσική παρουσία και ήταν χρονοβόρες. Για παράδειγμα, η δήλωση μαθημάτων πρώτα απαιτούσε την φυσική παρουσία και αρκετή αναμονή, ενώ τώρα έχει αυτοματοποιηθεί με την χρήση της πλατφόρμας, services, μειώνοντας δραστικά τόσο τον χρόνο όσο και την πιθανότητα λάθος από πλευράς του προσωπικού του Πανεπιστημίου.

1.2 Σκοπός και στόχος υλοποίησης

Η χρήση διαδικτυακών εφαρμογών, όμως χαίρει βελτιώσεων, αφού υπάρχουν ακόμα διαδικασίες που δεν έχουν την κατάλληλη εφαρμογή για την λύση του προβλήματος. Ένα από αυτά τα προβλήματα είναι και το θέμα που θα πραγματευτεί σε αυτή την Διπλωματική εργασία, πιο συγκεκριμένα η διαδικασία ανάληψης και υλοποίησης διπλωματικών εργασιών. Πρώτα από όλα, οφείλουμε να μιλήσουμε για το πως λειτουργούσε η διαδικασία αυτή μέχρι σήμερα. Ο εκάστοτε φοιτητής που επιθυμούσε να αναλάβει μία διπλωματική εργασία έπρεπε να στείλει μήνυμα ή να βρεθούνε δια ζώσης με τον καθηγητή που επιθυμεί να αναλάβει την διπλωματική του εργασία, με το θέμα που επιθυμεί. Στην συνέχεια, ο καθηγητής αξιολογεί το θέμα και βλέπει αν είναι σχετικό με το γνωστικό του επίπεδο και προχωράει στην αποδοχή ή την απόρριψη αυτής. Το στάδιο της υλοποίησης, που ακολουθεί, είναι το πιο δύσκολο με την μέχρι τώρα διαδικασία, καθώς η επικοινωνία μέσω ηλεκτρονικών μηνυμάτων γίνεται δύσκολη, ειδικά όταν ο επιβλέπων διδάσκοντας έχει να απαντήσει εκατοντάδες μηνύματα.

Επομένως, καταλήγουμε πως η χρήση μίας ξεχωριστής εφαρμογής είναι απαραίτητη. Η εφαρμογή που θα αναπτυχθεί έχει ως κύριους στόχους την καλύτερη επικοινωνία και την συνεχή επίβλεψη της εργασίας από τον διδάσκοντα. Επιπλέον, άλλο ένα προνόμιο μίας τέτοιας εφαρμογής, αποτελεί η διατήρηση ιστορικού, καθιστώντας την απώλεια δεδομένων πιο δύσκολη. Από την πλευρά του διδάσκοντα, γίνεται πιο εύκολη η διαχείριση των εργασιών που έχει αναλάβει, αφού τις έχει όλες συγκεντρωτικά, χωρίς να

χρειάζεται να ανατρέχει στα μηνύματα του, ή θα δημιουργεί μόνος του κάποια λίστα. Τέλος, η εφαρμογή παρέχει συνομιλία σε πραγματικό χρόνο, η οποία βοηθάει στην άμεση επίλυση απορίων και καλύτερη επίβλεψη.

1.3 Δομή διπλωματικής εργασίας

Στο επόμενη κεφάλαιο θα συζητηθούν παρόμοια συστήματα διαχείρισης δεδομένων, καθώς και εκπαιδευτικά συστήματα που υπάρχουν, με σκοπό την κατανόηση ενός τέτοιου συστήματος, καθώς και τις προδιαγραφές αυτού. Επιπλέον, θα αναλυθεί η μελέτη που έγινε σε διάφορες τεχνολογίες πρωτού επιλεχθούν οι κατάλληλες για το συγκεκριμένο εγχείρημα.

Στο τρίτο κεφάλαιο θα μιλήσουμε για την μεθοδολογία που ακολουθήθηκε για να υλοποιηθεί η εφαρμογή, με ιδιαίτερη έμφαση στην διεπαφή του χρήστη, την βάση δεδομένων πίσω από την εφαρμογή και γενικότερα τον κώδικα που δημιουργήθηκε.

Στο τέταρτο κεφάλαιο θα δούμε τις λειτουργίες του συστήματος εκτενέστερα, την αρχιτεκτονική του και θα γίνει μία πιο αναλυτική περιγραφή του συστήματος.

Κλείνοντας, θα δούμε συμπεράσματα και στατιστικά σχετικά με το πόσο εύχρηστο το σύστημα, και αν όντως αποτελεί μία λύση σε ένα πρόβλημα που αντιμετωπίζει η ακαδημαική κοινότητα.

2. Θεωρητικό υπόβαθρο

Σε αυτό το κεφάλαιο θα συζητηθεί η λειτουργία των διαδικτυακών εφαρμογών και οι πιο συνηθισμένες γλώσσες προγραμματισμού για την επίτευξη τέτοιων εφαρμογών. Στην συνέχεια θα δούμε μερικές από τις βασικές μεθοδολογίες ανάπτυξης συστημάτων, με τα πλεονεκτήματα τους και τα μειονεκτήματα τους. Τέλος, το κεφάλαιο αυτό θα κλείσει με μερικές εκπαιδευτικές εφαρμογές που βοήθησαν στην υλοποίηση του συστήματος που πραγματευόμαστε στην συγκεκριμένη διπλωματική εργασία.

2.1 Τεχνολογίες Διαδικτύου

2.1.1 CSS Frameworks

Όπως οι περισσότερες γλώσσες πλέον, έτσι και η CSS, έχει εξελιχθεί με καινούργια frameworks, που κάνουν πιο εύκολη την συγγραφή της ή παρέχουν έτοιμα πράγματα. Ωστόσο πριν αναλύσουμε μερικά από τα σημαντικότερα frameworks, πρέπει να δούμε πως λειτουργεί η css. Η γλώσσα αυτή στην ουσία παίρνει μία ιδιότητα από τον κώδικα HTML, την κλάση, ως αναγνωριστικό και προσθέτει δικά της στοιχεία όπως χρώμα, μέγεθος γραμματοσειρά κτλπ.

2.1.1.1 Bootstrap

Πλεονεκτήματα του Bootstrap:

1. **Ταχύτητα σχεδίασης:** Το Bootstrap επιτρέπει στους προγραμματιστές να δημιουργούν σε μικρό χρονικό διάστημα και με σχετική ευκολία διεπαφές.
2. **Ποικιλία:** Περίεχε μεγάλη ποικιλία στο τομέα του styling, με έτοιμα κομμάτια κώδικα σχετικά με φόρμες, είκονες κ.α.
3. **Κοινότητα:** Έχει μια μεγάλη και ενεργή κοινότητα χρηστών, καθώς και εκτενή τεκμηρίωση που βοηθά στην επίλυση προβλημάτων.
4. **Ευέλικτο:** Μπορεί να χρησιμοποιηθεί είτε για πολύ απλά έργα είτε για πιο πολύπλοκα, και μπορεί να προσαρμοστεί ανάλογα με τις ανάγκες.

Μειονεκτήματα του Bootstrap:

1. **Συνηθισμένη Εμφάνιση:** Επειδή αποτελεί ένα από τα πλέον χρησιμοποιημένα framework της CSS, και δεν είναι εύκολη η τροποποιήση του ήδη υπάρχοντα κώδικα, το αποτέλεσμα δεν είναι μοναδικό.
2. **Μέγεθος Αρχείων:** Το Bootstrap περιέχει πολλές λειτουργίες που μπορεί να μην χρησιμοποιήσεις πάντα, κάτι που οδηγεί σε μεγαλύτερο μέγεθος αρχείων κατά τη μεταφόρτωση.
3. **Δυσκολία Προσαρμογής:** Σε πιο προηγμένα και προσαρμοσμένα έργα, ενδέχεται να είναι πιο δύσκολο να πετύχεις εξειδικευμένα σχέδια χωρίς να αφαιρέσεις ή να τροποποιήσεις πολλά συστατικά.
4. **Περιορισμένος Έλεγχος:** Ενώ προσφέρει γρήγορες λύσεις, μπορεί να περιορίσει τον έλεγχο σου επί των λεπτομερειών σε σύγκριση με προσαρμοσμένες κατασκευές από το μηδέν.

2.1.1.2 Tailwind CSS

Πλεονεκτήματα του Tailwind CSS:

1. **Προσαρμογή και Ευελιξία:** Το Tailwind CSS προσφέρει μεγάλη ευελιξία και προσαρμοστικότητα. Μπορείς να προσαρμόσεις κάθε πτυχή του σχεδιασμού σου χρησιμοποιώντας τα utility classes.
2. **Μέγεθος Αρχείων:** Τα αρχεία που παράγονται από το Tailwind είναι συνήθως μικρότερα σε μέγεθος συγκριτικά με τα αρχεία που παράγονται από πιο συμβατικά πλαίσια.
3. **Απόδοση:** Η χρήση των utility classes μπορεί να οδηγήσει σε βελτιωμένη απόδοση, καθώς μόνο οι στυλ που χρησιμοποιούνται προστίθενται στα αρχεία CSS.
4. **Κοινότητά**Έχει μια ενεργή κοινότητα χρηστών και καλή τεκμηρίωση, παρέχοντας υποστήριξη για τους προγραμματιστές που χρησιμοποιούν το πλαίσιο.

Μειονεκτήματα του Tailwind CSS:

1. **Συντακτικό Εμπόδιο:** Η χρήση utility classes μπορεί να είναι δύσκολη για κάποιους προγραμματιστές που προτιμούν τον συνήθη συντακτικό των πλαισίων.
2. **Εμφανισιακή Ομοιομορφία:** Επειδή πολλά έργα χρησιμοποιούν τα ίδια utility classes, ενδέχεται να υπάρχει κάποια ομοιομορφία στην εμφάνιση των ιστοσελίδων.
3. **Μάθηση:** Ενδέχεται να απαιτηθεί χρόνος για να εξοικειωθείς με το συντακτικό και τη φιλοσοφία του Tailwind, ειδικά αν έχεις συνηθίσει άλλα πλαίσια.
4. **Μεγάλος Αριθμός Κλάσεων:** Όσο μεγαλώνει το έργο, τόσο γίνονται πιο πολλές και πιο πολύπλοκες οι κλάσεις

2.1.1.3 Semantic UI

Πλεονεκτήματα του Semantic UI:.

1. **Μεγάλη Ποικιλία :** Προσφέρει μια εκτεταμένη βιβλιοθήκη με πολλά έτοιμα UI components, επιτρέποντας γρήγορη και εύκολη ανάπτυξη.
2. **Κοινότητα και Τεκμηρίωση:** Διαθέτει μια ενεργή κοινότητα χρηστών και καλή τεκμηρίωση, παρέχοντας υποστήριξη και πόρους για τους προγραμματιστές.
3. **Ευκολία Εκμάθησης:** Το Semantic UI είναι σχετικά εύκολο στην εκμάθηση, ειδικά για αυτούς που έχουν εμπειρία με ανάλογα frameworks.

Μειονεκτήματα του Semantic UI:

1. **Μέγεθος Αρχείων:** Τα αρχεία CSS και JS του Semantic UI μπορεί να είναι μεγαλύτερα σε σχέση με άλλα ελαφρύτερα πλαίσια, όπως το Bootstrap.
2. **Περίπλοκη Δομή Κώδικα:** Σε πολύπλοκα σχέδια, η δομή του κώδικα μπορεί να γίνει περίπλοκη, ειδικά όταν χρησιμοποιούνται πολλά components.
3. **Περιορισμένη Κοινότητα:** Παρόλο που υπάρχει κοινότητα, η δραστηριότητά της δεν είναι τόσο έντονη όσο μερικά άλλα πλαίσια, όπως το Bootstrap ή το Tailwind CSS.

4. **Συντηρησιμότητα:** Εάν δεν τηρηθεί προσεκτικά η σημασιολογία του, η συντηρησιμότητα του κώδικα μπορεί να γίνει δύσκολη.

2.1.2 Javascript Frameworks/Libraries

Όπως η CSS, έτσι και η Javascript, έχει εξελιχθεί με τα χρόνια και πλέον αποτελεί αναπόσπαστο κομμάτι για την δημιουργία μία διαδικτυακής εφαρμογής. Η ανάπτυξη αυτή έφερε την ανάγκη για frameworks που θα διευκολύνουν τους προγραμματιστές, παρέχοντας πιο πολλές λειτουργίες σε καλύτερες ταχύτητες. Τα κυριότερα frameworks που υπάρχουν στην αγορά πλέον είναι η ReactJS (δεν θεωρείται ακόμα framework), η Angular και η VueJS.

Οι προτιμήσεις μεταξύ Angular, React και Vue εξαρτώνται σε μεγάλο βαθμό από τις ανάγκες, τις προτιμήσεις του αναπτυσσόμενου έργου και την εμπειρία των προγραμματιστών. Κάθε πλαίσιο έχει τα δικά του πλεονεκτήματα και χαρακτηριστικά. Ας δούμε ορισμένες κατευθυντήριες γραμμές που μπορεί να βοηθήσουν στην επιλογή:

1. **Angular:**
   - Κατάλληλο για μεγάλα και πολύπλοκα έργα.
   - Ενσωματώνει πλήρες σύστημα, συμπεριλαμβανομένου του Angular CLI, για αυτόματη διαχείριση κύκλων ζωής έργου.
   - Ενισχυμένη τηρητικότητα (strict typing) με τη χρήση TypeScript.
2. **React:**
   - Εξαιρετική ευελιξία και επεκτασιμότητα.
   - Ευρείς πόροι και κοινότητα, καθιστούν το React μια δημοφιλή επιλογή.
   - Ιδανικό για εφαρμογές μονής σελίδας (Single Page Applications - SPAs).
3. **Vue:**
   - Εύκολη εκμάθηση, ιδανική για αρχάριους προγραμματιστές.
   - Καλή επεκτασιμότητα και απλότητα χρήσης.
   - Χρησιμοποιεί πρότυπα HTML, CSS και JavaScript, κάτι που καθιστά ευκολότερο τον συνδυασμό με υπάρχοντα έργα.

Αποφάσισε με βάση τις ανάγκες του συγκεκριμένου έργου, το επίπεδο εμπειρίας της ομάδας ανάπτυξης, και τη συνολική αρχιτεκτονική που θες να υλοποιήσεις.

2.1.2.1 Angular

Το Angular είναι ένα πλαίσιο (framework) ανοιχτού κώδικα για την ανάπτυξη διαδικτυακών εφαρμογών (web applications) που δημιουργήθηκε από την Google. Εδώ είναι μερικά από τα πλεονεκτήματα και τα μειονεκτήματα του Angular:

Πλεονεκτήματα του Angular:

1. **Συνολική Δομή και Οργάνωση:** Το Angular προσφέρει μια συνολική δομή και οργάνωση για την ανάπτυξη εφαρμογών. Η χρήση του TypeScript (ένα υποσύνολο της JavaScript παρέχει μεγαλύτερή σαφήνεια και καθιστά το έργο πιο εύκολα διαχειρίσιμο.
2. **Δυνατότητες Δέσμευσης Δεδομένων (Two-Way Data Binding):** Το Angular προσφέρει δυνατότητες δέσμευσης δεδομένων που σημαίνει ότι αλλαγές στα

δεδομένα του μοντέλου αντικατοπτρίζονται αυτόματα στην παρουσίαση (UI) και αντιστρόφως.
3. **Ευελιξία και Επαναχρησιμοποίηση Κώδικα:** Η χρήση συστατικών (components) στο Angular προωθεί την επαναχρησιμοποίηση του κώδικα. Αυτό προσφέρει στον προγραμματιστή και περισσότερο χρόνο, και καθιστά πιο εύκολη την εύρεση πιθανών bug.
4. **Ενιαίο Περιβάλλον Ανάπτυξης (CLI):** Το Angular παρέχει ένα ισχυρό εργαλείο γραμμής εντολών (Command Line Interface - CLI) που διευκολύνει τη διαδικασία ανάπτυξης, δοκιμής και διανομής εφαρμογών.
5. **Κοινότητα:** Υπάρχει μια ενεργή κοινότητα γύρω από το Angular, που σημαίνει ότι υπάρχει πληθώρα πόρων, εκπαιδευτικών υλικών και επίλυση προβλημάτων.

Μειονεκτήματα του Angular:

1. **Μεγάλο Μέγεθος:** Η βιβλιοθήκη Angular είναι μεγαλύτερη σε σύγκριση με άλλα πλαίσια, κάτι που μπορεί να επηρεάσει το μέγεθος των εφαρμογών.
2. **Μάθηση:** Λόγω της πολυπλοκότητάς του, το Angular μπορεί να έχει μια δύσκολη καμπύλη μάθησης για νέους προγραμματιστές.
3. **Εγκατάσταση και Εκτέλεση Πολύπλοκη:** Η διαδικασία εγκατάστασης και εκτέλεσης μπορεί να φανεί περίπλοκη σε σχέση με άλλα πιο απλά πλαίσια.
4. **Υποχρεωτική Χρήση TypeScript:** Ενώ η χρήση της TypeScript μπορεί να είναι πλεονέκτημα για ορισμένους, κάποιοι προγραμματιστές μπορεί να προτιμούν την JavaScript, καθώς μπορεί η Typescript να προσφέρει πολλά αλλά ταυτόχρονα είναι και πιο δύσκολη κατά την δημιουργία του έργου.

Επομένως, η Angular είναι ένα ισχυρό framework για μεγάλες και πολύπλοκες εφαρμογές, αλλά απαιτεί επένδυση χρόνου για την εκμάθηση του. Η επιλογή του εξαρτάται συχνά από τις ανάγκες και την προτίμηση του εκάστοτε προγραμματιστή.

2.1.2.2 React

React είναι ένα ανοιχτού κώδικα βιβλιοθήκη για τη δημιουργία διαδικτυακών εφαρμογών (web applications). Εδώ είναι μερικά από τα πλεονεκτήματα και τα μειονεκτήματα του React:

Πλεονεκτήματα του React:

1. **Επαναχρησιμοποίηση Στοιχείων (Component Reusability):** Η δομή του React επιτρέπει την εύκολη επαναχρησιμοποίηση στοιχείων (components), πράγμα που οδηγεί σε καλύτερη οργάνωση και συντήρηση του κώδικα.
2. **Αποτελεσματική Ανανέωση (Efficient Updates):** Το React χρησιμοποιεί το Virtual DOM για αποτελεσματική διαχείριση και ανανέωση του UI, βελτιώνοντας την απόδοση της εφαρμογής.
3. **Κοινότητα:** Υπάρχει μια εκτεταμένη και ενεργή κοινότητα γύρω από το React, με πολλούς πόρους, βιβλιοθήκες και εργαλεία που υποστηρίζονται.
4. **Απλότητα:** Η σύνταξη του JSX, που επιτρέπει την ενσωμάτωση της HTML στον κώδικα JavaScript, καθιστά το React ευανάγνωστο και ευαίσθητο στη συντακτική ανάλυση.
5. **Ευελιξία:** Το React μπορεί να χρησιμοποιηθεί μόνο του ή σε συνδυασμό με άλλες τεχνολογίες, όπως το Redux, για τη διαχείριση της κατάστασης.

Μειονεκτήματα του React:

1. **Μάθηση:** Για νέους προγραμματιστές, η μάθηση του React μπορεί να απαιτεί χρόνο για να κατανοηθεί πλήρως, ειδικά με τη χρήση του JSX. Τα αρχεία JSX μπορεί να μπερδέψει κάποιο προγραμματιστή που δεν έχει συνηθίσει να έχει στο ίδιο αρχείο και την HTML και τις συναρτήσεις της Javascript.
2. **Υποχρεωτική Χρήση JSX:** Αν και το JSX είναι ένα ισχυρό εργαλείο, μερικοί προγραμματιστές μπορεί να προτιμούν την αποκλειστική χρήση HTML και JavaScript.
3. **Αλλαγές API:** Με το πέρασμα του χρόνου, ορισμένες φορές υπάρχουν αλλαγές στις API του React, που μπορεί να επηρεάσουν τον υπάρχοντα κώδικα. Αυτό οφείλεται στο γεγονός πως η React δεν θεωρείται ακόμα framework, αλλα περισσότερο βιβλιοθήκη.

Και τα δύο πλαίσια, Angular και React, έχουν τα πλεονεκτήματα και τα μειονεκτήματά τους, και η επιλογή μεταξύ τους εξαρτάται συχνά από τις απαιτήσεις και τις προτιμήσεις του έργου και της ομάδας ανάπτυξης.

2.1.2.3 Vue

Vue.js είναι ένα πλαίσιο ανοιχτού κώδικα για τη δημιουργία διαδικτυακών εφαρμογών. Εδώ είναι μερικά από τα πλεονεκτήματα και τα μειονεκτήματα του Vue.js:

Πλεονεκτήματα του Vue.js:

1. **Ευκολία Εκμάθησης**Το Vue.js είναι εύκολο στην κατανόηση και στην εκμάθηση, ειδικά για αρχάριους προγραμματιστές. Η σύνταξη του Vue.js είναι απλή και αναγνώσιμη.
2. **Συνολική Απόδοση (Overall Performance):** Η Vue.js προσφέρει καλή συνολική απόδοση, με ευέλικτες επιλογές για τον έλεγχο της απόδοσης της εφαρμογής.
3. **Κοινότητα:** Υπάρχει μια ισχυρή κοινότητα γύρω από το Vue.js, με πλούσια τεκμηρίωση και πολλούς πόρους για την επίλυση προβλημάτων.
4. **Μέγεθος:** Ο πυρήνας του Vue.js είναι σχετικά μικρός, κάτι που μπορεί να βοηθήσει στη γρήγορη φόρτωση των εφαρμογών.

Μειονεκτήματα του Vue.js:

1. **Μικρότερη Κοινότητα από Angular και React:** Ενώ η κοινότητα του Vue.js είναι ενεργή, είναι μικρότερη σε σχέση με τις κοινότητες του Angular και του React.
2. **Λιγότερες Βιβλιοθήκες και Εργαλεία:** Σε σύγκριση με τα άλλα δύο πλαίσια, το Vue.js έχει λιγότερες βιβλιοθήκες και εργαλεία που υποστηρίζονται.
3. **Επεκτασιμότητα** Η λιγότερη πολυπλοκότητα του Vue.js μπορεί να σημαίνει ότι δεν είναι τόσο κατάλληλο για μεγάλα έργα όπως το Angular.

Κάθε ένα από τα τρία πλαίσια, Angular, React και Vue, έχει τα δικά του πλεονεκτήματα και μειονεκτήματα, και η επιλογή μεταξύ τους εξαρτάται από τις ανάγκες και τις προτιμήσεις του έργου και της ομάδας ανάπτυξης.

2.1.2.4 Επιλογή Frameword

Η επιλογή που έγινε μεταξύ αυτών των τριών ήταν το framework της React. Αρχικά, από όλα τα πλεονεκτήματα που παρέχει αυτή η βιβλιοθήκη, το κυριότερο ήταν η μεγάλη κοινότητα που υπάρχει. Αυτό καθιστούσε πιο εύκολη την επίλυση προβλημάτων. Επιπλέον, αποτέλεσε προσωπική επιλογή, λόγω της εξοικείωσης που υπάρχει με αυτό το framework, καθώς και το μέγεθος της κοινότητας που το χρησιμοποιεί, το κατέστησε το πλέον κατάλληλο για την υλοποίηση της συγκεκριμένης εφαρμογής. Ωστόσο, αυτό δεν σημαίνει πως τα άλλα δύο δεν θα μπορούσαν να υλοποιήσουν την εφαρμογή αυτή.

2.1.3 Rest API (Representational State Transfer)

Το Rest API είναι μία από τις βασικότερες και πιο χρησιμοποιημένες αρχιτεκτονικές για την σχεδίαση διαδικτυακών εφαρμογών. Αυτό οφείλεται στο γεγονός πως είναι μία ιδιαίτερα ελαφριά αρχιτεκτονική και εύκολα επεκτάσιμη. Αποτελεί ένα παρακλάδι του μοντέλου πελάτη-εξυπηρετητή, αφού η βασική του λειτουργία είναι να μεταφέρει δεδομένα από τον εξυπηρετητή στον πελάτη, όταν αυτός κάνει κάποιο αίτημα. Η επικοινωνία των δύο πλευρών γίνεται πάνω στο HTTP πρωτόκολλο, επομένως υποστηρίζει τις εξής μεθόδους:

- GET: χρησιμοποιείται όταν ο πελάτης θέλει να λάβει δεδομένα από τον εξυπηρετητή. Για παράδειγμα να λάβει την λίστα με τις κρατήσεις που έχει κάνει
- POST: χρησιμοποιείται όταν ο πελάτης θέλει να δημιουργήσει κάποια καινούργια δεδομένα. Για παράδειγμα να κάνει μία νέα κράτηση
- PUT: πάλι ο πελάτης στέλνει κάποια δεδομένα, αυτή την φορά με σκοπό να ενημερώσει κάποιο από τα δεδομένα που έχει ο εξυπηρετητής. Για παράδειγμα να αλλάξει ώρα σε μία κράτηση που έχει κάνει
- DELETE: χρησιμοποιείται για να γίνει διαγραφή δεδομένων. Για παράδειγμα να ακυρώσει την κράτηση του
- PATCH: έχει παρόμοια λειτουργία με το PUT, ωστόσο αυτό ενημερώνει μόνο συγκεκριμένα πεδία.

Για να μπορεί όμως ο εξυπηρετητής για να διαχειριστεί τα αιτήματα που δέχεται, ορίζει κάποια URLs, ή αλλιώς endpoints που είναι μοναδικά. Στην μόνη περίπτωση που δύο endpoint είναι ακριβώς τα ίδια είναι μόνο αν χρησιμοποιούν διαφορετική μέθοδος (GET,POST Κτλπ).

Στην περίπτωση που ο πελάτης θέλει να στείλει δεδομένα στον εξυπηρετητή αυτό γίνεται από το request body. Τα δεδομένα σε αυτό το σημείο του αιτήματος πρέπει να είναι της μορφής JSON ή XML. Ανάλογα όταν ο εξυπηρετητής θέλει να στείλει δεδομένα στον πελάτη τα στέλνει με την ίδια δομή και επισυνάπτει στο αίτημα και ένα HTTP status code. Οι πιο συνηθισμένοι κώδικες είναι οι εξής:

- 200(success): Δηλώνει πως το αίτημα πραγματοποιήθηκε με επιτυχία
- 400: Ο εξυπηρετητής δηλώνει στον πελάτη πως δεν κατάλαβε το αίτημα

- 401: Ο εξυπηρετητής δηλώνει στον πελάτη πως προσπαθεί να έχει πρόσβαση σε πόρους που δεν έχει δικαίωμα
- 404: Ο εξυπηρετητής δηλώνει πως ο πόρος που ζητήθηκε δεν βρέθηκε

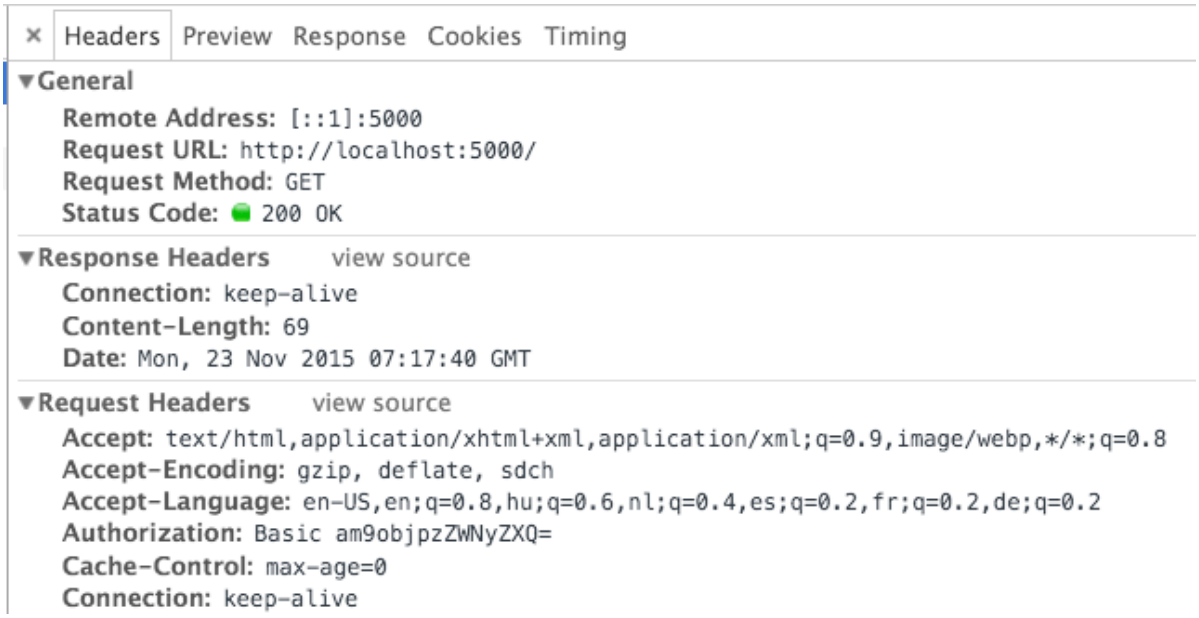
Οι βασικές αρχές του REST περιλαμβάνουν:

1. **Διακριτικότητα Πόρων (Resource Identification):** Κάθε πόρος (resource) στο σύστημα πρέπει να έχει ένα μοναδικό αναγνωριστικό (URI) που τον προσδιορίζει.
2. **Πρόσβαση στους Πόρους μέσω Αναπαραστάσεων (Representation):** Οι πελάτες αναπαριστούν και αλληλεπιδρούν με τους πόρους μέσω αναπαραστάσεων, που μπορεί να είναι σε μορφή κειμένου, XML, ή JSON.
3. **Συνεδρία Κατάστασης (Stateless Communication):** Κάθε αίτημα από έναν πελάτη πρέπει να περιέχει όλες τις πληροφορίες που χρειάζονται για να κατανοηθεί και να επεξεργαστεί από τον διακομιστή. Καμία κατάσταση δεν αποθηκεύεται στον διακομιστή ανάμεσα στα αιτήματα.
4. **Κριτήρια Επικοινωνίας (Uniform Interface):** Το REST παρέχει μια ομοιογενή διεπαφή επικοινωνίας μεταξύ πελάτη και διακομιστή. Αυτό περιλαμβάνει την ονοματοδοσία των πόρων με μοναδικά URIs, την χρήση αναπαραστάσεων για τους πόρους, και την αυτοπεριγραφή των περιορισμών (constraints).

Η αρχιτεκτονική REST είναι ευέλικτη και επιτρέπει τη δημιουργία απλών και αποτελεσματικών διασυνδέσεων ανάμεσα σε διάφορες υπηρεσίες και εφαρμογές. Είναι ευρέως χρησιμοποιούμενη σε web εφαρμογές για την ανταλλαγή δεδομένων.

2.1.4 Δομή Αιτήματος HTTP

Αφού αναλύσαμε πως λειτουργεί ένα Rest API, πρέπει να δούμε και την δομή μερικών αιτημάτων HTTP. Καταλαβαίνουμε πως η μορφή τους είναι αυστηρή, προκειμένου να επιτευχθεί η επικοινωνία μεταξύ πελάτη και εξυπηρετητή. Προκειμένου να δούμε τέτοια αιτήματα, πρέπει να πάμε στην επισκόπηση του φυλλομετρήτη, στο κομμάτι του Δικτύου. Ένα απλό παράδειγμα είναι το εξής



Εικόνα 1: HTTP αίτημα

Το συγκεκριμένο παράδειγμα αποτελεί ίσως το πιο απλό αίτημα. Παρατηρούμε στις Γενικές πληροφορίες παρέχεται το URL που θα γυρίσει τους πόρους που θέλουμε, την μέθοδο, κάνοντας με αυτό τον τρόπο γνωστό αν θέλουμε να ανεβάσουμε δεδομένα ή να λάβουμε και το Status Code που υποδεικνύει το αποτέλεσμα του αιτήματος. Δηλαδή σε αυτό το αίτημα πραγματοποιούμε ένα GET αίτημα στο http://localhost:5000 με θετικό αποτέλεσμα. Στην συνέχεια βλέπουμε κάποιες ρυθμίσεις που απαιτεί ο εξυπηρετητής από τον πελάτη για την σωστή επικοινωνία, όπως τι είδους δεδομένα δέχεται. Αξίζει να αναφερθεί πως το συγκεκριμένο αίτημα έχει και αυθεντικοποιήση, η οποία δίνεται μέσω του Request Header, στον τομέα Authorization.

2.1.5 Έρευνα Framework για Rest API

Όπως στο κομμάτι του πελάτη, έτσι και στο κομμάτι του εξυπηρετητή, οι επιλογές για το framework ποικίλλουν. Στην περίπτωση του εξυπηρετητή υπήρχαν επιλογές διάφορων γλωσσών και όχι μόνο της Javascript. Τα δημοφιλέστερα από αυτά είναι τα εξής

- NodeJS (express.js)
- Laravel PHP
- Django Python

2.1.5.1 NodeJS (express.js)

Το Node.js είναι ένα open-source πλαίσιο ανάπτυξης για την εκτέλεση server-side JavaScript. Ακολουθούν πλεονεκτήματα και μειονεκτήματα του Node.js:

Πλεονεκτήματα του Node.js:

1. **Μη Μπλοκαρισμένη Εκτέλεση (Non-blocking Execution):** Ο Node.js χρησιμοποιεί ένα μη μπλοκαριστικό I/O μοντέλο, το οποίο το καθιστά κατάλληλο για εφαρμογές με πολλά αιτήματα και συμβάντα.
2. **Απόδοση:** Είναι γρήγορο και αποτελεσματικό λόγω της μη-μπλοκαριστικής αρχιτεκτονικής του, καθιστώντας το κατάλληλο για πραγματικού χρόνου εφαρμογές.
3. **Ευελιξία:** Μπορεί να χρησιμοποιηθεί για την ανάπτυξη σε JavaScript, που είναι ευρέως γνωστή και χρησιμοποιείται.
4. **Κοινότητα:** Έχει μια ισχυρή κοινότητα προγραμματιστών και ένα εκτεταμένο οικοσύστημα πακέτων (npm).
5. **Ιδανικό για Real-time Εφαρμογές:** Κατάλληλο για εφαρμογές πραγματικού χρόνου όπως τσατ, παιχνίδια, και streaming.

Μειονεκτήματα του Node.js:

1. **Ασφάλεια:** Λόγω του χρήση της JavaScript για τον προγραμματισμό στην πλευρά του server, μπορεί να υπάρχουν προκλήσεις στον τομέα της ασφάλειας, εάν δεν χειρίζεσαι σωστά την είσοδο από τον χρήστη.
2. **Ασύγχρονη Προγραμματισμός:** Ο ασύγχρονος προγραμματισμός μπορεί να καταστήσει τον κώδικα δυσανάγνωστο για ορισμένους προγραμματιστές. Επιπλέον δεν είναι ένα συνηθισμένο concept, οπότε μπορεί να μην είναι εύκολο για όλους τους προγραμματιστές.

2.1.5.2 Laravel PHP

Το Laravel είναι ένα δημοφιλές PHP framework για την ανάπτυξη web εφαρμογών. Ας εξετάσουμε τα πλεονεκτήματα και τα μειονεκτήματα του Laravel:

Πλεονεκτήματα του Laravel:

1. **Απλός και Ευκολία:** Το Laravel υποστηρίζει έναν καθαρό και ευανάγνωστο κώδικα, επιτρέποντας στους προγραμματιστές να αναπτύξουν εύκολα και γρήγορα εφαρμογές.
2. **Eloquent ORM:** Το Eloquent ORM παρέχει μια ευέλικτη και εκφραστική τρόπο για την αλληλεπίδραση με τη βάση δεδομένων, καθιστώντας την ανάπτυξη και τη συντήρηση της βάσης δεδομένων ευκολότερη.
3. **Artisan:** Το Laravel παρέχει ένα ενσωματωμένο εργαλείο γραμμής εντολών, το Artisan, που διευκολύνει τις καθημερινές εργασίες των προγραμματιστών, όπως η δημιουργία μοντέλων, controllers, migrations κ.ά.
4. **Κοινότητα:** Έχει μια ενεργή και έκτακτη κοινότητα προγραμματιστών, καθώς και καλή τεκμηρίωση, που διευκολύνει τους προγραμματιστές να βρουν υποστήριξη και πληροφορίες.

Μειονεκτήματα του Laravel:

1. **Επίδοσης:** Η χρήση ενός framework μπορεί να οδηγήσει σε μια ελαφρά μείωση της απόδοσης σε σχέση με τον καθαρό PHP, αν και αυτή η διαφορά έχει μειωθεί στα τελευταία χρόνια.
2. **Μεγάλο Μέγεθος Πυρήνα Κώδικα:** Ο πυρήνας του Laravel είναι σχετικά μεγάλος, και αυτό μπορεί να επιφέρει μεγαλύτερη κατανάλωση πόρων σε σχέση με ελαφρύτερα frameworks.

2.1.5.3 Django Python

Το Django είναι ένα δημοφιλές Python web framework που χρησιμοποιείται για την ανάπτυξη web εφαρμογών. Ας εξετάσουμε τα πλεονεκτήματα και τα μειονεκτήματα του Django:

Πλεονεκτήματα του Django:

1. **Πλούσια Λειτουργικότητα και Έτοιμα Χαρακτηριστικά:** Το Django παρέχει ένα εκτενές σύνολο λειτουργιών και έτοιμα χαρακτηριστικά, όπως αυθεντικοποίηση, διαχείριση διαδραστικών φορμών, ORM (Object-Relational Mapping) κ.ά.
2. **ORM:** Το ORM του Django επιτρέπει στους προγραμματιστές να αλληλεπιδρούν με τη βάση δεδομένων χρησιμοποιώντας αντικείμενα Python αντί για SQL.
3. **Ενσωματωμένο Διαχειριστικό Σύστημα** Το Django παρέχει ένα έτοιμο και εύχρηστο διαχειριστικό σύστημα που επιτρέπει τη διαχείριση των δεδομένων της εφαρμογής με ελάχιστο κώδικα.
4. **Κοινότητα:** Η κοινότητα Django είναι ενεργή και υπάρχει εκτενής τεκμηρίωση, παρέχοντας υποστήριξη και πληροφορίες για τους προγραμματιστές.
5. **Ασφάλεια:** Το Django έχει ενσωματωμένα μέτρα ασφαλείας που βοηθούν στην προστασία των εφαρμογών από επιθέσεις όπως η εγχείρηση SQL και οι επιθέσεις Cross-Site Scripting (XSS).

Μειονεκτήματα του Django:

1. **Μάθηση:** Λόγω της πλούσιας λειτουργικότητάς του, το Django μπορεί να απαιτεί κάποιο χρόνο για να μάθεις όλες τις λεπτομέρειες και τα χαρακτηριστικά του.
2. **Βάρος στη Μνήμη:** Κατά τη διάρκεια της εκκίνησης, το Django μπορεί να χρησιμοποιεί περισσότερη μνήμη σε σύγκριση με άλλα ελαφρύτερα frameworks.
3. **Λιγότερη Ευελιξία:** Σε ορισμένες περιπτώσεις, η έτοιμη λειτουργικότητα του Django μπορεί να οδηγήσει σε λιγότερη ευελιξία σε σχέση με άλλα frameworks που προσφέρουν λιγότερες προκαθορισμένες λύσεις.

2.1.5.4 Επιλογή Framework Εξυπηρετητή

Στην εφαρμογή μας, επιλέξαμε τη χρήση του Node.js κυρίως λόγω του live chat. Η απόφαση αυτή βασίστηκε σε αρκετούς καίριους λόγους που συνδυάζουν απόδοση και ευελιξία. Το Node.js, χρησιμοποιώντας το μη-μπλοκαριστικό I/O μοντέλο, είναι εξαιρετικά αποδοτικό στην αντιμετώπιση πολλών συνδυασμένων συνομιλιών, επιτρέποντας στους χρήστες να αλληλεπιδρούν σε πραγματικό χρόνο με το live chat χωρίς καθυστέρηση. Η δυνατότητα αυτή είναι ιδιαίτερα σημαντική για μια εφαρμογή που δίνει έμφαση στην άμεση και αποτελεσματική επικοινωνία με τους χρήστες. Επιπλέον, η ευελιξία της JavaScript σε συνδυασμό με το Node.js μας παρέχει τη δυνατότητα επαναχρησιμοποίησης κώδικα και αποτελεσματικής ανάπτυξης, ενώ η ζωντανή επικοινωνία παρέχει έναν πιο συναρπαστικό και διαδραστικό χαρακτήρα στην εμπειρία του χρήστη.

2.2 Παρόμοια Συστήματα

Το σύστημα που πραγματευόμαστε στην παρούσα διπλωματική εργασία αποτελεί έναν συνδυασμό συστημάτων εκπαιδευτικών και μη που ήδη υπάρχουν στην αγορά και είναι

ευρέως γνωστά. Για να μιλήσουμε για αυτά τα συστήματα όμως, πρέπει πρώτα να χωρίσουμε το συγκεκριμένο σύστημα στα επιμέρους συστήματα του. Επομένως, αρχικά έχουμε ένα σύστημα που βασίζεται στην καταγραφή της προόδου μίας εργασίας και στην συνέχεια ένα σύστημα για την εύκολη επικοινωνία μεταξύ των χρηστών.

Συστήματα εκπαιδευτικού χαρακτήρα που είναι υπεύθυνα για την πρόοδο σε ένα μάθημα ή μία εργασία είναι το Moodle, Blackboard Learn και το Google Classroom. Ας πάρουμε για παράδειγμα το Moodle, που δίνει την δυνατότητα στον διαχειριστή του συστήματος να δημιουργήσει ένα μάθημα, να προσθέσει ασκήσεις ή ακόμα και διαγωνίσματα και στην συνέχεια όσοι φοιτητές είναι εγγεγραμμένοι στο συγκεκριμένο μάθημα να έχουν πρόσβαση στους πόρους του μαθήματος. Επιπλέον, παρέχει την δυνατότητα για να ανάρτηση αρχείων, όλων των τύπων, κάνοντας πιο εύκολη την διαμοίραση της πληροφορίας και την βαθμολόγηση του καθηγητή. Το κοινό σημείο που έχει το σύστημα που αναπτύχθηκε σε αυτή την διπλωματική εργασία με το Moodle είναι η ομαδοποίηση φοιτητών, στο ένα ανά διπλωματική εργασία, στο άλλο ανά μάθημα. Επιπλέον, και τα δύο έχουν ευδιάκριτους ρόλους, δηλαδή και στα δύο υπάρχει ο μαθητής και ο καθηγητής, ο οποίος έχει και ρόλους διαχειριστή. Τέλος και τα δύο, έχουν έναν υπερδιαχειριστή, που έχει πρόσβαση σε κάποιες ρυθμίσεις του συστήματος, όπως διαγραφή χρηστών.

Επίσης όμως, η δυνατότητα για αποστολή μηνυμάτων σε πραγματικό χρόνο υποστηρίζεται από λογισμικά όπως το Microsoft Teams. Το κοινό σημείο που έχουν οι δύο εφαρμογές είναι πως παρέχουν την δυνατότητα είτε για ομαδικό, είτε για προσωπική συνομιλία. Σε αυτή την συνομιλία υποστηρίζεται και η προώθηση αρχείων.

Τέλος, άλλο ένα σύστημα που μελετήθηκε εκτενώς και επηρέασε σε μεγάλο βαθμό την υλοποίηση αυτής της Διπλωματικής Εργασίας, είναι το Github/Gitlab. Η βασική ιδέα αυτών των δύο συστημάτων είναι όρος repository. Σε αυτό το repository, υπάρχει κώδικα και αναρτημένα αρχεία, και ο χρήστης προωθεί όσες αλλαγές θέλει, που ονομάζονται commits, διατηρώντας μία ιστορικότητα σχετικά με τις αλλαγές του. Το σύστημα αυτό αποτέλεσε βασική ιδέα, καθώς συνδυάζει τόσο την ανάρτηση αρχείων, όσο και την διατήρηση ιστορικότητας και καταγραφής της προόδου του έργου. Η κυριότερη διαφορά ωστόσο είναι ο τρόπος δημιουργίας του repository. Πιο συγκεκριμένα, στο Github ο χρήστης έχει την ικανότητα να δημιουργήσει δημόσια ή ιδιωτικά repositories, καθώς δίνεται και η επιλογή σε ένα repository να υπάρχει μόνο ένα άτομο. Στο σύστημα όμως αυτής της εργασίας, δεν υπάρχει κάποιο νόημα σε ατομικές διπλωματικές εργασίες, δηλαδή να μην έχει πρόσβαση ο καθηγητής στην πρόοδο, άρα κάθε εργασία θα μπορούσε να παραλληλισθεί με ένα ίδιωτικό repository που έχουν δικαιώματα πρόσβασης περισσότεροι από ένας χρήστης.

2.3 Σύνοψη Κεφαλαίου

Το συγκεκριμένο κεφάλαιο εστιάζει περισσότερο στην έρευνα και στην επιλογή κατάλληλων μεθόδων. Αφού αναλύθηκε ο τρόπος λειτουργίας του φυλλομετρητή και βασικοί όροι όπως τα αιτήματα και το Rest Api, συζητήθηκαν οι τεχνολογίες μαζί με τα πλεονεκτήματα και τα μειονεκτήματα, προκειμένου να βγουν έγκυρα συμπεράσματα σχετικά με την επιλογή μας.

3. Σχεδιασμός Εφαρμογής

Στο κεφάλαιο αυτό θα γίνει αναλυτική συζήτηση σχετικά με την αλληλεπίδραση ανθρώπου-υπολογιστή, τον σχεδιασμό της βάσης, καθώς και τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση του συστήματος.

3.1 Αλληλεπίδραση Ανθρώπου-Υπολογιστή

Η αλληλεπίδραση μεταξύ ανθρώπου και υπολογιστή είναι κομβική για ένα καλά δομημένο σύστημα. Κανένα σύστημα, ακόμα και αν πληρεί όλες τις προδιαγραφές, δεν μπορεί να τεθεί σε λειτουργία αν δεν φιλικό προς τους χρήστες του (user-friendly). Επομένως, όταν υλοποιήθηκαν οι λειτουργίες του συστήματος, που θα αναλυθούν στο επόμενο κεφάλαιο, έπρεπε να δοθεί εμφάση στην ευχρηστεία του συστήματος, από τους χρήστες του (καθηγητές και φοιτητές.

3.1.1 Βασικές αρχές UI/UX

Προκειμένου να επιτευχθεί ορθή αλληλεπίδραση ανθρώπου-υπολογιστή πρέπει να ακολουθηθούν κάποια συγκεκριμένες σταθερές. Πιο συγκεκριμένα υπάρχουν σταθερές που κρίνουν αν μία εφαρμογή ακολουθεί σωστά πρότυπα για την διευκόλυνση των χρηστών της. Μερικές από αυτές τις σταθερές είναι οι εξής:

- Απλότητα: Μία εφαρμογή οφείλει να έχει μόνο τα απαραίτητα στοιχεία. Η χρήση μη απαραίτητων στοιχείων οδηγεί στην σύγχυση του χρήστη και στην δυσκολία εύρεσης τη λειτουργίας που θέλει.
- Σταθερότητα: Το σχέδιο της εφαρμογής πρέπει να ακολουθεί σταθερά κάποιες μεταβλητές, όπως χρώμα, γραμματοσειρά, padding και margin. Η σταθερά αυτή βοηθάει στην καλύτερη οπτική ανάλυση της εφαρμογής από τον χρήστη

- Σαφήνεια: Η χρήση label καθιστά πιο ξεκάθαρο στον χρήστη την λειτουργία που κρύβεται πίσω από την διεπαφή.
- Αποκριτικότητα: Μία εφαρμογή οφείλει να λειτουργεί και να είναι εύχρηστη σε κάθε είδους οθόνης, ανεξαρτήτως των διαστάσεων της.
- Προσιτότητα: Η εφαρμογή πρέπει να μπορεί να είναι εύχρηστη και για άτομα με ειδικές ανάγκες. Αυτό μπορεί να επιτευχθεί με κατάλληλη αντίθεση στα χρώματα, εναλλαγή εικόνων με κείμενο και πλοήγηση στην εφαρμογή αποκλειστικά μέσω του πληκτρολογίου.

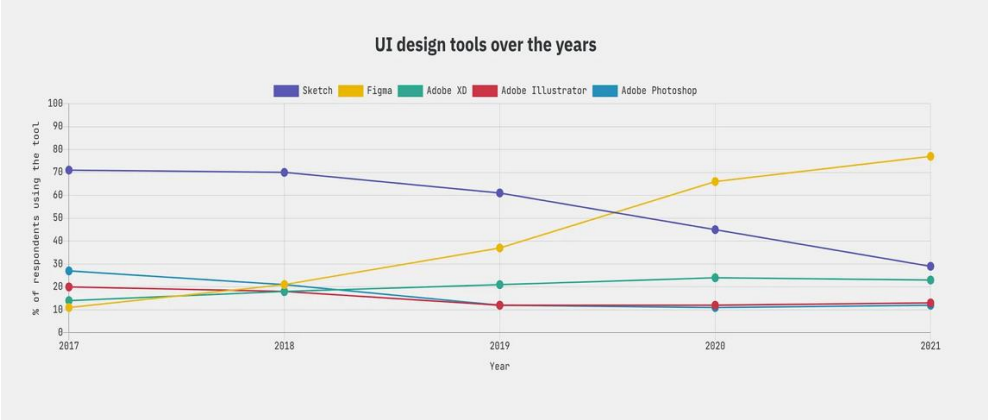
Εικόνα 2: Λόγοι κακής διαδικτυακής εφαρμογής

3.1.2 Διαδικασία δημιουργίας UI

Έχοντας κατά νου όσα αναπτύχθηκαν στο προηγούμενο κεφάλαιο, για τον σχεδιασμό της εφαρμογής έπρεπε να ακολουθηθούν συγκεκριμένα βήματα, ώστε να αποφευχθούν λάθη που θα κρατούσανε χρονικά πίσω το έργο μας. Τα βήματα αυτά ήταν τα εξής:

- Έρευνα: Απαραίτητο βήμα προκειμένου να αναλυθούν στατιστικά σχετικά με τις προτιμήσεις των χρηστών. Αυτό το βήμα είναι τα θεμέλια για την δημιουργία της εφαρμογής σε επίπεδο πελάτη. Σε αυτό το στάδιο αναλύονται και τι προσδοκούμε από την εφαρμογή, διαμορφώνοντας ανάλογα την εφαρμογή.
- Ιδέα: Αφού έχει γίνει η έρευνα και έχουμε καταλήξει σε κάποιες αρχές που θα ακολουθηθούν, ξεκινάμε να «οραματιζόμαστε» πως θα είναι η εφαρμογή από άποψη χρωμάτων, δομής και γραμματοσειράς

- Δημιουργία πρωτότυπου: Κομβικό βήμα πριν αρχίσουμε να υλοποιούμε την εφαρμογή, είναι η δημιουργία ενός πρωτότυπου. Το πρωτότυπο μπορεί να δημιουργηθεί σε αρκετές εφαρμογές, με σκοπό την αξιολόγηση, πριν αρχίσει η υλοποίηση. Μερικές χαρακτηριστικές εφαρμογές για την δημιουργία πρωτότυπου είναι το Figma, Adobe Photoshop, Sketch. Στην συγκεκριμένη εφαρμογή η δημιουργία του πρωτότυπου έγινε με χρήση του εργαλείου Figma, αφού μετά την έρευνα που έγινε στο πρώτο βήμα καταλήξαμε ότι



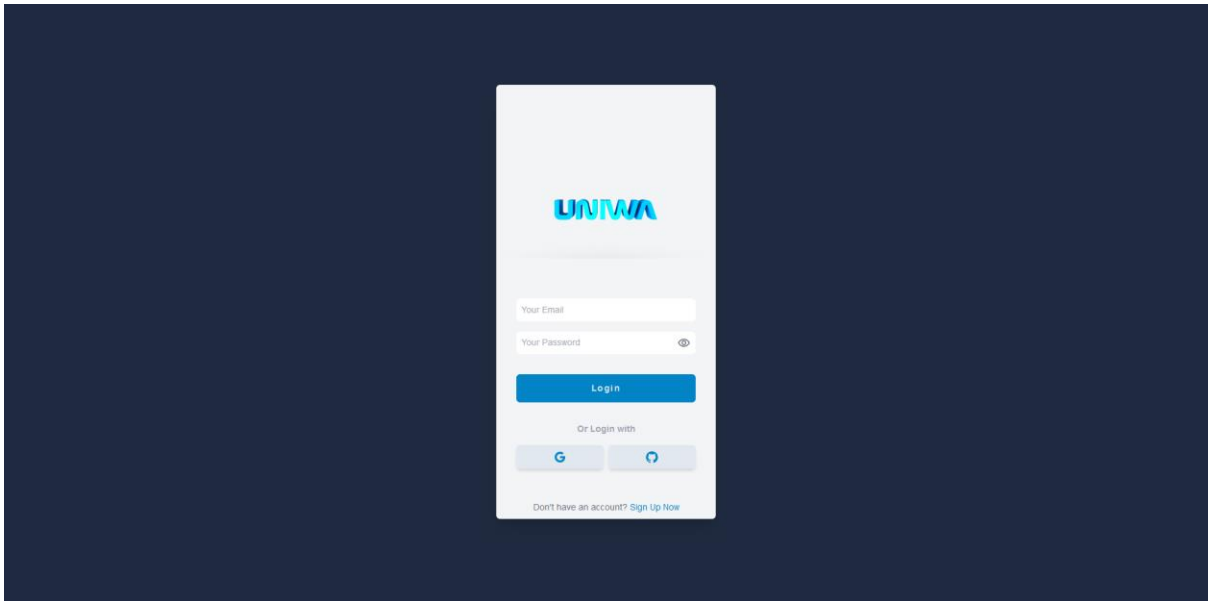
Εικόνα 3: Εργαλεία ανάπτυξης UI/UX

- Υλοποίηση: Στο τελευταίο βήμα ξεκινάμε με χρήση κάποια γλώσσας προγραμματισμού να δημιουργούμε την αλληλεπίδραση ανθρώπου-υπολογιστή.

3.1.3 Τελικό UI

Λαμβάνοντας όλα τα παραπάνω υπόψιν, οι σελίδες που δημιουργήθηκαν είναι οι εξής. Αρχικά, η πρώτη σελίδα που βλέπει ο χρήστης είναι η σελίδας σύνδεσης (Login page).

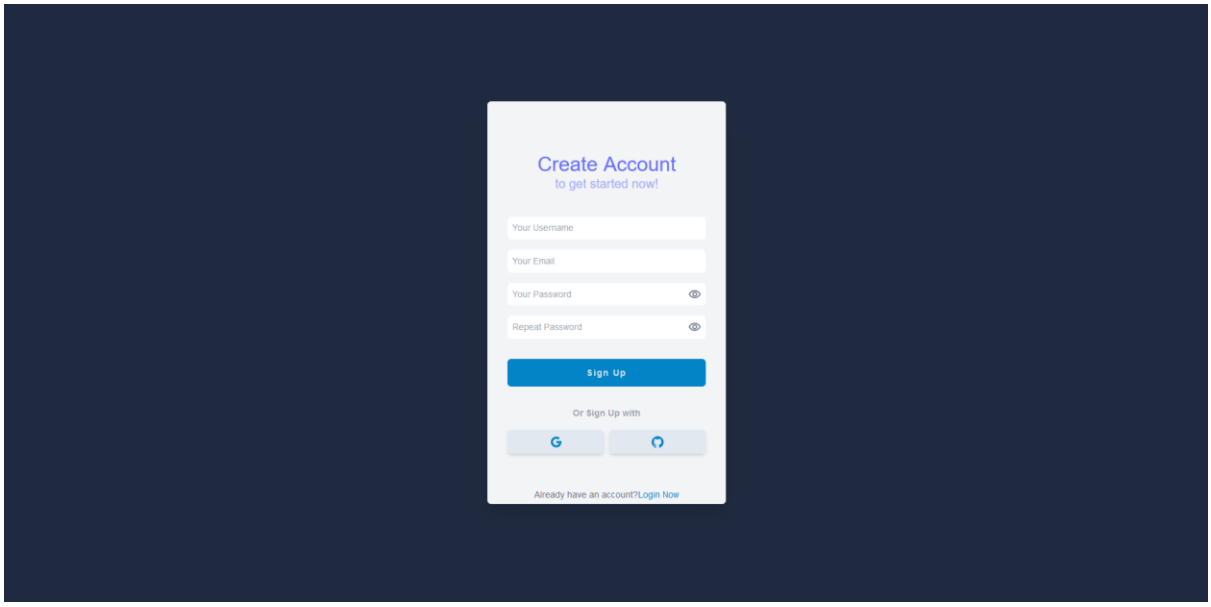
3.1.3.1 Login Page



Εικόνα 4: Σελίδα Σύνδεσης

Παρατηρούμε πως η συγκεκριμένη σελίδα αποτελείται από δύο περιοχές για να βάλει ο χρήστης το email του και τον κωδικό του. Βασική αρχή είναι είτε να υπάρχει κάποιο κείμενο πάνω από το input, ή το λεγόμενο placeholder, προκειμένου ο χρήστης να ξέρει τι πρέπει να συμπληρώσει. Επιπλέον, υπάρχουν τρία κουμπιά, το ένα είναι υπεύθυνο για την σύνδεση μέσω χρήσης email και κωδικού, και τα άλλα δύο για να γίνει σύνδεση είτε μέσω Google, είτε μέσω Github. Επιπλέον, υπάρχει υπερσύνδεσμος σε περίπτωση που δεν έχει ο χρήστης λογαριασμό, οδηγώντας τον στην ανάλογη σελίδα για να δημιουργήσει.

3.1.3.2 Register Page

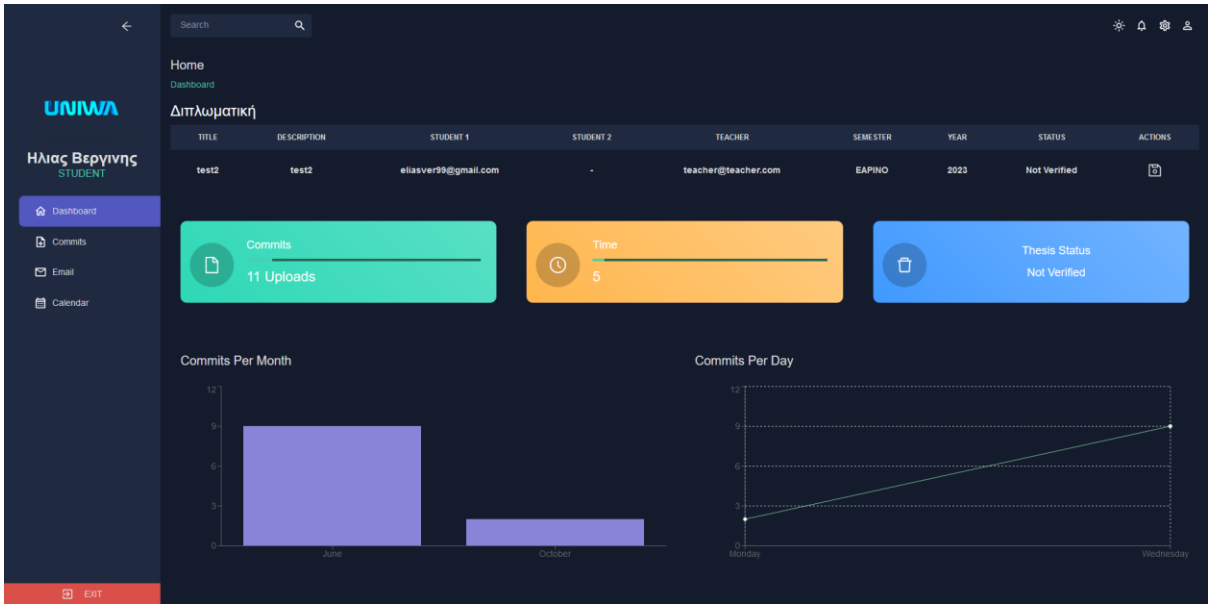


Εικόνα 5: Σελίδα εγγραφής χρήστη

Η σελίδα που είναι υπεύθυνη για την εγγραφή ενός χρήστη. Αυτή την φορά έχει να περισσότερα πεδία να συμπληρώσει ο χρήστης, καθώς χρειάζεται η επιβεβαίωση του κωδικού για την αποφυγή λαθών. Παρέχεται επίσης η δυνατότητα στον χρήστη να δει τον κωδικό που έβαλε για μεγαλύτερη ευκολία. Τέλος, έχει την δυνατότητα να κάνει εγγραφή είτε μέσω Google είτε μέσω Github.

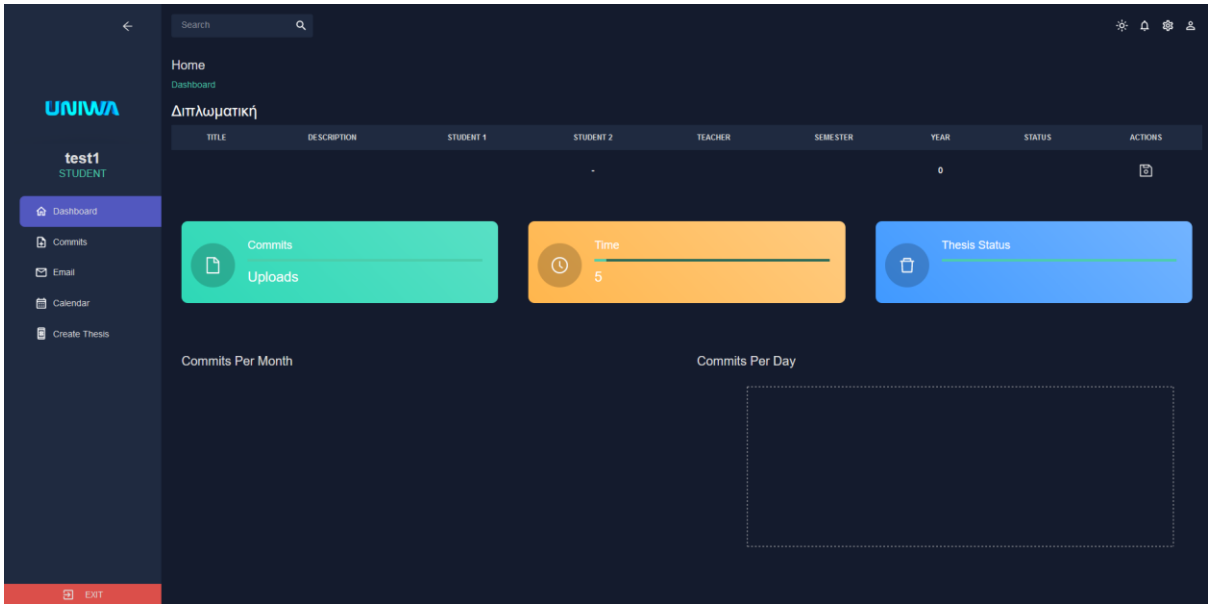
3.1.3.3 Dashboard Page

Μόλις ο χρήστης συνδεθεί στην πλατφόρμα, σε περίπτωση που έχει δεδομένα (όπως στο παράδειγμα που θα παρουσιαστεί), θα δει το dashboard, με κάποια δεδομένα σχετικά με την πρόοδο του. Πιο συγκεκριμένα



Εικόνα 6: Αρχική σελίδα εφαρμογής

Στην σελίδα αυτή παρατηρούμε αρχικά πως υπάρχει ένα μενού πλοήγησης στα αριστερά. Στο σημείο αυτό υπάρχει το logo της εφαρμογής, το όνομα του χρήστη που έχει συνδεθεί, καθώς και η ιδιότητα του (είτε διαχειριστής, είτε καθηγητής, είτε φοιτητής). Από κάτω υπάρχουν οι επιλογές πλοήγησης και η έξοδος από την εφαρμογή με τα κατάλληλα εικονίδια, για να είναι πιο κατανοητά. Στο περιεχόμενο της σελίδα υπάρχει η διπλωματική, με τον τίτλο της, μία μικρή περιγραφή, τους φοιτητές που συμμετέχουν σε αυτή, τον επιβλέποντα καθηγητή, το εξάμηνο που δηλώθηκε, το έτος και η κατάσταση αυτής. Από κάτω υπάρχουν κάποια γραφήματα, που χρησιμοποιούν κάποια δεδομένα για την καλύτερη διαχείριση της διπλωματικής. Προς το παρόν, υπάρχουν τα γραφήματα σχετικά με τους μήνες που ήταν ενεργός ο φοιτητής πάνω στην διπλωματική και ποιες μέρες ήταν πιο ενεργός.
Σε περίπτωση που ο χρήστης είναι καινούργιος και δεν έχει δηλωμένη διπλωματική εργασία, η σελίδα αυτή είναι ως εξής

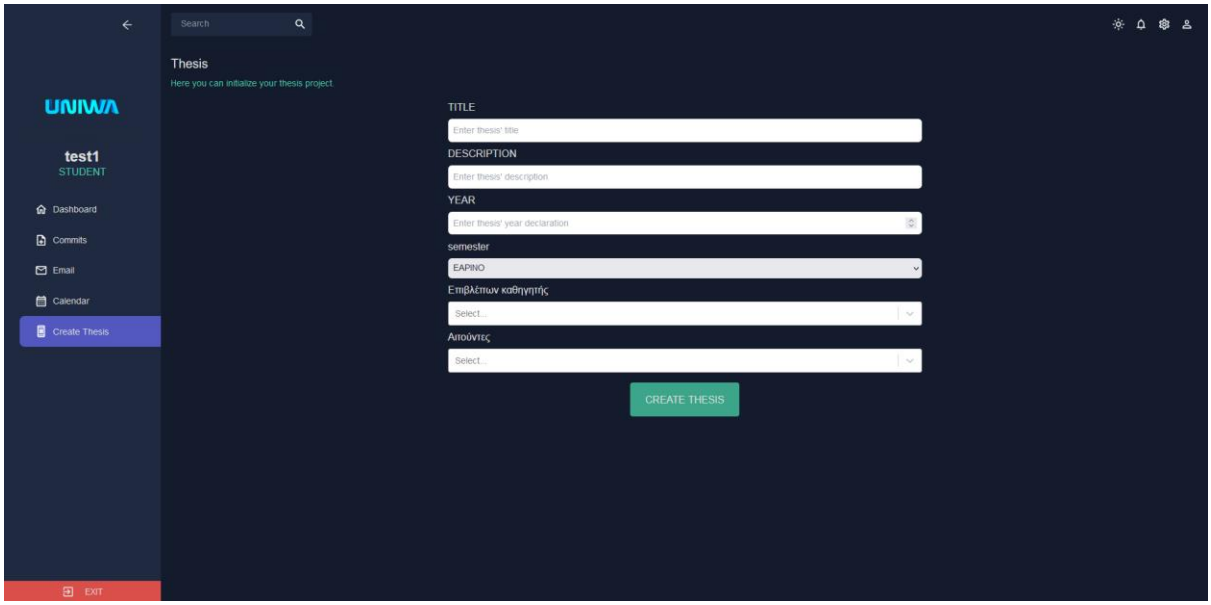


Εικόνα 7: Αρχική σελίδα χωρίς δεδομένα

Παρατηρούμε όμως, πως στο μενού τώρα υπάρχει άλλη μία επιλογή πλοήγησης, την δημιουργία διπλωματικής εργασίας.

3.1.3.4 Create Thesis Page

Η σελίδα αυτή είναι μία απλή φόρμα, με τα απαραίτητα πεδία προς συμπλήρωση για να δημιουργηθεί η διπλωματική εργασία.



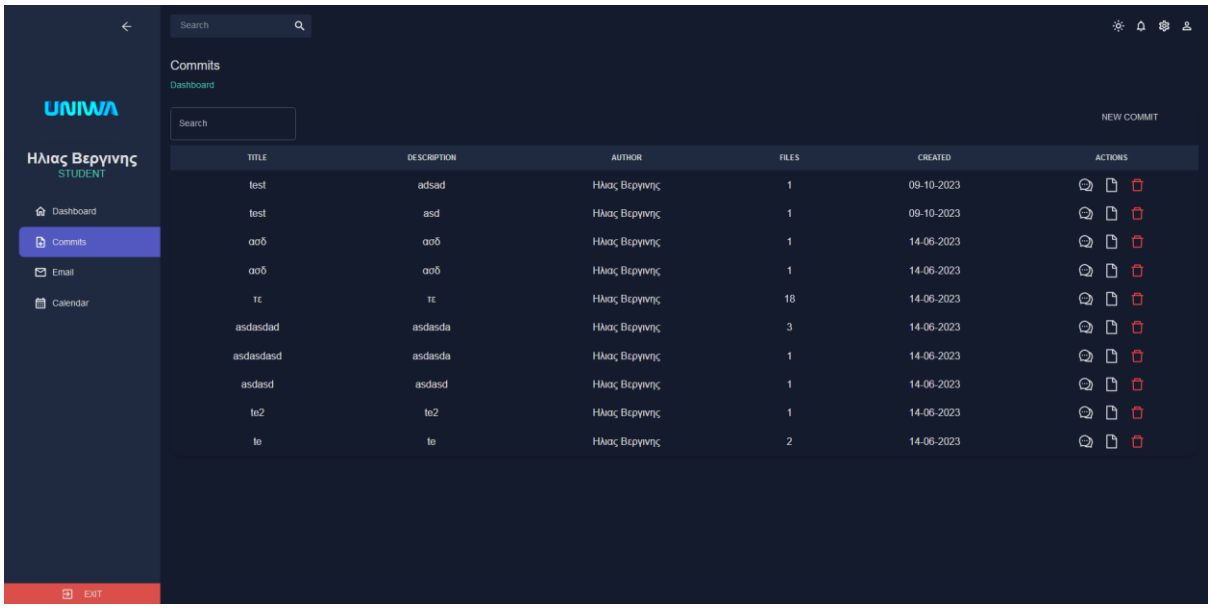
Εικόνα 8: Φόρμα δημιουργία διπλωματικής εργασίας

Στα πεδία αυτά του δίνεται η δυνατότητα να διαλέξει μέχρι έναν από τους διδάσκοντες που είναι εγγεγραμμένοι στην εφαρμογή, και μέχρι δύο φοιτητές. Σε περίπτωση που η δημιουργία επιτύχει τότε προβάλλεται στην οθόνη ανάλογο μήνυμα.

Εικόνα 9: Ειδοποίηση θετικού αποτελέσματος

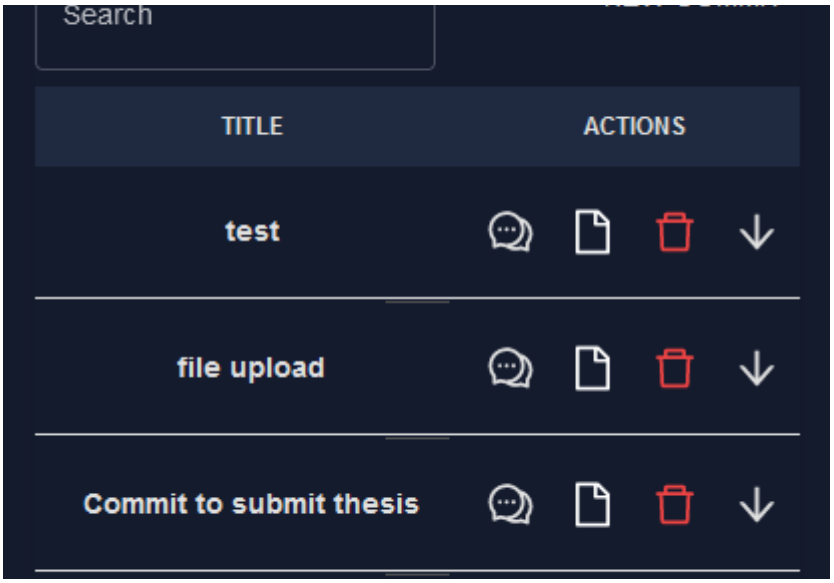
3.1.3.5 Commit Page

Σκοπός αυτής της σελίδας είναι να καταγράφεται η πρόοδος της διπλωματικής εργασίας. Ο φοιτητής δημιουργεί μία καταχώρηση, με όνομα τίτλο και μπορεί να επισυνάψει και αρχεία. Στην συνέχεια ενημερώνεται ο καθηγητής και ανοίγεται μία "συνομιλία", προκειμένου να αναπτυχθούν σχόλια και διορθώσεις. Σε περίπτωση που υπάρχουν τέτοιες καταχωρίσεις η σελίδα είναι ως εξής

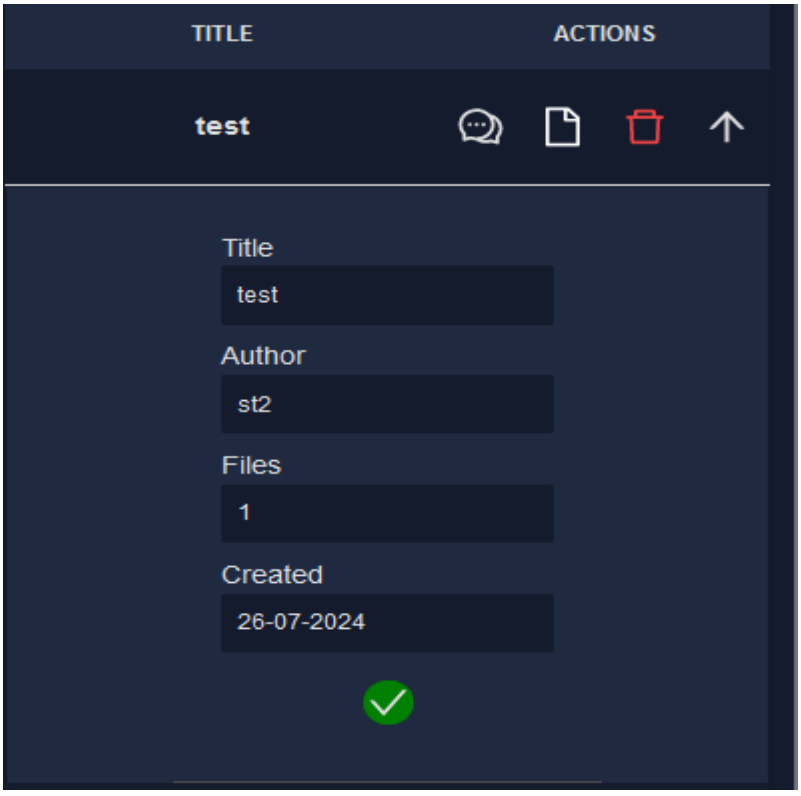


Εικόνα 10: Λίστα commits

Στην σελίδα αυτή υπάρχουν όλες οι καταχωρίσεις που έχει κάνει ο χρήστης, μαζί με τις απαραίτητες πληροφορίες. Η σελίδα αυτή έπρεπε να έχει αρκετές διαφοροποιήσεις ανάλογα με τις διαστάσεις της οθόνης, πιο συγκεκριμένα σε διαστάσεις κινητής συσκευής, γίνεται απόκρυψη πληροφοριών προκειμένου να μπορεί να προβληθεί. Δηλαδή η σελίδα παίρνει της εξής μορφή με μόνο τον τίτλο και τις ενέργειες που μπορεί να πραγματοποιήσει ο χρήστης

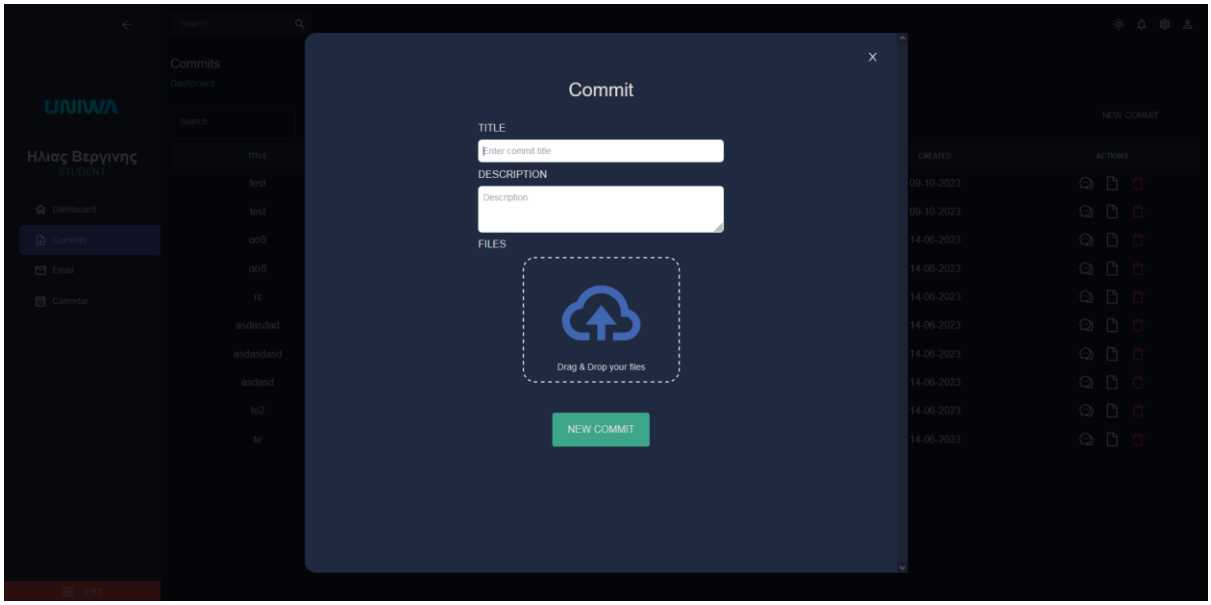


Στην περίπτωση αυτή προστίθεται και μία ακόμα ενέργεια, προκειμένου να ανοίξει η γραμμή και να προβληθούν όλες οι πληροφορίες.



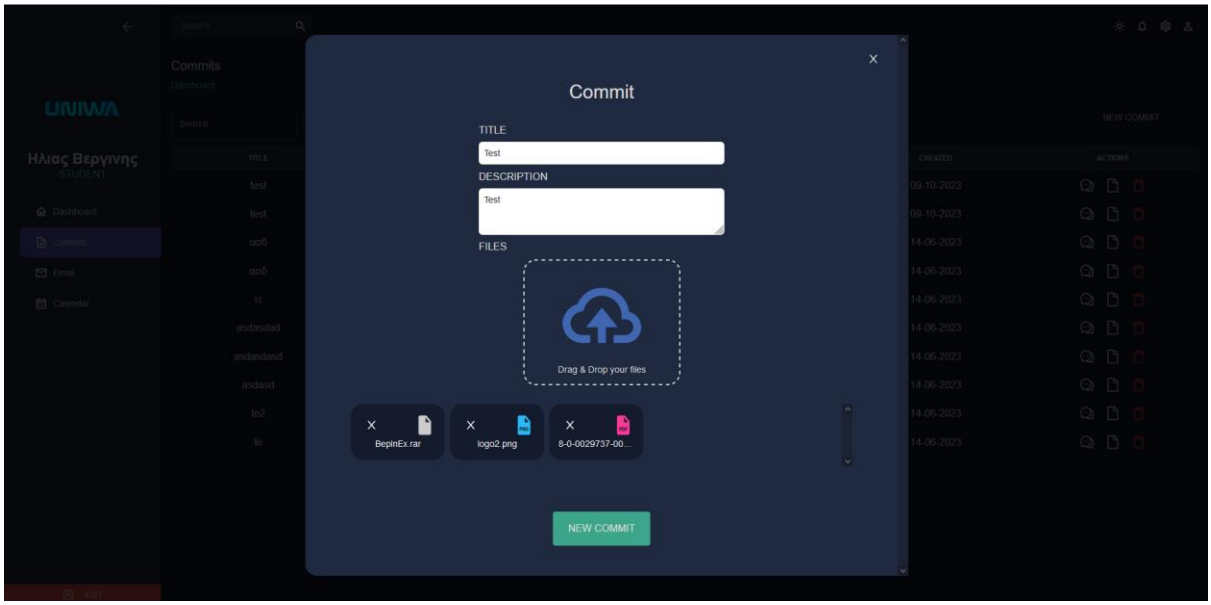
Εικόνα 11: Πίνακας κινητών επεκτείνομενος

Επιπλέον για κάθε καταχώριση, μπορεί να ανοίξει την συνομιλία με τον διδάσκοντα, να δει τα αρχεία που επισύναψε και τέλος να διαγράψει την καταχώριση που έκανε. Η δημιουργία μίας καταχώρισης γίνεται με την συμπλήρωσης μίας φόρμας.



Εικόνα 12: Φόρμα δημιουργίας commit

Όταν συμπληρωθεί η φόρμα παίρνει την εξής μορφή



Εικόνα 13: Συμπληρωμένη φόρμα commit

Ανάλογα με τον τύπο αρχείο αλλάζει και το εικονίδιο. Σε περίπτωση που το αρχείο είναι αγνώστου τύπου, τότε βάζει ένα default εικονίδιο. Σε περίπτωση επιτυχίας δημιουργίας προβάλλεται και το ανάλογο μήνυμα στον χρήστη

Εικόνα 14: Θετική ειδοποίηση δημιουργίας commit

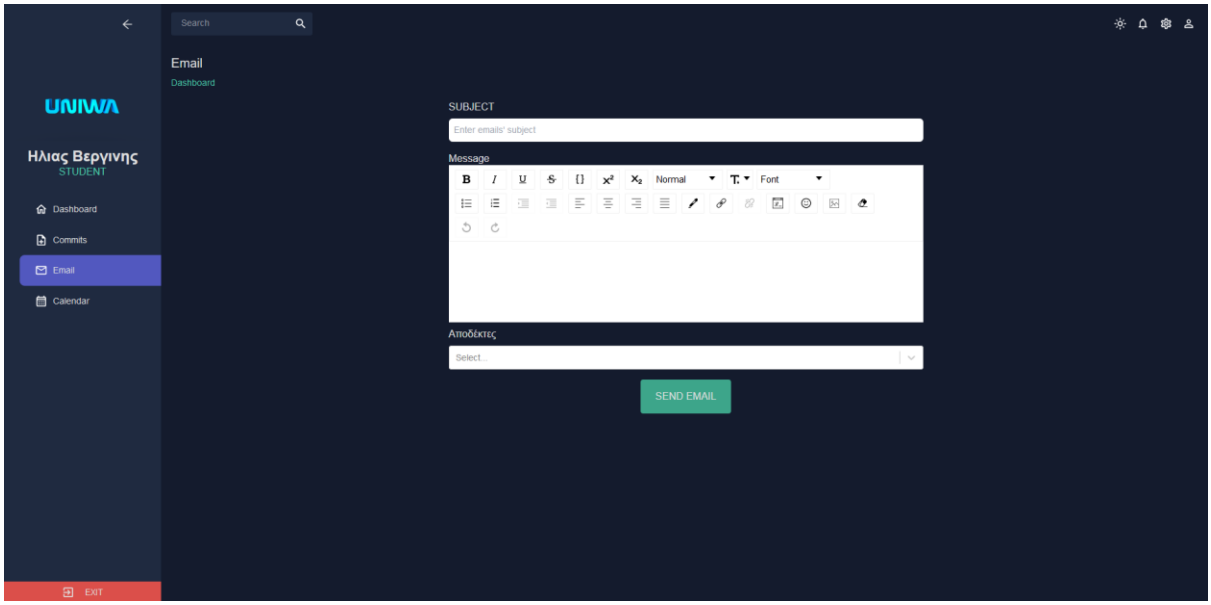
Το chatroom μεταξύ καθηγητή και φοιτητών έχει την εξής μορφή

Εικόνα 15: Ζωντανή επικοινωνία

Αξίζει να σημειωθεί πως τα μηνύματα έρχονται σε πραγματικό χρόνο, ωστόσο η υλοποίηση αυτού θα αναλυθεί σε επόμενο κεφάλαιο.

3.1.3.6 Email Page

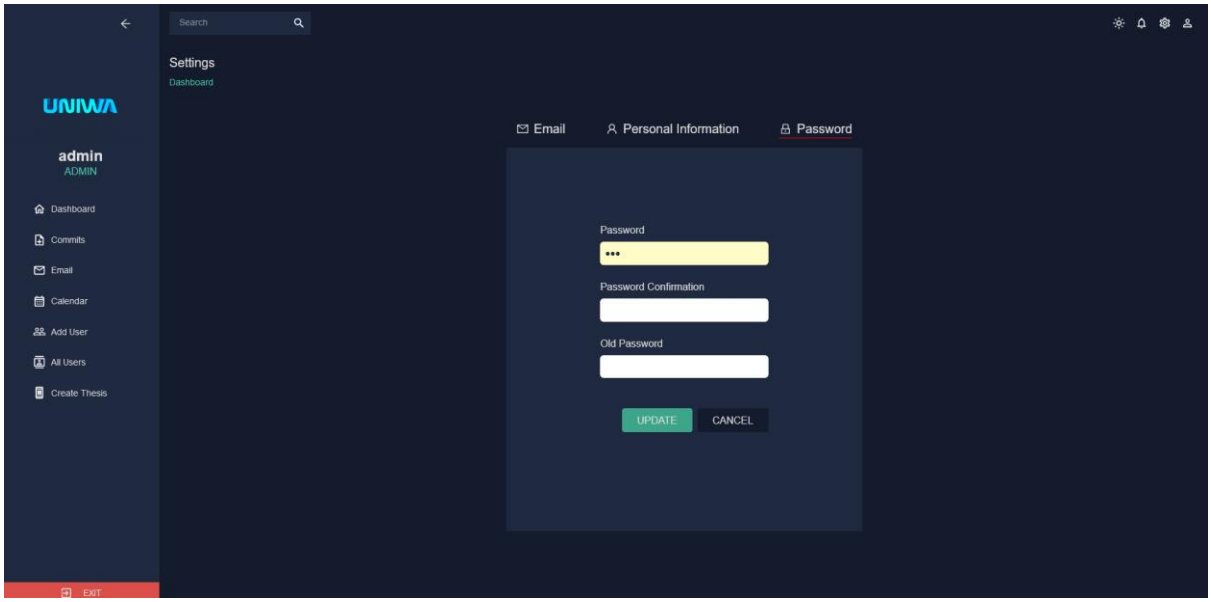
Οι χρήστες σε περίπτωση που δεν θέλουν να χρησιμοποιήσουν το chatroom, μπορούν να στείλουν email, συμπληρώνοντας την εξής φόρμα. Η φόρμα αυτή αποτελείται από το θέμα του μηνύματος, το μήνυμα και τους παραλήπτες αυτού



Εικόνα 16: Δημιουργία ηλεκτρονικού μηνύματος

3.1.3.7 Setting Page

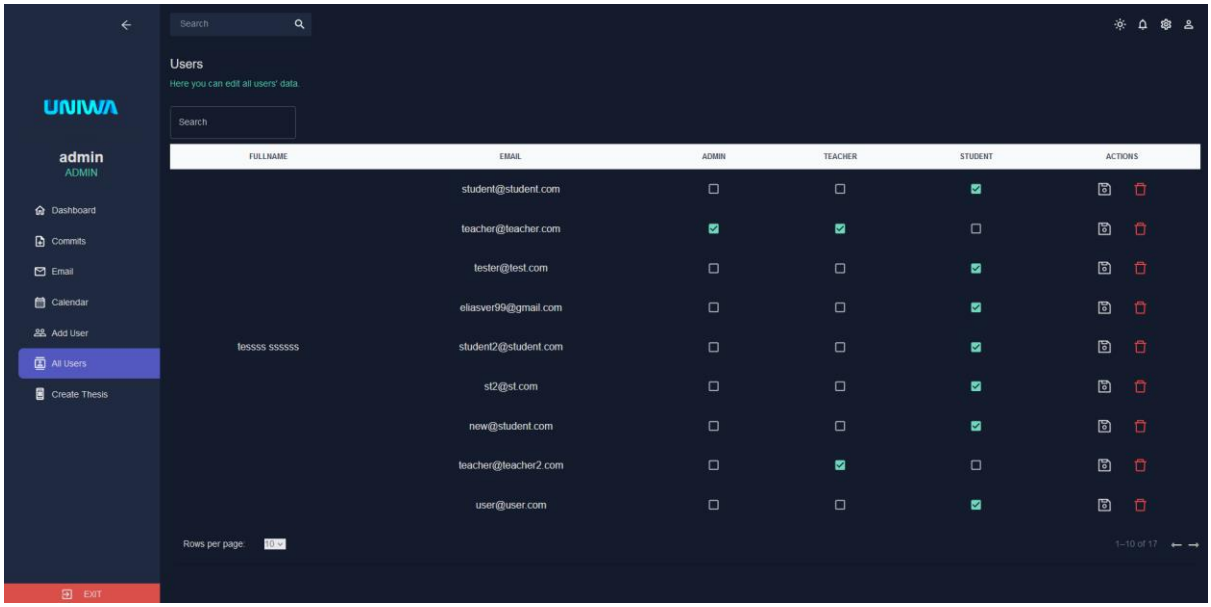
Η σελίδα αυτή αποτελείται από τρεις φόρμες ανάλογα με τις ρυθμίσεις που θέλει να κάνει ο χρήστης. Η πρώτη είναι για την αλλαγή του ηλεκτρονικού ταχυδρομίου με το οποίο συνδεέται στην εφαρμογή. Η δεύτερη είναι για αλλαγή προσωπικών στοιχείων και τέλος δίνεται και η επιλογή αλλαγής κωδικού στον χρήστη.



Εικόνα 17: Σελίδα ρυθμίσεων

3.1.3.8 Admin Page

Η σελίδα του διαχειριστή της εφαρμογής έχει παραπάνω λειτουργίες από αυτές ενός φοιτητή ή ενός καθηγητή. Αρχικά ο διαχειριστής έχει δικαίωμα να δημιουργήσει παραπάνω από μία διπλωματική εργασία. Επιπλέον, του παρέχεται η δυνατότητα να δημιουργήσει χρήστες (σε περίπτωση που κάποιος δεν μπορεί να δημιουργήσει τον λογαριασμό του), καθώς και να τροποποιήσει τους ρόλους ενός χρήστη. Αυτό γίνεται με τον εξής τρόπο

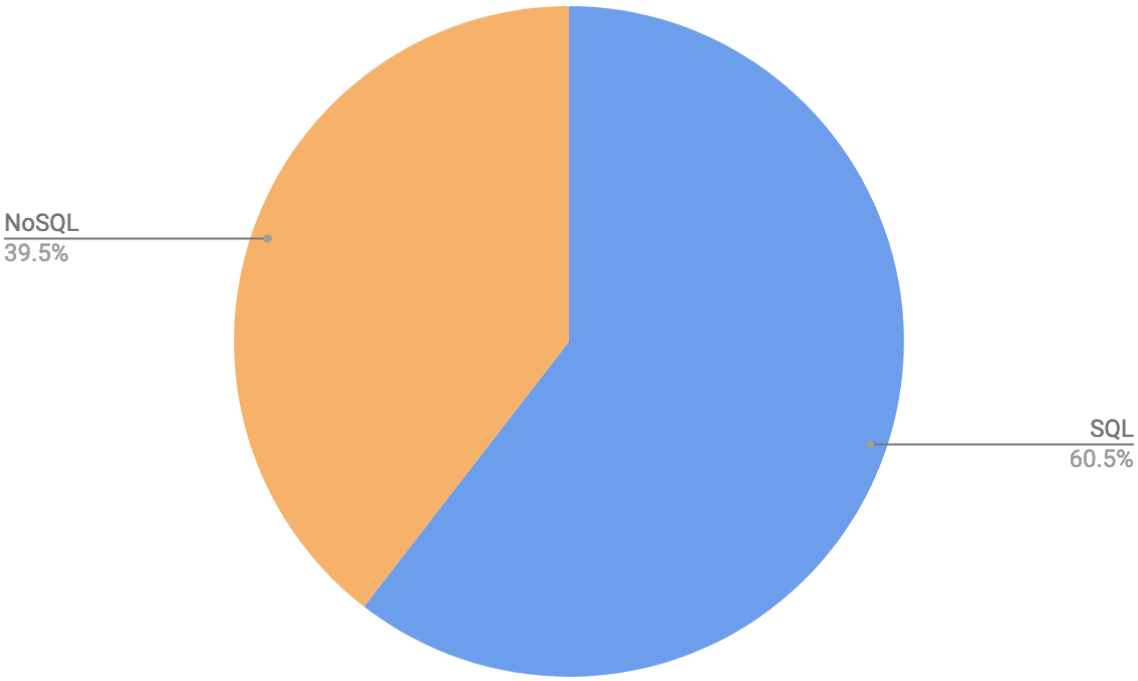


Εικόνα 18: Σελίδα διαχειριστή, επεξεργασία χρηστών

3.2 Βάση Δεδομένων

3.2.1 Είδη Βάσεων

Οι δύο κυριότερες κατηγορίες βάσεων είναι οι σχεσιακές βάσεις δεδομένων(SQL) και οι μη σχεσιακές βάσεις δεδομένων (NoSQL). Η γλώσσα προγραμματισμού SQL, ή αλλιώς Structured Query Language, δημιουργήθηκε από τους Donald D. Chamberlin και Raymond F. Boyce στις αρχές της δεκαετίας του 70', ενώ η NoSQL, ή αλλιώς Non Structured Query Language, δημιουργήθηκε το 1998 από τον Carl Strozz. Μπορεί ο σκοπός τους να είναι ο ίδιος, ωστόσο έχουν αρκετές διαφορές, καθώς και πολλούς υποστηρικτές η κάθε μία από αυτές. Χαρακτηριστικό είναι το στατιστικό για την προτίμηση των προγραμματιστών με βάση έρευνας που έγινε το 2019.



Εικόνα 19: Προτιμήσεις βάσεων

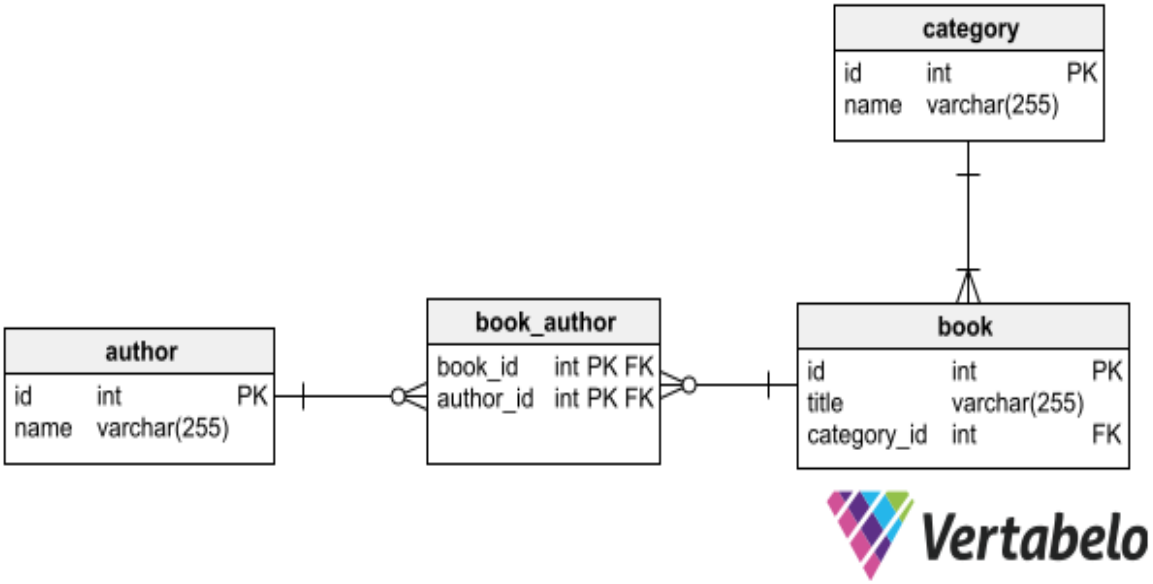
Όπως γίνεται ξεκάθαρο και από την εικόνα, υπάρχει μία ελαφριά κλίση προς την χρήση της SQL. Ωστοσο για την υλοποιήση αυτής της διπλωματικής οφείλαμε να κάνουμε έρευνα προκειμένου να δούμε τις διαφορές και να καταλήξουμε σε μία από τις δύο κατηγορίες.

3.2.2 Διαφορές μεταξύ SQL και NoSQL

Οι διαφορές μεταξύ των βάσεων δεδομένων είναι οι εξής:

- Η SQL δημιουργεί μία σχεσιακή βάση, ενώ η NoSQL όχι. Πιο αναλυτικά τα tables που δημιουργούνται στην sql έχουν σχέσεις εξάρτησεις μεταξύ τους (many-to-many,one-to-many etc). Η NoSQL από την άλλη δημιουργεί μοντέλα και οντότητες, επομένως αυτή η σχέση στην ουσία κρύβεται σε ένα εμφωλευμένο αντικείμενο. Για παράδειγμα, έστω ότι θέλουμε να γράψουμε μία βάση που να έχει συγγραφείς και τα βιβλία τους. Όπως είναι λογικό ένας συγγραφέας μπορεί να έχει γράψει πολλά βιβλία.

Επίσης όμως ένα άρθρο μπορεί να έχει γραφτεί από παραπάνω από έναν συγγραφέα. Ας δούμε πως αυτό θα μπορούσε να υλοποιηθεί σε SQL



Εικόνα 20: Many-to-Many σχέση

Παρατηρούμε πως η σχεσιακή βάση αποτελείται από 4 tables. Το author είναι ο συγγραφέας, το book είναι το βιβλίο και το category είναι η κατηγορία του βιβλίου. Παρατηρούμε πως το category_id υπάρχει μέσα στο table book ως foreign key. Αυτό μας δείχνει πως ένα βιβλίο έχει μία κατηγορία. Ωστόσο για μία σχέση many-to-many, πρέπει να υλοποιηθεί ένα table μεσάζον (book_author), που κάνει δηλώνει πως πολλά βιβλία έχουν πολλούς συγγραφείς.

Τώρα θα δούμε και μία υλοποίηση σε NoSQL. Αρχικά θα χρειαστούμε τρία μοντέλα, ένα για τον συγγραφέα, ένα για το βιβλίο και ένα για την κατηγορία

- Η Σχεσιακή βάση υποστηρίζει join μεταξύ των tables της, ενώ η μη σχεσιακή όχι
- Διαφορά στον τρόπο scaling

Ωστόσο, στο συγκεκριμένο σύστημα χρησιμοποιήθηκε κάτι πιο υβριδικό. Πιο συγκεκριμένα, μπορεί η βάση δεδομένων να είναι sql, ωστόσο με χρήση της βιβλιοθήκη sequelize. Στην ουσία μας παρέχει κάποιες συναρτήσεις για να γίνεται πιο εύκολο το σχεσιακό μοντέλο.

3.2.3 Βασικές αρχές

Η βάση δεδομένων αποτελεί την καρδιά της εφαρμογής, αφού είναι υπεύθυνη για την διατήρηση και την δημιουργία όλων των δεδομένων. Η δημιουργία της βάσης δεδομένων είναι μια ιδιαίτερα πολύπλοκη διαδικασία, καθώς πρέπει ο σχεδιασμός της να γίνει με τον καλύτερο δυνατό τρόπο για να αποφευχθούν μελλοντικά προβλήματα. Όταν γίνεται ο σχεδιασμός μίας βάσης πρέπει να έχουμε κατά νου τα εξής:

- Επαναληπτικότητα: Πρέπει όσο το δυνατόν περισσότερο να μην υπάρχουν επαναλήψεις. Οι επαναλήψεις μπορούν να δημιουργήσουν σοβαρό πρόβλημα στην βάση κυρίως κατά την διάρκεια εγγραφής. Ένα χαρακτηριστικό παράδειγμα είναι:

3.3 Υλοποίηση Βάσης

Όπως αναφέραμε και σε προηγούμενο κεφάλαιο, η βάση δημιουργήθηκε με χρήση της βιβλιοθήκης sequelize. Ας δούμε πως αρχικοποιείται η βάση, τα μοντέλα και στην συνέχεια τις σχέσεις μεταξύ αυτών.

3.3.1 Αρχικοποίηση βάσης

Με την χρήση του framework της Express, πρέπει να δημιουργήσουμε μόνη μας την σύνδεση με την βάση, μόλις ξεκινάει να λειτουργεί ο σέρβερ. Στον κώδικα μας για λόγους ασφάλειας και καλύτερης διαχείρισης έχουν δημιουργηθεί κάποιες μεταβλητές περιβάλλοντος για αυτό τον σκοπό, οι οποίες στην ουσία δίνουν αρχικές τιμές στην σύνδεση. Αυτές οι μεταβλητές είναι οι εξής:

```
DB_HOST= 127.0.0.1
DB_USER= root
DB_PASS=
DB_DIALECT= mysql
DB_DATABASE= thesis
```

Εικόνα 21: Μεταβλητές Περιβάλλοντος

Ορίζουμε τον host με την διεύθυνση ip του δικτύου μας, μετά βάζουμε τον user και το συνθηματικό (για απλότητα είναι κενό), θέτουμε το είδος βάσης και το όνομα του database. Μετά δημιουργούμε ένα config αρχείο για να είναι πιο εύκολη η επεξεργασία τυχόν αλλαγών στην βάση το οποίο επιστρέφει όλες τις μεταβλητές περιβάλλοντος που αφορούν την βάση, καθώς και κάποιες τιμές σχετικά με τις παράλληλες συνδέσεις στην βάση.

```
module.exports = {
  HOST: process.env.DB_HOST,
  USER: process.env.DB_USER,
  PASSWORD: process.env.DB_PASS,
  DB: process.env.DB_DATABASE,
  dialect: "mysql",
  logging: false,
  pool: {
    max: 12,
    min: 0,
    acquire: 30000,
    idle: 10000,
  },
};
```

Εικόνα 22: Αρχικοποίηση δεδομένων βάσης

Τελευταίο βήμα για την σύνδεση της βάσης είναι να δημιουργηθεί ένα αντικείμενο της βιβλιοθήκης. Με χρήση των μεταβλητών για την βάση, θέτουμε τα όρια της βάσης, που θα σηκωθεί, καθώς και το είδος

```
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  operatorsAliases: false,
  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle,
  },
});
```

Εικόνα 23: Αρχικοποίηση βιβλιοθήκης sequelize

3.3.2 Μοντέλο Χρήστη

```
module.exports = (sequelize, Sequelize) => {
  const User = sequelize.define("user", {
    username: {
      type: Sequelize.STRING,
    },
    fname: {
      type: Sequelize.STRING,
    },
    lname: {
      type: Sequelize.STRING,
    },
    email: {
      type: Sequelize.STRING,
    },
    password: {
      type: Sequelize.STRING,
    },
    googleId: {
      type: Sequelize.STRING,
      allowNull: true,
    },
    githubId: {
      type: Sequelize.STRING,
      allowNull: true,
    },
    color: {
      type: Sequelize.STRING,
      allowNull: true,
    },
  });

  return User;
};
```

Εικόνα 24: Μοντέλο Χρήστη

Το μοντέλο του χρήστη αποτελείται από τα εξής:

- Username: το ψευδώνυμο του χρήστη στην εφαρμογή
- Fname: το μικρό όνομα του χρήστη
- Lname: το επώνυμο του χρήστη
- Email: το email του χρήστη
- Password: το συνθηματικό του χρήστη για να μπορεί να συνδεθεί στην εφαρμογή
- googleId: ένας μοναδικός αριθμός που δίνεται στον χρήστη για να γίνεται αναγνώριση σε περίπτωση που συνδεθεί με την χρήση του Google.

- githubId: ένας μοναδικός αριθμός που δίνεται στον χρήστη για να γίνεται αναγνώριση σε περίπτωση που συνδεθεί με την χρήση του Github.
- Color: χρώμα του χρήστη που χρησιμοποιείται σε για το avatar του.

Παρατηρούμε πως το μοντέλο του χρήστη δεν είναι ιδιαίτερα περίπλοκο, καθώς αποτελείται από τις βασικές ιδιότητες και όλες είναι τύπου συμβολοσειράς. Ωστόσο επίσης παρατηρούμε πως λείπει ο ρόλος του. Αυτό γίνεται σε ξεχωριστό μοντέλο για να βοηθηθεί η επεκταστιμότητα της εφαρμογής. Στο τέλος θα δούμε όλη την βάση και θα γίνει ακόμα περισσότερο κατανοητό πως επιτυγχάνεται αυτό.

3.3.3 Μοντέλο Ρόλων

```
module.exports = (sequelize, Sequelize) => {      unknown, 9 months ago • login signup
  const Role = sequelize.define("roles", {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
    },
    name: {
      type: Sequelize.STRING,
    },
  });

  return Role;
};
```

Εικόνα 25: Μοντέλο Ρόλων

Το μοντέλο των ρόλων θα μπορούσε να χαρακτηριστεί και βοηθητικό, αφού δεν περιέχει τίποτα παραπάνω πέρα από ένα μοναδικό αριθμό και το όνομα του ρόλου. Στην παρούσα περίπτωση οι ρόλοι είναι οι εξής (admin,teacher και student).
Αποτελείται από τα εξής πεδία:

- Id: το αναγνωριστικό του μοντέλου
- Name: το όνομα του ρόλου.

3.3.4 Μοντέλο Διπλωματικής Εργασίας

```
module.exports = (sequelize, Sequelize) => {
  const Project = sequelize.define("project", {
    title: {
      type: Sequelize.STRING,
    },
    description: {
      type: Sequelize.STRING,
      allowNull: true,
    },
    status: {
      type: Sequelize.STRING,
      defaultValue: "Not Verified",
    },
    semester: {
      type: Sequelize.STRING,
      defaultValue: "ΧΕΙΜΕΡΙΝΟ",
    },
    year: {
      type: Sequelize.INTEGER,
    },
  });

  return Project;
};
```

Εικόνα 26: Μοντέλο διπλωματικής Εργασίας

Μετά τον χρήστη το πιο κομβικό μοντέλο είναι αυτό των εργασιών. Ωστόσο το συγκεκριμένο μοντέλο έχει πολλές σχέσεις με τα μοντέλα που θα αναπτυχθούν σε επόμενα κεφάλαια, επομένως οι λειτουργίες του θα γίνουν αντιληπτές αργότερα. Το μοντέλο αυτό έχει τις εξής ιδιοτήτες:

- Title: ο τίτλος της εργασίας που είναι μία συμβολοσειρά
- Description: μία σύντομη περιγραφή της διπλωματικής εργασίας, που μπορεί να μην υπάρχει κίολας
- Status: η κατάσταση της διπλωματικής εργασίας
- Semester: το εξάμηνο που ξεκινάει η διπλωματική εργασία, είτε χειμερινό είτε εαρινό. Η default τιμή είναι το χειμερινό.
- Year: η χρονιά που δηλώθηκε η διπλωματική εργασία.

4.3.5 Μοντέλο Commit

```
unknown, 8 months ago | 1 author (unknown)
module.exports = (sequelize, Sequelize) => {
  const Commit = sequelize.define("commit", {
    title: {
      type: Sequelize.STRING,
    },
    description: {
      type: Sequelize.STRING,
      allowNull: true,
    },
  });

  return Commit;
};      unknown, 8 months ago • commits

```

Εικόνα 27: Μοντέλο Commit

Το commit σχετίζεται με το μοντέλο των εργασίων. Δηλαδή μία εργασία περιέχει πολλά commit, τα οποία μπορεί να μην περιέχουν πολύ πληροφορία, καθώς μόνο τα απαραίτητα πεδία είναι ο τίτλος, η περιγραφή και τα αρχεία. Ωστόσο τα αρχεία, επείδη συνδέονται με το μοντέλο αυτό και αποτελούν μία σχέση one to many, δεν υπάρχει κάποιο πεδίο στο μοντέλο αυτό.
Αποτελείται από τα εξής πεδία:

- Title: μία συμβολοσειρά που δηλώνει τον τίτλο
- Description: μία συμβολοσειρά που δηλώνει μία σύντομη περιγραφή.

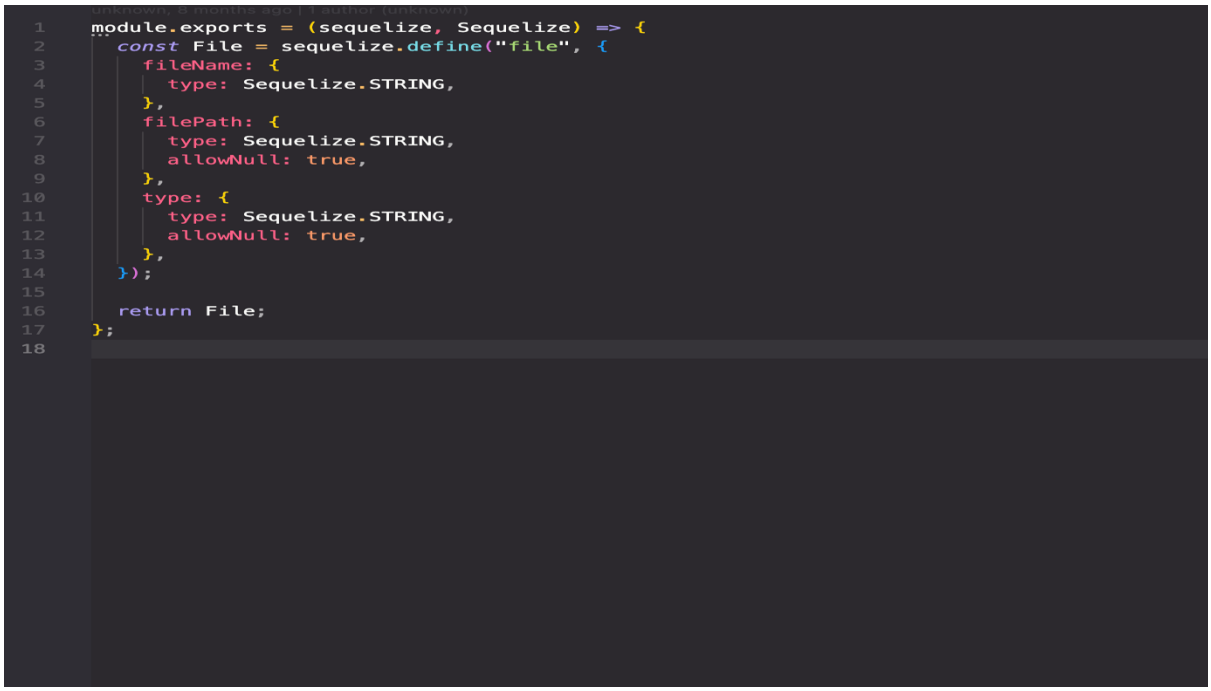
3.3.6 Μοντέλο Σχολίων

```
unknown, 7 months ago | 1 author (unknown)
module.exports = (sequelize, Sequelize) => {
  const Comment = sequelize.define("comment", {
    message: {
      type: Sequelize.STRING,
    },
    visible: {
      type: Sequelize.BOOLEAN,
      allowNull: true,
    },
  });

  return Comment;
};
```

Εικόνα 28: Μοντέλο σχολίων

Το μοντέλο των σχολίων σχετίζεται με το μοντέλο των commit. Τα μόνα πεδία που είναι απαραίτητα είναι το μήνυμα και αν είναι ορατό σε όλους. Το visible πεδίο παρέχει την δυνατότητα στον καθηγητή να έχει και κρυφά σχόλια.

3.3.7 Μοντέλο Αρχείων



```
unknown, 8 months ago | 1 author (unknown)
module.exports = (sequelize, Sequelize) => {
  const File = sequelize.define("file", {
    fileName: {
      type: Sequelize.STRING,
    },
    filePath: {
      type: Sequelize.STRING,
      allowNull: true,
    },
    type: {
      type: Sequelize.STRING,
      allowNull: true,
    },
  });

  return File;
};
```

Εικόνα 29: Μοντέλο αρχείων

Το μοντέλο των αρχείων σχετίζεται με των commit. Το βασικότερο που χρειάζεται από τα πεδία είναι το filepath,για να έχει την δυνατότητα ο εξυπηρετητής, να δώσει στον πελάτη το αρχείο όταν αυτός το ζητήσει.

3.4 Σχεσιακά μοντέλα

Τα μοντέλα που δείξαμε στα προηγούμενα κεφάλαια δεν έχουν καμία υπόσταση χωρίς τις σχέσεις μεταξύ αυτών. Πιο συγκεκριμένα, το μοντέλο των εργασιών θα έχανε την πλειοψηφία των λειτουργιών του χωρίς τις σχέσεις εξάρτησης με τα commits. Παρακάτω παραθέτουμε τον κώδικα με τις συνδέσεις μεταξύ των μοντέλων και ακολουθεί ο σχολιασμός τους.

```
db.role.belongsToMany(db.user, {
  through: "user_roles",
  foreignKey: "roleId",
  otherKey: "userId",
});
db.user.belongsToMany(db.role, {
  through: "user_roles",
  foreignKey: "userId",
  otherKey: "roleId",
});

db.user.hasMany(db.project, {
  foreignKey: "teacherId",
});

db.project.belongsTo(db.user, { foreignKey: "teacherId", as: "teacher" });

db.project.belongsToMany(db.user, {
  through: "student_project",
  foreignKey: "projectId",
  otherKey: "studentId",
  as: "students",
});

db.user.belongsToMany(db.project, {
  through: "student_project",
  foreignKey: "studentId",
  otherKey: "projectId",
  as: "project",
});

db.user.hasMany(db.commit, {
  foreignKey: "userId",
});

db.commit.belongsTo(db.user, { foreignKey: "userId", as: "user" });

db.project.hasMany(db.commit, {
  foreignKey: "projectId",
});
db.commit.belongsTo(db.project, { foreignKey: "projectId", as: "project" });

db.commit.hasMany(db.file, {
  foreignKey: "commitId",
  as: "files",
```

Εικόνα 30: Σχεσιακά μοντέλα

```
75  db.file.belongsTo(db.commit, {
76    foreignKey: "commitId",
77    as: "commit",
78  });
79  db.comment.belongsTo(db.user, { foreignKey: "senderId", as: "sender" });
80  db.comment.belongsToMany(db.user, {
81    through: "receiver_comment",
82    foreignKey: "commentId",
83    otherKey: "receiverId",
84    as: "receivers",
85  });
86  db.commit.hasMany(db.comment, {
87    foreignKey: "commitId",
88    as: "comments",
89    onDelete: "CASCADE",
90  });
91  db.comment.belongsTo(db.commit, {
92    foreignKey: "commitId",
93    as: "commit",
94  });
95  db.ROLES = ["student", "admin", "teacher"];
96
97  module.exports = db;
98
```

Εικόνα 31: Σχεσιακά μοντέλα 2

Στην ουσία αυτός είναι ο τρόπος που το πακέτο Sequelize δημιουργεί τα σχεσιακά μοντέλα. Αρχικά, γνωρίζουμε πως ένας χρήστης μπορεί να έχει πολλούς ρόλους (πχ ένας καθηγητής μπορεί να είναι και διαχειριστής). Επομένως, μιλάμε για μία σχέση many-to-many. Από την γραμμή 28 έως και την γραμμή 37 υλοποιείται αυτό, με χρήση pivot table, το οποίο ονομάζεται user_roles και η δομή του είναι η εξής

Στην συνέχεια δημιουργούμε την σχέση μεταξύ χρήστη και εργασίας. Τα προαπαιτούμενα για αυτή την σχέση είναι τα εξής:

- Ένας χρήστης με ρόλο καθηγητή μπορεί να έχει παραπάνω από μία εργασία
- Μία εργασία δεν μπορεί να έχει παραπάνω από έναν επιβλέποντα καθηγητή
- Μία εργασία μπορεί να έχει έως και 2 χρήστες
- Ένας χρήστης με ρόλο μαθητή μπορεί να έχει μέχρι μία εργασία.

Επομένως, για αρχή δημιουργούμε μία σχέση one-to-many, για τον καθηγητή και την εργασία (σειρές 39-41). Στην συνέχεια δηλώνουμε πως η εργασία ανήκει σε έναν καθηγητή (σειρά 43). Μετά πρέπει να δηλώσουμε τους φοιτητές, που η σχέση τους δηλώνεται στις σειρές 45 μέχρι 57. Όπως έγινε με τους χρήστες και τους ρόλους, έτσι ακριβώς λειτουργούμε και με τους χρήστες και την εργασία. Δηλαδή ένας χρήστης ανήκει σε μία εργασία, αλλά σε μία εργασία μπορεί να είναι πολλοί χρήστες.

Αφού ολοκληρώθηκαν και οι σχέσεις μεταξύ χρηστών και εργασιών, εστιάζουμε μόνο στις εργασίες πλέον. Για τις ανάγκες της παρούσας διπλωματικής εργασίας οι απαιτήσεις από τις εργασίες ήταν οι εξής:

- Μία εργασία θα έχει πολλά commits από φοιτητές
- Το κάθε commit θα μπορεί να έχει κανένα ή και παραπάνω αρχεία
- Σε κάθε commit θα πρέπει να υπάρχουν κανένα ή παραπάνω σχόλια από χρήστες (είτε φοιτητές είτε τον καθηγητή).

Για αρχή πρέπει να δηλωθεί πως τα commit ανήκουν σε έναν χρήστη. Αυτό γίνεται στην σειρά 59-61, που λέμε πως ένας χρήστης έχει πολλά commits, και στην σειρά 63 που

δηλώνουμε πως το commit ανήκει σε έναν χρήστη. Αυτή η σχέση γίνεται με χρήση foreign key το id του χρήστη. Στην συνέχεια δηλώνουμε πως μία εργασία έχει πολλα commits και ότι τα commits ανήκουν στην εργασία (σειρές 65-68). Στην συνέχεια δημιουργούμε την σχέση των commit με τα αρχεία (σειρές 70-78). Όπως λειτουργήσαμε με τα commit, ανάλογη διαδικασία ακολουθούμε και με τα σχόλια στις σειρές 79-94.

3.5 Τελικό αποτέλεσμα MySQL

Όλες αυτές οι σχέσεις που αναλύθηκαν σε προηγούμενο κεφάλαιο, δημιούργησαν τα tables στην βάση δεδομένων. Στην ουσία μπορεί εμείς να μην ορίσαμε πουθενά ξένα κλειδιά, αλλά το έκανε μόνο του το sequelize, μέσω των σχέσεων που δημιουργήθηκαν. Πιο αναλυτικά, το table των χρηστών

Όνομα	Τύπος	Null	Κλειδί
id	Int(11)	ΟΧΙ	κύριο
username	Varchar(255)	ΝΑΙ	-
email	Varchar(255)	ΝΑΙ	-
password	Varchar(255)	ΝΑΙ	-
googleId	Varchar(255)	ΝΑΙ	-
githubId	Varchar(255)	ΝΑΙ	-
color	Varchar(255)	ΝΑΙ	-
fname	Varchar(255)	ΝΑΙ	-
lname	Varchar(255)	ΝΑΙ	-

Πίνακας 1 Πίνακας χρηστών

Παρατηρούμε πως το συγκεκριμένο table δεν έχει ξένα κλειδιά ωστόσο ενώνεται με άλλα tables. Για παράδειγμα ας δούμε το table με τους ρόλους και πως ενώνεται με τους χρήστες.

Όνομα	Τύπος	Null	Κλειδί
id	Int(11)	ΟΧΙ	Κύριο
name	Varchar(255)	ΝΑΙ	-

Πίνακας 2 Πίνακας Ρόλων

Τώρα αυτά τα δύο tables, συνδέονται μέσως ενός τρίτου (user_roles). Αυτό είναι απαραίτητο, καθώς ο κάθε χρήστης μπορεί να έχει παραπάνω από έναν ρόλο. Σε αντίθετη περίπτωση στο table των χρηστών μπορούσαμε να έχουμε το roleId, σαν ξένο κλειδί και να γίνει η σχέση ένα προς ένα.

Όνομα	Τύπος	Null	Κλειδί

roleId	Int(11)	OXI	Κύριο
userId	Int(11)	OXI	Κύριο - Ξένο

Πίνακας 3 Συνδετικός πίνακας, χρήστη -ρόλου

Για να γίνει πιο εύκολα αντιληπτό πως λειτουργεί αυτή η σχέση ας δούμε κάποιες εγγραφές του. Ας υποθέσουμε πως ο χρήστης με ID 12 είναι ένας καθηγητής με όνομα Γιάννης Ιωάννου. Επίσης έστω πως το id 2 δηλώνει τον ρόλο διαχειριστή και το id 3 δηλώνει τον ρόλο του καθηγητή. Ωστόσο μπορεί να θέλουμε να μην έχει ιδιότητες μόνο ως καθηγητής, αλλά και ως διαχειριστής της εφαρμογής. Επομένως στην βάση περνάνε οι εξής δύο εγγραφές στο table user_roles.

roleId	userId
2	12
3	12

Πίνακας 4 Παράδειγμα πίνακα χρήστη-ρόλου

Ακολουθεί το βασικό table που είναι αυτό των εργασιών, καθώς έχει πολλές σχέσεις για να επιτευχθεί η λειτουργικότητα που θέλαμε.

Όνομα	Τύπος	Null	Κλειδί
Id	Int(11)	OXI	Κύριο
Title	Varchar(255)	NAI	-
Description	Description(255)	NAI	-
Status	Varchar(255)	NAI	-
teacherId	Int(11)	NAI	Ξένο
semester	Varchar(255)	NAI	-
year	Int(11)	NAI	-

Πίνακας 5 Πίνακας Διπλωματικής Εργασίας

Η σχέση που περιγράφεται σε αυτό το table μέσω του ξένου κλειδιού είναι πως κάθε εργασία ανήκει σε έναν καθηγητή. Με ανάλογο τρόπο με τους ρόλους, δηλαδή με δημιουργία ενδιάμεσου table, δημιουργείται η σχέση των μαθητών με την εργασία τους, στο table student_project.

Όνομα	Τύπος	Null	Κλειδί
roleId	Int(11)	OXI	Κύριο
userId	Int(11)	OXI	Κύριο - Ξένο

Πίνακας 6 Σχεσιακός Πίνακας μαθητή-εργασίας

Στην συνέχεια έχουμε τα commits που ανήκουν σε χρήστη και σε εργασία ταυτόχρονα.

Όνομα	Τύπος	Null	Κλειδί
Id	Int(11)	OXI	κύριο
Title	Varchar(255)	NAI	-

Description	Varchar(255)	NAI	-
userId	Int(11)	NAI	Ξένο
projectId	Int(11)	NAI	Ξένο

Πίνακας 7 Πίνακας Commit

Επιπλέον, όμως κάθε commit έχει και κάποια σχόλια, που ανήκουν και αυτά σε χρήστες .

Όνομα	Τύπος	Null	Κλειδί
Id	Int(11)	OXI	κύριο
message	Varchar(255)	NAI	-
visible	Tinyint(1)	NAI	-
senderId	Int(11)	NAI	Ξένο
commitId	Int(11)	NAI	Ξένο

Πίνακας 8 Πίνακας Σχολίων

Δίνεται επίσης η δυνατότητα σε κάθε commit ο χρήστης να επισυνάπτει και αρχεία, τα οποία μπαίνουν στην βάση ως εξής

Όνομα	Τύπος	Null	Κλειδί
Id	Int(11)	OXI	κύριο
filename	Varchar(255)	NAI	-
filepath	Varchar(255)	NAI	-
type	Varchar(255)	NAI	Ξένο
commitId	Int(11)	NAI	Ξένο

Πίνακας 9 Πίνακας αρχείων

Τέλος, πρέπει να ορίσουμε ποιος θα λαμβάνει τα σχόλια των commit.

Όνομα	Τύπος	Null	Κλειδί
commentId	Int(11)	OXI	Κύριο
receiverId	Int(11)	OXI	Κύριο - Ξένο

Πίνακας 10 Ενδιάμεσος Πίνακας σχολίου-παραλήπτη

Αφού αναπτύχθηκε τόσο το κομμάτι του κώδικα που υλοποιεί την βάση, όσο και η βάση που σχεδιάζεται, αξίζει να δούμε και σχηματικά το τελικό αποτέλεσμα, προκειμένου οι σχέσεις να είναι πιο κατανοητές. Αξίζει να σημειωθεί πως το σχέδιο δημιουργήθηκε πριν την υλοποίηση, με σκοπό την καλύτερη δομή της βάσης και την αποφυγή λαθών που θα δημιουργούσαν μία βάση αργή ή θα δημιουργούσαν δυσκολίες στην επέκταση της

3.6 Σύνοψη Κεφαλαίου

Στο συγκεκριμένο κεφάλαιο αναλύθηκαν οι αρχές της αλληλεπίδρασης ανθρώπου-υπολογιστή προτού αρχίσει η υλοποίηση της διεπαφής. Στην συνέχεια αναλύθηκε το UI/UX της εφαρμογής, με ανάλυση κάθε σελίδας ξεχωριστά, για την καλύτερη κατανόηση του σκοπού κάθε σελίδας. Τέλος, γίνεται εκτενής αναφορά στην βάση δεδομένων και πως αυτή προέκυψε μέσω του κώδικα του εξυπηρετητή.

4. Λειτουργίες και υλοποίηση

4.1 Περιγραφή απαιτήσεων συστήματος

Στα προηγούμενα κεφάλαια μιλήσαμε επιφανειακά για τον τρόπο λειτουργίας του συστήματος. Ωστόσο, κρίνεται απαραίτητο να γίνει μία εκτενέστερη ανάλυση σχετικά με την αρχιτεκτονική του και τις λειτουργίες του. Αρχικά, το σύστημα διαχείρισης διπλωματικών εργασιών που πραγματευόμαστε, έχει ως πυλώνες τους χρήστες και τους ρόλους αυτών. Σαν σύστημα οι βασικές του λειτουργίες είναι η καταχώρηση, διαχείριση και παρακολούθηση των διπλωματικών εργασιών. Για να επιτευχθεί αυτό όμως έπρεπε να γίνει διάκριση των ρόλων και των δικαιωμάτων που έχει ο κάθε ρόλος, δημιουργώντας microservices, ανάλογα με τον ρόλο του κάθε χρήστη. Παραδείγματος χάρη:

1. **Διαχειριστής (Admin):**
   - Έχει πρόσβαση σε όλες τις λειτουργίες της εφαρμογής
   - Έχει το δικαίωμα να διαγράφει και να προσθέτει χρήστες
   - Μπορεί να αλλάξει τους ρόλους των χρηστών
   - Μελλοντικά θα έχει και πρόσβαση σε ρυθμίσεις του συστήματος, όπως να αυξομειώνει τον μέγιστο αριθμό φοιτητών ανά διπλωματική εργασία
2. **Φοιτητής:**
   - Μπορεί να καταχωρίσει την διπλωματική του εργασία
   - Πρόσβαση σε συνομιλία ζωντανού χρόνου
   - Βλέπει μόνο την δική του διπλωματική εργασία
3. **Καθηγητής:**
   - Βλέπει όλες τις διπλωματικές εργασίες που καθοδηγεί
   - Αρχεία όλων των φοιτητών που τον έχουν ως διδάσκοντα
   - Σχόλια τα οποία δεν είναι φανερά στους φοιτητές

Η δυσκολία του εγχειρήματος ανέβηκε όμως με την χρήση της συνομιλίας σε πραγματικό χρόνο, καθώς και με την πολυπλοκότητα των microservices, καθώς είναι σημαντικό να μην υπάρχει πρόσβαση σε πόρους που δεν πρέπει.

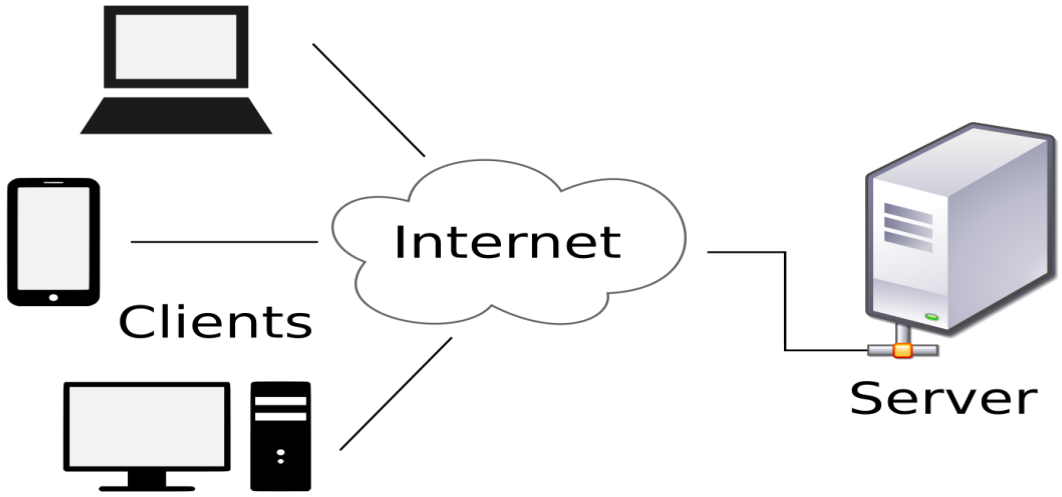
Επομένως, η αρχή της υλοποιήσης του backend ξεκινάει με την αρχικοποίηση στην βάση των ρόλων, καθώς το authentication της εφαρμογής είναι role-based.

```
const db = require("../api/models/index");
const Role = db.role;
module.exports = function initial() {
  Role.create({
    id: 1,
    name: "student",
  });

  Role.create({
    id: 2,
    name: "admin",
  });

  Role.create({
    id: 3,
    name: "teacher",
  });
};
```

Εικόνα 32: Ρόλοι συστήματος

4.2 Αρχιτεκτονική

Κατά κύριο λόγο η αρχιτεκτονική που χρησιμοποιήθηκε στο συγκεκριμένο σύστημα ήταν η client-server. Περιληπτικά η αρχιτεκτονική αυτή χωρίζεται στον πελάτη και τον διακομιστή, μέσω της επικοινωνίας αυτών των δύο προχωράνε τα δεδομένα. Πιο συγκεκριμένα ο πελάτης κάνει αιτήματα στον διακομιστή και ο διακομιστής μετά από επεξεργασία των δεδομένων τα στέλνει πίσω στον πελάτη.



Εικόνα 33: Αρχιτεκτονική Πελάτη-Εξυπηρετητή

Η επικοινωνία αυτών των δύο είναι ασύγχρονη. Αυτό σημαίνει πως ο πελάτης δεν περιμένει την απάντηση του διακομιστή, αλλά συνεχίζει κανονικά με τις λειτουργίες του μέχρι να του δοθεί η απάντηση που περιμένει. Στην συγκεκριμένη περίπτωση ο διακομιστής είναι γραμμένος σε NodeJS, ενώ ο πελάτης σε React. Η υλοποίηση έγινε με χρήση του μοντέλου REST API (Representational State Transfer), όπου μέσω του HTTP πρωτοκόλλου γίνεται εφικτή η επικοινωνία αυτών των δύο.

4.3 Τεχνολογία και υλοποίηση Backend

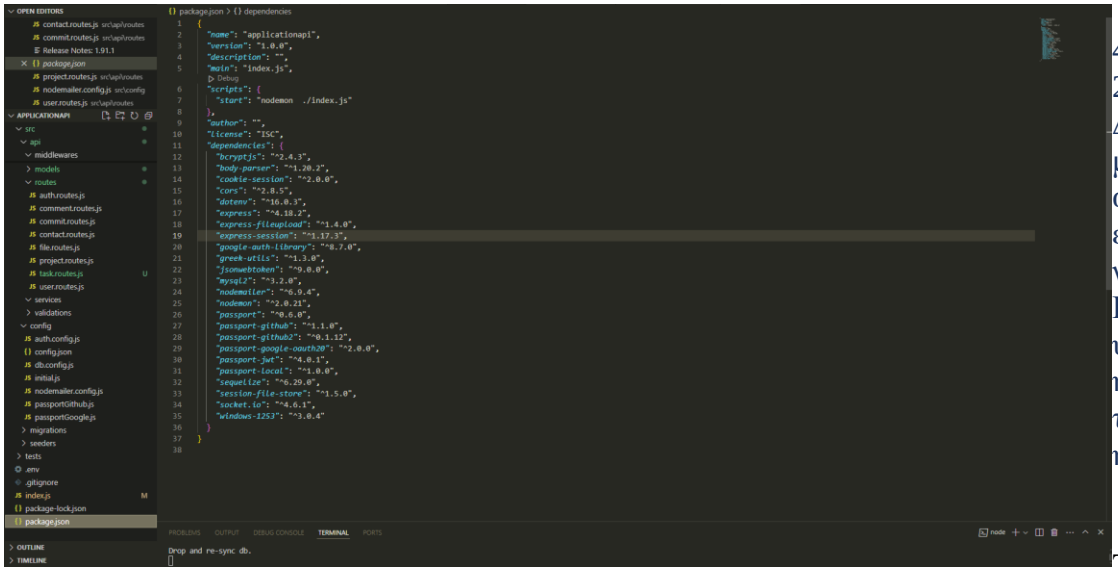
Για την υλοποίηση του εξυπηρετητή, όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο, χρησιμοποιήθηκε η Nodejs. Ωστοσο, επειδή πρόκειται για μία βιβλιοθήκη με πολλές ελευθερίες, η δομή του προγραμμάτος, καθώς και τα πακέτα που χρησιμοποιήθηκαν, ήταν στην δική μας ευχέρεια, επομένως κρίνεται απαραίτητο να αναλυθούν περεταίρω.

4.3.1 Βιβλιοθήκες Εξυπηρετητή

Μερικές από τις πιο βασικές βιβλιοθήκες, καθώς και η λειτουργία τους είναι οι εξής:

- Bcryptjs (https://www.npmjs.com/package/bcryptjs (Kalita, 2021)): χρησιμοποιήθηκε για την κρυπτογράφηση και αποκρυπτογράφηση των κωδικών των χρηστών.
- Dotenv(https://www.npmjs.com/package/dotenv): χρησιμοποιήθηκε για να υπάρχει πρόσβαση στις μεταβλητές περιβάλλοντος. Με αυτόν τον τρόπο γίνεται αποφυγή διάσπασης σημαντικών πληροφοριών.

- Cors(https://www.npmjs.com/package/cors): χρησιμοποιήθηκε καθώς η εφαρμογή του πελάτη και η εφαρμογή του εξυπηρετητή δεν τρέχουν ούτε στο ίδιο domain, ούτε στην ίδια πόρτα.
- Express-fileupload(https://www.npmjs.com/package/express-fileupload): χρησιμοποιήθηκε για την λειτουργία ανάρτησης αρχείων από τους φοιτητές στην εφαρμογή.
- Jsonwebtoken(https://www.npmjs.com/package/jsonwebtoken): χρησιμοποιήθηκε για την αυθεντικοποίηση των χρηστών της εφαρμογής.
- Nodemailer(https://www.npmjs.com/package/nodemailer): χρησιμοποιήθηκε για την αποστολή ηλεκτρονικών μηνυμάτων μέσω της εφαρμογής.
- Passport(https://www.npmjs.com/package/passport): χρησιμοποιήθηκε για να γίνει αυθεντικοποίηση μέσω google και github.
- Sequelize(https://www.npmjs.com/package/sequelize): χρησιμοποιήθηκε προκειμένου να δωθεί η δυνατότητα στο σύστημα να έχει ORM το συστήμα.
- Socket.io(https://www.npmjs.com/package/socket.io): η βιβλιοθήκη αυτή παρέχει την δυνατότητα για ζωντανή επικοινωνία μεταξύ του πελάτη και εξυπηρετήτη.



Εικόνα 34: Package.json εξυπηρετητή

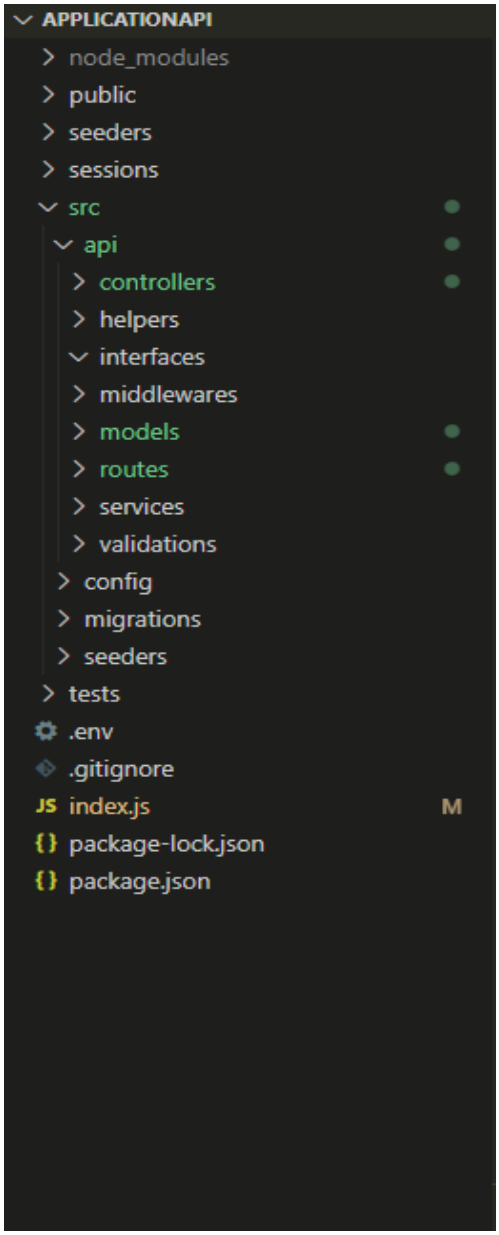
4.3.2 Δομή αρχείων Εξυπηρετητή

Το μειονέκτημα της NodeJS, σε σχέση για παράδειγμα με το Laravel, είναι πως τα αρχεία δεν έχουν μία συγκεκριμένη δομή. Δυστυχώς η ελευθερία της δομής, μπορεί να οδηγήσει σε κάκο design της εφαρμογής. Για αυτό τον λόγο, ακολουθήθηκε ένα μοντέλο, με βάση τα μοντέλα και τους controllers. Αρχικά υπάρχει το αρχείο index.js, το οποίο εκκινεί τον εξυπηρετητή, μαζί με το .env, που περιέχει τις μεταβλητές περιβάλλοντος. Στην συνέχεια υπάρχει ο φάκελος src που περιέχει τους υποφακέλους api και config. Ο φάκελος config, περιέχει όλα τα αρχεία για την εκκίνηση λειτουργιών, όπως το nodemail, την βάση δεδομένων και την αυθεντικοποίηση μέσω Google. Ο σημαντικότερος φάκελος του αρχείου όμως, είναι το api, καθώς περίεχει μέσα:

- Controllers: τα αρχεία αυτά είναι υπεύθυνα για την υλοποιήση της λειτουργίκοτητας, δηλαδή να βρουν κάποια εγγραφή μέσα στην βάση ή να ενημερώσουν κάποια από της ήδη υπάρχουσες.
- Middlewares: τα αρχεία αυτά κάνουν ελέγχους στα routes, προκειμένου να μην δοθεί πρόσβαση σε κάποιον που δεν πρέπει στην εφαρμογή.

- Models: δημιουργούν τα μοντέλα για την βάση δεδομένων
- Routes: τα αρχεία αυτά ορίζουν ποια θα είναι τα endpoints, τι υλοποίηση θα κάνει το καθένα από αυτά, καθώς και ποια middleware θα το προστατεύουν.

Οπτικά η δομή των αρχείων στο backend είναι η εξής



Εικόνα 35: Δομή αρχείων εξυπηρετητή

4.3.3 Middlewares

Η χρήση της τεχνικής των middlewares παρέχει πολλά πλεονεκτήματα στην εφαρμογή. Το βασικότερο είναι πως στην ουσία αποτελούν συναρτήσεις που μπορούν να επαναχρησιμοποιηθούν σε όλα τα σημεία της εφαρμογής. Το βασίκοτερο middleware είναι ο έλεγχος του token. Δεν θα μπορούσε να υπάρχει εφαρμογή αν ο καθένας μπορεί να έχει πρόσβαση στους πόρους της. Αντιθέτως, χρειάζεται token και να είναι έγκυρο προκειμένου να έχει πρόσβαση σε βασικές λειτουργίες της εφαρμογής. Όμως, μόνο το token δεν αρκεί, αφού δεν θέλουμε να έχει τις ίδιες δυνατότητες ένα καθηγητής με έναν φοιτητή. Για αυτό τον λόγο δημιουργήθηκαν middlewares για τον έλεγχο του ρόλου. Με αυτά τα δύο βασικά middleware, καλύπτουμε το μεγαλύτερο ποσοστό των routes που θέλουμε να μην είναι ανοικτά προς όλους. Τέλος, υπάρχουν και middlewares, τα οποία είναι υπεύθυνα για τον έλεγχο της ύπαρξης διπλωματικής, ώστε να αποτρέπει την δημιουργίας δεύτερης.

Ο κώδικας για τα middlewares της εφαρμογής είναι ο εξής:

- Το παρακάτω middleware είναι υπεύθυνο για τον έλεγχο του token που στέλνει ο client στον server. Αρχικά λαμβάνει το token από τις κεφαλίδες του αιτήματος και ελέγχει αν είναι έγκυρο, αλλιώς επιστρέφει κωδικό μη εξουσιοδοτημένης πρόσβασης.

```
verifyToken = (req, res, next) => {
  let token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send({
      message: "No token provided!",
    });
  }

  jwt.verify(token, config.secret, (err, decoded) => {
    if (err) {
      return res.status(401).send({
        message: "Unauthorized!",
      });
    }
    req.userId = decoded.id;
    next();
  });
};
```

Εικόνα 36: Middleware αυθεντικοποίησης

- Τα επόμενο middleware είναι υπεύθυνα για τον έλεγχο του ρόλου που έχει ο client. Σκοπός του είναι να μην υπάρξει πρόσβαση σε πόρους που δεν θα πρέπει να έχει. Γίνεται έλεγχος για όλους τους ρόλους ξεχωριστά.

```
isAdmin = (req, res, next) => {
  User.findByPk(req.userId).then((user) => {
    user.getRoles().then((roles) => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "admin") {
          next();
          return;
        }
      }

      re (property) message: string
        message: "Require Admin Role!",
      });
      return;
    });
  });
};
```

Εικόνα 37: Middleware ρόλου διαχειριστή

```
isTeacher = (req, res, next) => {
  User.findByPk(req.userId).then((user) => {
    user.getRoles().then((roles) => {
      for (let i = 0; i < roles.length; i++) {
        if (roles[i].name === "teacher") {
          next();
          return;
        }
      }

      res.status(403).send({
        message: "Require Teacher Role!",
      });
    });
  });
};
```

Εικόνα 38: Middleware ρόλου καθηγητή

- Επιπλέον υπάρχουν τα middlewares που είναι υπεύθυνα για την εγγραφή του χρήστη. Πιο συγκεκριμένα, δημιουργήθηκαν middlewares για τον έλεγχο διπλότυπου λογαριασμού, και υπάρξεις ρόλου (για αποφυγή λάθους)

```
checkDuplicateUsernameOrEmail = (req, res, next) => {
  // Email
  User.findOne({
    where: {
      email: req.body.email,
    },
  }).then((user) => {
    if (user) {
      res.status(400).send({
        message: "Failed! Email is already in use!",
      });
      return;
    }

    next();
  });
};
```

Εικόνα 39: Έλεγχος διπλότυπου email

```
checkRolesExisted = (req, res, next) => {
  if (req.body.roles) {
    for (let i = 0; i < req.body.roles.length; i++) {
      if (!ROLES.includes(req.body.roles[i])) {
        res.status(400).send({
          message: "Failed! Role does not exist = " + req.body.roles[i],
        });
        return;
      }
    }
  }

  next();
};
```

Εικόνα 40: Middleware ύπαρξης ρόλου

4.3.4 Routes

Τα routes ή αλλίως endpoints είναι τα σημεία του εξυπηρετητή που καλεί ο πελατής για να λάβει τους ανάλογους πόρους. Επειδή το πλήθος των endpoint ήταν αρκετά μεγάλο, χωρίστηκαν σε διάφορα αρχεία ανάλογα με τον σκοπό τους.

Endpoint Αυθεντικοποίησης

- **/api/auth/signup**, η λειτουργία του είναι να γίνει εγγραφή ενός μαθητή, σε περίπτωση που το αίτημα το πραγματοποιεί κάποιος διαχειριστής ή κάποιος καθηγητής. Για αυτό τον λόγο τα middleware που προστατεύουν το endpoint αυτό είναι ο έλεγχος αν είναι διπλό το email ή το username, αν υπάρχει ο επιθυμητός

ρόλος, αν αυτός που κάνει το αίτημα έχει token και αν ο ρόλος του πελάτη είναι καθηγητή ή διαχειριστή

```
app.post(
  "/api/auth/signup",
  [
    verifySignUp.checkDuplicateUsernameOrEmail,
    verifySignUp.checkRolesExisted,
    authJwt.verifyToken,
    authJwt.isTeacherOrAdmin,
  ],
  controller.signup
);
```

Εικόνα 41: Route εγγραφή, μέσω διαχειριστή

- **/api/auth/signup/user**, η λειτουργία του είναι να κάνει εγγραφή ένας απλός χρήστης και θέτει ως default ρόλο αυτόν του μαθητή, μόνο έλεγχος που γίνεται αν το email ή το username χρησιμοποιείται ήδή.

```
app.post(
  "/api/auth/signup/user",
  [verifySignUp.checkDuplicateUsernameOrEmail],
  controller.openSignup
);
```

Εικόνα 42: Route εγγραφής

- **/api/auth/signin** , η λειτουργία που παρέχει είναι της σύνδεσης στο σύστημα, δηλαδή αν το ψευδώνυμο και ο κωδικός είναι σωστός (υπάρχει στην βάση), τότε δίνει token στον πελάτη

```
app.post("/api/auth/signin", controller.signin);
```

Εικόνα 43: Route σύνδεσης

Endpoint Χρήστη

- **/api/users/all (GET)**, επιστρέφει από την βάση όλους τους χρήστες του συστήματος. Το συγκεκριμένο endpoint δεν έχει κάποιον έλεγχο, πέρα από το αίτημα να γίνεται από κάποιον πελάτη με token.

```
app.get("/api/users/all", [authJwt.verifyToken], controller.getAllUsers);
```

Εικόνα 44: Route επιστροφής χρηστών

- /

api

/users/pass (POST) , έλεγχει αν υπάρχει έγκυρο token και παρέχει την δυνατότητα για αλλαγή κωδικού.

```
app.post("/api/users/pass", [authJwt.verifyToken], controller.updateUserPass);
```

Εικόνα 45: Route αλλαγής κωδικού

- **api/users/email (POST)**, έλεγχει αν υπάρχει έγκυρο token και παρέχει την δυνατότητα για αλλαγή email.

```
app.post(
  "/api/users/email",
  [authJwt.verifyToken, verifySignUp.checkDuplicateUsernameOrEmail],
  controller.updateUserEmail
);
```

Εικόνα 46: Route αλλαγής email

- **/api/users/:id (DELETE),** το route αυτό είναι υπεύθυνο για την διαγραφή κάποιου χρήστη, περνώντας το id του στο url. Για να έχει κάποιος πρόσβαση σε αυτό το route πρέπει να έχει ρόλο διαχειριστή.

```
app.delete(
  "/api/users/:id",
  [authJwt.verifyToken, authJwt.isAdmin],
  controller.deleteUser
);
```

Εικόνα 47: Route διαγραφής χρήστη

- **/api/users/:id (POST),** το route αυτό είναι υπεύθυνο για την ενημέρωση κάποιου χρήστη, περνώντας το id του στο url. Για να έχει κάποιος πρόσβαση σε αυτό το route πρέπει να έχει ρόλο διαχειριστή.

```
app.post(
  "/api/users/:id",
  [authJwt.verifyToken, authJwt.isTeacherOrAdmin],
  controller.updateUser
);
```

Εικόνα 48: Route ενημέρωση χρήστη

- **/api/users/personal/:id (POST),** το route αυτό παρέχει στον χρήστη την δυνατότητα να αλλάξει προσωπικές του πληροφορίες που έχει περάσει στην εφαρμογή.

```
app.post(
  "/api/users/personal/:id",
  [authJwt.verifyToken],
  controller.updateUserPersonalInfo
);
```

Εικόνα 49: Route ενημέρωσης προσωπικών στοιχείων χρήστη

Endpoint Email

- **/api/contact (POST),** δέχεται ως request body, τον τίτλο, το θέμα, το μήνυμα και τους παραλήπτες και στέλνει το email στους αποδέκτες.

```
app.post("/api/contact", [authJwt.verifyToken], controller.createEmail);
```

Εικόνα 50: Route δημιουργίας email

Endpoint Εργασίας

- **/api/project/create (POST),** δέχεται ως request body, τον τίτλο της διπλωματικής, την περιγραφή,το εξάμηνο, τον χρόνο, το id του καθηγητή και τα id των φοιτητών που συμμετέχουν στην διπλωματική. Για να υπάρχει πρόσβαση σε αυτό το route, πρέπει να υπάρχει έγκυρο token, ο ρόλος να είναι καθηγητή ή φοιτητή, και σε περίπτωση που είναι φοιτητής να μην έχει άλλη εργασία.

```
app.post(
  "/api/project/create",
  [authJwt.verifyToken, authJwt.isTeacherOrStudent, project.hasOneProject],
  controller.createProject
);
```

Εικόνα 51: Route δημιουργίας διπλωματικής

- **/api/project/update/:id (POST),** με τα ίδια middlewares, παίρνει ως request body την νέα κατάσταση της διπλωματικής εργασίας και την ενημερώνει.

```
app.post(
  "/api/project/update/:id",
  [authJwt.verifyToken, authJwt.isTeacherOrStudent, project.hasOneProject],
  controller.updateProject
);
```

Εικόνα 52: Route ενημέρωσης διπλωματικής

- **/api/project/:id (GET),** επιστρέφει την εργασία με το id της παραμέτρου.

```
app.get("/api/project/:id", [authJwt.verifyToken], controller.getProject);
```

Εικόνα 53: Route επιστροφής διπλωματικής

- **/api/project (GET)**, επιστρέφει όλα τις εργασίες του καθηγητή που κάνει το αίτημα.

```
app.get(
  "/api/project",
  [authJwt.verifyToken, authJwt.isTeacher],
  controller.getAllProjects
);
```

Εικόνα 54: Route διδάσκοντα διπλωματικής

Endpoint Commit

- **/api/commit/create/:id (POST),** είναι υπεύθυνο για την δημιουργία μίας νέας καταχώρησης. Τα πεδία που χρειάζεται είναι ο τίτλος, η περιγραφή και τα αρχεία που έχει επισυνάψει ο δημιουργός του

```
app.post("/api/commit/create/:id", controller.createCommit);
```

Εικόνα 55: Route δημιουργίας commit

- **/api/commit/:id (DELETE),** διαγράφει μία καταχώριση.

```
app.delete("/api/commit/:id", controller.deleteCommit);
```

Εικόνα 56: Route διαγραφής commit

- **/api/commit/:userId/:id (GET),** επιστρέφει μία καταχώρηση με βάση το id του χρήστη και το id της καταχώρησης.

```
app.get(
  "/api/commit/:userId/:id",

  [authJwt.isTeacherOrStudent, authJwt.verifyToken, project.projectOwner],

  controller.getCommit
);
```

Εικόνα 57: Route επιστροφής commit, με βάση τον χρήστη

- **/api/commit/ (GET),** επιστρέφει όλες τις καταχωρήσεις.

```
app.get(
  "/api/commit/",
  [authJwt.verifyToken, authJwt.isTeacher],
  controller.getAllCommits
);
```

Εικόνα 58: Route επιστροφής όλων των commit

Endpoint αρχείων

- **/api/file/:id,** επιστρέφει ένα αρχείο με βάση το id που περνάει στο αίτημα ο πελάτης.

```
app.get("/api/file/:id", [authJwt.verifyToken], controller.getFile);
```

Εικόνα 59: Route επιστροφής αρχείων

Endpoint σχολίων

- **/api/comment/create/:id,** το endpoint αυτό δημιουργεί ένα σχόλιο στην βάση δεδομένων.

```
app.post("/api/comment/create/:id", controller.createComment);
```

Εικόνα 60: Route δημιουργίας σχολίου

- **/api/comment/:userId/:id,** το endpoint αυτό παίρνει δύο παραμέτρους, το id του χρήστη και το id του σχολίου, και επιστρέφει τα σχόλια.

```
app.get(
  "/api/comment/:userId/:id",
  [authJwt.isTeacherOrStudent, authJwt.verifyToken, project.projectOwner],
  controller.getComment
);
```

Εικόνα 61: Route επιστροφής σχολίων, βάση χρήστη

4.3.5 Αυθεντικοποίηση στον εξυπηρετητή

Όπως αναλύθηκε και σε προήγουμενο κεφάλαιο, στην εφαρμογή υπάρχει η δυνατότητα δημιουργίας λογαριασμού, μέσω της εφαρμογής καθέ αυτής, καθώς και από εξωτερικές εφαρμογές (google, github). Αρχικά αξίζει να δούμε την διαδικασία δημιουργίας ενός καινούργιου χρήστη

```
exports.openSignup = (req, res) => {
  var authorities = [];
  User.create({
    username: req.body.username,
    email: req.body.email,
    password: bcrypt.hashSync(req.body.password, 8),
  })
    .then((user) => {
      var token = jwt.sign({ id: user.id }, config.secret, {
        expiresIn: 86400, // 24 hours
      });
      user.setRoles([1]).then(() => {
        User.findOne({
          attributes: { exclude: ["password"] },
          where: { id: user.id },
          include: [db.role],
        }).then((newUser) => {
          newUser.getRoles().then((roles) => {
            for (let i = 0; i < roles.length; i++) {
              authorities.push(roles[i].name.toLowerCase());
            }
            res.status(200).send({
              id: newUser.id,
              username: newUser.username,
              email: newUser.email,
              roles: authorities,
              accessToken: token,
            });
          });
        });
      });
    })
    .catch((err) => {
      res.status(500).send({ message: err.message });
    });
```

Εικόνα 62: Εγγραφή

Η συνάρτηση που τρέχει όταν ένας πελάτης κάνει αίτημα για να δημιουργήσει νέο χρήστη είναι η openSignup. Σαν παραμέτρους δέχεται το request.body και επιστρέφει μία απάντηση στον πελάτη. Αφού κληθεί η συνάρτηση User.create περνώντας τα στοιχεία του χρήστη (email, username και κωδικός encrypted), περνάμε και τον αντίστοιχο ρόλο, με default ρόλο αυτό του μαθητή. Σε περίπτωση που η διαδικασία αυτή πετύχει, τότε επιστρέφεται απάντηση με κωδικό 200, με το token που αναλογεί στον χρήστη. Σε αντίθετη περίπτωση, υπάρχει exception με κωδικό 500, και μήνυμα το λάθος. Επομένως, ο χρήστης καταχωρείται στην βάση ως εξής

10	student	student@student.com	$2a$08$zbE2IyXZ5pbtiT5Gc0jIP.W881UCmGvy83LB6694yeb...	NULL	2023-05-24 08:07:43	2023-05-24 08:07:43

Εικόνα 63: Εγγραφή στην Βάση

Η διαδικασία αυτή διαφοροποιείται σε περίπτωση που ο χρήστης εγγραφεί ή συνδεθεί μέσω Google ή Github. Πιο συγκεκριμένα, καλείται η συνάρτηση googleSignup, με δεδομένα που έρχονται από το Google Authentication, και ακολουθεί την ίδια λογική με την απλή εγγραφή.

```
exports.googleSignup = (req, res) => {
  var token = jwt.sign({ id: req.user.id }, config.secret, {
    expiresIn: 86400, // 24 hours
  });
  var authorities = [];
  User.findOne({
    attributes: { exclude: ["password"] },
    where: { id: req.user.id },
    include: [db.role],
  })
    .then((user) => {
      user.getRoles().then((roles) => {
        if (roles.length == 0) {
          user.setRoles([1]).then((r) => {
            res.status(200).send({
              id: user.id,username: user.username,email: user.email,roles: r,token: token,color: user.color,
            });
          });
        } else {
          for (let i = 0; i < roles.length; i++) {
            authorities.push(roles[i].name.toLowerCase());
          }
          res.status(200).send({
            id: user.id,username: user.username,email: user.email,roles: authorities,token: token,color: user.co
          });
        }
      });
    })
    .catch((error) => res.status(401));
};
```

Εικόνα 64: Εγγραφή μέσω Google

Η διαδικασία αυτή ακολουθείται είτε ο χρήστης κάνει εγγραφή είτε σύνδεση στην εφαρμογή. Επιπλέον, σε περίπτωση που υπάρχει ήδη λογαριασμός με αυτό το email, περνάει στην βάση

το googleId και προχωράει με σύνδεση. Σε περίπτωση απλής σύνδεσης η συνάρτηση είναι η εξής

```
exports.signin = (req, res) => {
  User.findOne({where: {email: req.body.email,},}).then((user) => {
    if (!user) return res.status(404).send({ message: "User Not found." });
    var passwordIsValid = bcrypt.compareSync(
      req.body.password,
      user.password
    );
    if (!passwordIsValid) {
      return res.status(401).send({
        accessToken: null,
        message: "Invalid Password!",
      });
    }
    var token = jwt.sign({ id: user.id }, config.secret, {
      expiresIn: 86400, // 24 hours
    });
    var authorities = [];
    user.getRoles().then((roles) => {
      for (let i = 0; i < roles.length; i++) {
        authorities.push(roles[i].name.toLowerCase());
      }
      res.status(200).send({
        id: user.id,username: user.username,email: user.email,roles: authorities,accessToken: token,color: us
      });
    });
  })
  .catch((err) => {
    res.status(500).send({ message: err.message });
  });
};
```

Εικόνα 65: Σύνδεση Χρήστη

Αρχικά γίνεται αναζήτηση στην βάση για χρήστη με το ίδιο email, σε περίπτωση που δεν βρέθει επιστρέφει απάντηση με 404 και ανάλογο μήνυμα. Στην συνέχεια, αποκρυπτογραφεί τον κωδικό, προκειμένου να δει ο εξυπηρετητής, πως τα credentials είναι σωστά,αν είναι λάθος επιστρέφει 401 (unauthorized) με ανάλογο μήνυμα, αλλιώς κάνει attach στον χρήστη ένα token και επιστρέφει τα δεδομένα.

4.3.6 Δημιουργία Διπλωματικής και Επεξεργασία

Όπως αναφέραμε σε προηγούμενο κεφάλαιο, η διπλωματική εργασία χρειάζεται τίτλο, επιβλέπων καθηγητή και τουλάχιστον έναν φοιτητή. Προκειμένου να δημιουργηθεί η εργασία, ο εξυπηρετητής επεξεργάζεται το αίτημα του πελάτη ως εξής:

```
exports.createProject = (req, res) => {
  Project.create({title: req.body.title,description: req.body.description,semester: req.body.semester,
    year: req.body.year,
  }).then((project) => {
    User.findOne({where: {id: req.body.teacher,},})
      .then((user) => {
        project.setTeacher(user).then(() => {
          User.findAll({where: {email: {[Op.or]: req.body.students,},},}).then((users) => {
            project.setStudents(users).then(() => {
              Project.findOne({where: { id: project.id },attributes: { exclude: ["teacherId"] },include: [
                  {
                    model: User,
                    as: "teacher",
                    attributes: { exclude: ["password"] },
                    include: [db.role],
                  },
                  {
                    model: User,
                    as: "students",
                    include: [db.role],
                    attributes: { exclude: ["password", "student_project"] },
                  },
                ],
              }).then((newProject) => {
                res.send({
                  message: "Project was registered successfully! ",
                  project: newProject,
                });
              });
            });
          });
        });
      });
```

Εικόνα 66: Δημιουργία Διπλωματικής

Η συνάρτηση createProject είναι υπεύθυνη για τη δημιουργία ενός νέου project και τη συσχέτισή του με έναν καθηγητή και μια λίστα φοιτητών. Η λειτουργία ξεκινά δημιουργώντας ένα νέο project μέσω της μεθόδου Project.create, χρησιμοποιώντας δεδομένα από το σώμα του αιτήματος (req.body) για να ορίσει τον τίτλο, την περιγραφή, το εξάμηνο και το έτος του νέου project. Αφού δημιουργηθεί το project, η συνάρτηση αναζητά έναν καθηγητή με τη χρήση της μεθόδου User.findOne, χρησιμοποιώντας το ID του καθηγητή που αναμένεται να έχει δοθεί στο req.body.teacher. Αν βρεθεί ο καθηγητής, το project συνδέεται με αυτόν μέσω της μεθόδου project.setTeacher. Στη συνέχεια, η συνάρτηση αναζητά όλους τους φοιτητές των οποίων τα email αντιστοιχούν σε αυτά που περιλαμβάνονται στο req.body.students, χρησιμοποιώντας την μέθοδο User.findAll. Αφού βρεθούν οι φοιτητές, το project συνδέεται μαζί τους μέσω της μεθόδου project.setStudents. Τέλος, η συνάρτηση πραγματοποιεί μια αναζήτηση για το project που δημιουργήθηκε, εξαιρώντας το πεδίο teacherId από τα αποτελέσματα, και συμπεριλαμβάνοντας πληροφορίες για τον καθηγητή και τους φοιτητές (χωρίς τους κωδικούς πρόσβασης) μαζί με τους ρόλους τους. Αφού ολοκληρωθεί αυτή η αναζήτηση, στέλνει μια απάντηση με μήνυμα επιτυχίας και τις λεπτομέρειες του project που καταχωρήθηκε στην βάση δεδομένων.

Η συνάρτηση που είναι υπεύθυνη για την τροποιήση μίας διπλωματικής εργασίας είναι η updateProject.

```
exports.updateProject = (req, res) => {
  Project.update({ status: req.body.status },{ where: { id: req.params.id }, returning: true, plain: true })
    .then(() =>
      Project.findOne({
        where: { id: req.params.id },
        include: [
          {
            model: User,
            as: "teacher",
            where: { id: req.userId },
            attributes: { exclude: ["password"] },
          },
          {
            model: User,
            as: "students",
            attributes: { exclude: ["password", "roles", "student_project"] },
          },
          {
            model: Commit,
            include: [
              {
                model: User,
                as: "user",
                attributes: { exclude: ["password"] },
              },{ model: Project, as: "project" },{ model: F, as: "files" },{
                model: Comment,
                as: "comments",
                include: [
                  {
                    model: User,
```

Εικόνα 67: Ενημέρωση Διπλωματικής

```
                    as: "sender",
                    attributes: { exclude: ["password", "googleId", "githubId", "createdAt", "updatedAt"] },
                  },
                ],
              },
            ],
          },
        ],}).then((proj) => res.status(200).send({ project: proj }))).catch((error) => {res.status(400).send({ m
};
```

Εικόνα 68: Ενημέρωση Διπλωματικής 2

Η λειτουργία της ξεκινάει με την χρήση της συνάρτησης update, με κάποιες παραμέτρους. Αυτοί οι παράμετοι είναι το status, το id για να την ταυτοποιήση, καθώς και το returning ώστε να μας γυρίσει το αντικείμενο που θέλουμε. Στην συνέχεια όμως, επειδή το returning δεν γυρίζει όλες τις σχέσεις του αντικειμένου "Διπλωματική", γράφουμε όλες τις σχέσεις που θέλουμε να επιστραφούν, καθώς και ποια πεδία δεν θέλουμε (πχ τους κωδικούς των χρηστών που συμμετέχουν στην διπλωματική. Με ανάλογο τρόπο απλώς χωρίς ενημέρωση των στοιχείων λειτουργούν και οι συναρτήσης getProject (που γυρίζει μία Διπλωματική με βάση το id) και η getAllProjects (που γυρίζει όλες τις διπλωματικές για έναν Καθηγητή).

4.3.7 Δημιουργία Commit

Η συνάρτηση αυτή αρχικά θέτει σε μεταβλητές τόσο τα αρχεία που αφορούν το commit, όσο και το project. Στην συνέχεια, δημιουργείται το commit στην βάση, χωρίς κάποια σχέση εξάρτησης προς το παρόν. Μετά δημιουργούνται οι σχέσεις τόσο των χρηστών όσο και του project. Τέλος, αποθηκεύονται τα αρχεία που αφορούν το commit.

```
exports.createCommit = async (req, res) => {
  try {
    console.log(req.files.files);
    const images = req.files.files;
    const project = await Project.findOne({ where: { id: req.params.id } });

    const commit = await Commit.create({
      title: req.body.title,
      description: req.body.description,
    });

    await commit.setProject(project);

    const user = await User.findOne({ where: { id: req.body.user } });
    await commit.setUser(user);

    const savePath = path.join("public/files/", `${project.id}/`, `${commit.id}`);
    fs.mkdirSync(savePath, { recursive: true });

    let arr = [];
    const saveFile = (file) => {
      const filePath = `${savePath}/${file.name}`;
      file.mv(filePath, (err) => {
        if (err) throw err;
      });
      return {
        fileName: file.name,
        filePath: savePath,
        type: file.mimetype.split("/")[1],
```

Εικόνα 69: Δημιουργία commit 1

```
    };
  };

  if (Array.isArray(images)) {
    arr = images.map((file) => saveFile(file));
  } else {
    arr.push(saveFile(images));
  }

  const files = await F.bulkCreate(arr);
  await commit.setFiles(files);

  const commitWithDetails = await Commit.findOne({
    where: { id: commit.id },
    attributes: { exclude: ["userId", "projectId"] },
    include: [
      { model: User, as: "user", attributes: { exclude: ["password"] } },
      { model: Project, as: "project" },
      { model: F, as: "files" },
    ],
  });

  res.status(200).send({ commit: commitWithDetails });
} catch (err) {
  console.log(err);
  res.status(500).send({ message: err.message });
}
};
```

Εικόνα 70: Δημιουργία commit 2

4.3.8 Δημιουργία Email

Στην εφαρμογή παρέχεται η δυνατότητα σε κάποιον χρήστη που έχει ρόλο Διδάσκοντα, να μπορεί να στέλνει emails, με βάση μία λίστα από email. Για να επιτευχθεί αυτό δημιουργείται ένα αντικείμενο transporter με χρήση της βιβλιοθήκης nodemailer, για να μπορεί να δημιουργηθεί σύνδεση με το mailtrap. Προφανώς, αντί για το mailtrap, μπορεί να χρησιμοποιηθεί οποιοσδήποτε host (gmail,yahoo κτλπ.).

```
module.exports = {
  transporter: nodemailer.createTransport({
    host: "sandbox.smtp.mailtrap.io",
    port: 2525,
    auth: {
      user: "0b61e0f0a8ea8e",
      pass: "20dbaccedba23c",

    },
  }),

};
```

Εικόνα 71: Transporter αντικείμενο

Το αντικείμενο αυτό στην συνέχεια καλείται όποτε ο εξυπηρετητής δεχτεί κάποιο αίτημα για δημιουργία ηλεκτρονικού μηνύματος.

```
exports.createEmail = async (req, res) => {
console.log(req.body)
  const {title,subject,message,receivers} = req.body;
  try{
    await transporter.sendMail({
        from: req.body.email,
        to: receivers,
        subject: subject,
        text: message,
        html: message
    });
    return res.status(200).send({message:"Message sent!"})
  }catch(error){
    console.log(error);
    return res.status(400).json({message: error.message})
  }
};
```

Εικόνα 72: Δημιουργία Email

Το αντικείμενο αυτό έχει μία built-in συνάρτηση για να στέλνει ηλεκτρονικά μηνύματα, ανάλογα με τις τιμές που το αρχικοποιήσαμε.

4.3.9 Δημιουργία Σχολίου

Η συνάρτηση που δημιουργεί σχόλια είναι η εξής

```
exports.createComment = async (req, res) => {
  const sender = req.body.sender;
  const commitId = req.body.commit;

  try {
    const comment = await Comment.create({
      message: req.body.message,
      visible: req.body.visible,
    });
    await comment.setSender(sender);
    const commit = await Commit.findOne({ where: { id: commitId } });
    await comment.setCommit(commit);
    const project = await Project.findOne({ where: { id: req.params.id } });
    const teacher = await project.getTeacher();

    if (teacher.id == sender) {
      const students = await project.getStudents();
      await comment.setReceivers(students);
    } else {
      const students = await project.getStudents();
      const receivers = students.filter(student => student.id !== sender);
      receivers.push(teacher);
      await comment.setReceivers(receivers);
    }
```

Εικόνα 73: Δημιουργία Σχολίου 1

```
    const completeComment = await Comment.findOne({
      where: { id: comment.id },
      include: [
        {
          model: User,
          as: "sender",
          attributes: { exclude: ["password"] },
        },
        {
          model: User,
          as: "receivers",
          attributes: { exclude: ["password", "receiver_comment"] },
        },
      ],
    });

    res.status(200).send({ comment: completeComment });
  } catch (err) {
    console.log(err);
    res.status(500).send({ message: "An error occurred." });
  }
};
```

Εικόνα 74: Δημιουργία Σχολίου 2

Η συνάρτηση αυτή δεν αφορά την διαδικασία της ζωντανής επικοινωνίας μεταξύ εξυπηρετητή και πελάτη, αλλά αποτελεί ένα backup, καθώς αποθηκεύει τα μηνύματα της συνομιλίας στην βάση δεδομένων. Μία καλή τεχνική, που δεν χρησιμοποιήθηκε για λόγους απλότητας, θα ήταν η κρυπτογράφηση των μηνυμάτων, προκειμένου να αποφευχθεί πιθανή διαρροή μηνυμάτων. Αρχικά, δημιουργείται με την χρήση του create, ένα μήνυμα στην βάση, με στοιχεία μόνο αν είναι ορατό και το μήνυμα. Στην συνέχεια τοποθετείται ο αποστελέας του μηνύματος, και βρίσκουμε για ποιο commit είναι αυτό το μήνυμα, έχοντας με αυτό τον τρόπο και του μαθητές και τον καθηγητή που αφορά το μήνυμα (με τις συναρτήσεις getTeacher και get Students). Αφού περάσουμε όλη την πληροφορία, την επιστρέφουμε με ανάλογο μήνυμα.

Η συνάρτηση για την επιστροφή σχολίων (μαζικά ανάλογα το project) είναι η εξής

```
exports.getComment = (req, res) => {
  Comment.findAll({
    where: {
      projectId: req.params.id,
    },
    include: [
      {
        model: User,
        as: "user",
        attributes: { exclude: ["password"] },
      },
      {
        model: Project,
        as: "project",
      },
    ],
  })
    .then((comments) => res.status(200).send({ comments: comments }))
    .catch((error) => res.status(500).send({ message: "Comments not found" }));
};
```

Εικόνα 75: Fetch Comment

4.3.10 Ασφάλεια Εξυπηρετητή

Ένα τέτοιο σύστημα, ειδικά από την πλευρά του εξυπηρετητή, πρέπει να είναι πλήρως ασφαλές, αφού σε μελλοντική του έκδοση θα περιέχει ευαίσθητες πληροφορίες των χρηστών. Επιπλέον, δεν είναι αποδεκτό να μπορεί να διαγραφεί κάποια διπλωματική, καθώς θα χαθεί τόσο η πρόοδος του φοιτητή πάνω στο θέμα που πραγματεύεται, όσο και η εμπιστοσύνη στην εφαρμογή. Για αυτό τον λόγο τηρήθηκαν κάποιες βασικές αρχές για την προστασία των δεδομένων.

Αρχικά, δεν μπορεί στην βάση δεδομένων να περαστεί ο κωδικός του χρήστη χωρίς να έχει κρυπτογραφηθεί, αφού μία ενδεχόμενη επιτυχημένη επίθεση στην βάση θα δώσει πρόσβαση σε όλους τους λογαριασμούς. Για να επιτευχθεί η κρυπτογράφηση χρησιμοποιείται η συνάρτηση της βιβλιοθήκης bcrtypt, hashSync, η οποία δέχεται ως όρισμα τον κωδικό και τις φορές κρυπτογράφησης. Σύμφωνα με μελέτες, έχει αποδειχθεί πως οκτώ φορές είναι αρκετές για να θεωρείται ένας κωδικός ασφαλής. Σίγουρα με παραπάνω κύκλους κρυπτογράφησης το σύστημα θα ήταν ακόμα πιο ασφαλές, ωστόσο οι υπολογιστικές πράξεις του εξυπηρετητή θα ανέβαιναν αρκετά, καθιστώντας το σύστημα πιο αργό.

Άλλο ένα μέσο ασφαλείας, προκείμενου η πρόσβαση στην βάση να γίνει πιο δύσκολη σε κάποια πιθανή επίθεση, είναι οι μεταβλητές περιβάλλοντος. Για να επιτευχθεί αυτό χρειάστηκε το πακέτο dotenv, το όποιο μας παρέχει πρόσβαση στις μεταβλητές αυτές. Το προνόμιο με αυτή την στρατηγική είναι πως όταν η εφαρμογή ανέβει σε κάποιον εξυπηρετητή, ορίζονται οι μεταβλητές εξωτερικά από τα αρχεία του κώδικα, καθιστώντας το αδύνατο να έχει κάποιος πρόσβαση σε αυτές. Μερικές μεταβλητές που πρέπει να συμπεριληφθούν σε .env αρχείο είναι το κλειδί αποκρυπτογράφησης, οι κωδικοί της βάσης, το google Id και το github Id. Η μορφή που έχει το αρχείο .env είναι η εξής (έχουν κρυφτεί τα Google client secret id και github client secret id.

```
APP_PORT=4000
DB_HOST= 127.0.0.1
DB_USER= root
DB_PASS=
DB_DIALECT= mysql
DB_DATABASE= thesis

GOOGLE_CLIENT_ID=1076945913606-gjic6739lu51cifse0e47lgmihlf452b.apps.googleusercontent.com
GITHUB_CLIENT_ID=d325c10ee845ab754329
```

Εικόνα 76: .env εξυπηρετητή

4.4 Τεχνολογία και υλοποίηση Frontend

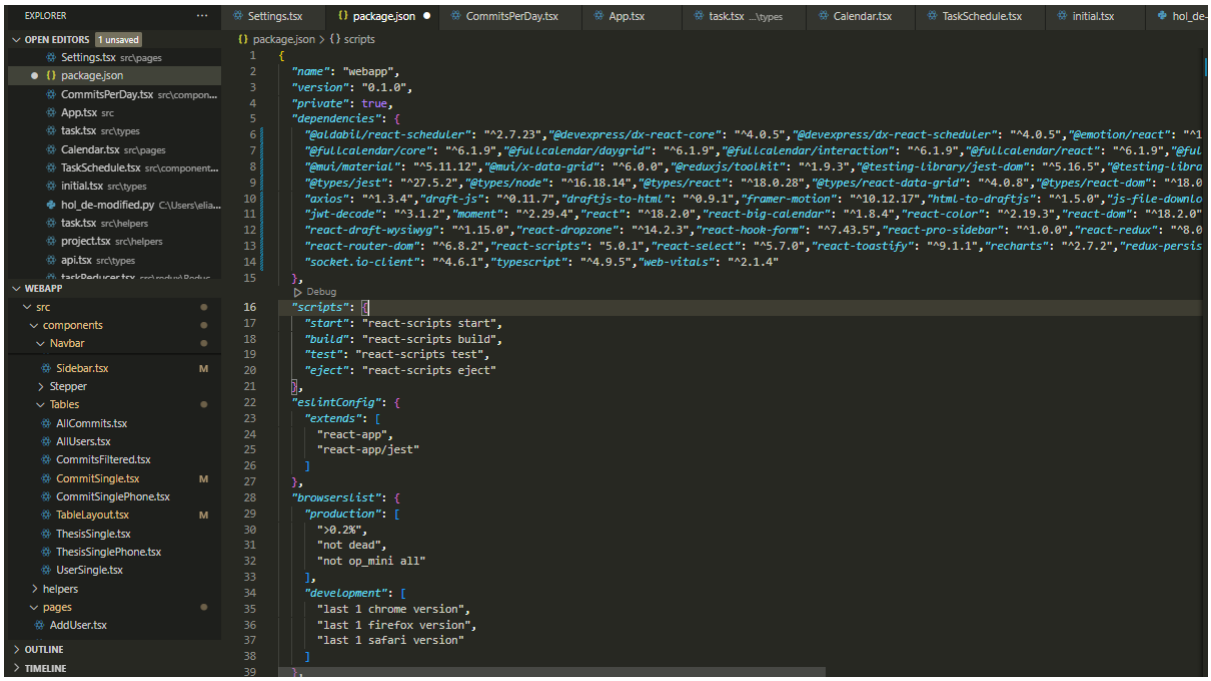
Για την υλοποίηση του πελάτη, έγινε χρήση της ReactJS. Η επιλογή αυτή βασίστηκε σε κριτήρια που αναλύθηκαν σε προηγούμενο κεφάλαιο, ωστόσο όπως αναλύθηκε για τον εξυπηρετητή, θα αναλυθεί και εδώ η δομή των αρχείων, οι βασικές βιβλιοθήκες, δίνοντας περισσότερη έμφαση στην διατήρηση των δεδομένων στον φυλλομετρητή.

4.4.1 Βιβλιοθήκες Πελάτη

Μερικές από τις πιο βασικές βιβλιοθήκες, καθώς και η λειτουργία τους είναι οι εξής:

- @mui/material(https://mui.com/): Η βιβλιοθήκη αυτή παρέχει μερικά έτοιμα components για να γίνει πιο εύκολη η δημιουργία του UI.
- Axios(https://www.npmjs.com/package/axios): Μας παρέχει την δυνατότητα να κάνει αιτήματα ο πελάτης προς κάποιον εξυπηρετητή.
- Framer-motion(https://www.framer.com/motion/): Η βιβλιοθήκη αυτή αποτελεί την δημοφιλέστερη για την υλοποιήση animation σε μία εφαρμογή με χρήση ReactJS
- Draft-js(https://draftjs.org/): Η χρήση της είναι να παρέχει ένα πλούσιο διαχειριστή κειμένου για την καλύτερη δημιουργία ηλεκτρονικών μηνυμάτων
- React-hook-form(https://react-hook-form.com/): Αποτελεί την πιο ελαφριά βιβλιοθήκη για την δημιουργία φόρμας. Βασικό της προνόμιο είναι τα ελάχιστα rerender που δημιουργεί στην εφαρμογή.
- React-redux(https://react-redux.js.org/): μία από τις βασικότερες βιβλιοθήκες για την εφαρμογή αφού διατηρεί όλα τα δεδομένα χωρίς να χρειάζεται να γίνονται συνέχεια αιτήματα από τον πελάτη στον εξυπηρετητή.
- Redux-persist(https://www.npmjs.com/package/redux-persist):συμπληρωματική βιβλιοθήκη της react-redux, κρατώντας τα δεδομένα ακόμα και μετά από κλείσιμο της εφαρμογής ή ανανέωσης της.
- React-router-dom(https://reactrouter.com/en/main): βασική λειτουργία της είναι η πλοήγηση μεταξύ των σελίδων της εφαρμογής.
- Recharts(https://recharts.org/en-US/): η βιβλιοθήκη αυτή παρέχει κάποια αίτημα γραφήματα
- Typescript(https://www.typescriptlang.org/): η χρήση της δεν είναι απαραίτητη, ωστόσο παρέχει σαφήνεια στον κώδικα, αφού εισάγει τον όρο strong typing. Γνωρίζοντας ότι η Javascript, όπως και η Python, δεν νοιάζονται για το είδος της μεταβλητής, η βιβλιοθήκη αυτή εισάγει τον όρο type, που κάνει πιο σαφή τον κώδικα.

Οι βιβλιοθήκες αυτές αποθηκεύονται στο package.json αρχείο το οποίο έχει την εξής μορφή



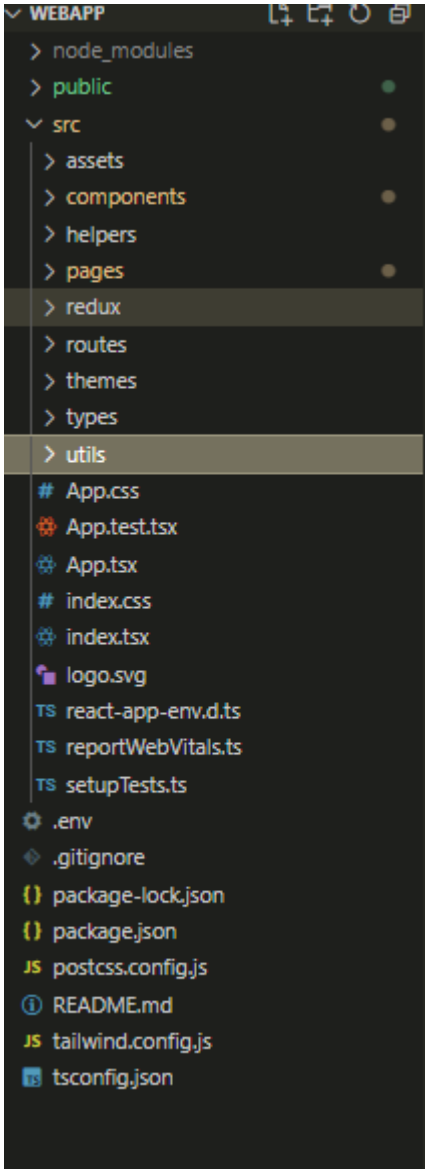
Εικόνα 77: package.json πελάτη

4.4.2 Δομή αρχείων Πελάτη

Η δομή των αρχείων στην πλευρά του πελάτη ακολουθεί παρόμοιο μοτίβο με αυτό του εξυπηρετητή. Πιο αναλυτικά, ο κώδικας χωρίζεται σε φακέλους για την κατανόηση του κάθε αρχείου, χωρίς αυτό όμως να είναι απαραίτητο στην react. Τα αρχεία με τους φακέλους είναι τα εξής:

- .env: για την χρήση μεταβλητών περιβάλλοντος
- Src/assets: περιέχει εικόνες που χρησιμοποιούνται μέσα στην εφαρμογή. Οι εικόνες αυτές θα μπορούσαν να είναι και στον public φάκελο
- Src/components: περιέχει τα επιμέρους κομμάτια του κώδικα που συνθέτουν τις σελίδες της εφαρμογής
- Src/pages: οι σελίδες της εφαρμογής
- Src/redux: περιέχει μέσα τους reducer που είναι υπεύθυνοι για την διατήρηση των δεδομένων. Ωστόσο επειδή η λειτουργία τους είναι πιο πολύπλοκη θα αναλυθεί σε ξεχωριστό κεφάλαιο.
- Src/routes: ο κώδικας που είναι υπεύθυνος για την πλοήγηση στην εφαρμογή
- Src/themes: περιέχει τα θέματα με τα χρώματα, παρέχοντας μία σταθερότητα στο θέμα της εφαρμογής.
- Src/types: ξεχωριστά αρχεία με τους τύπους των δεδομένων για την typescript
- Src/utils: συναρτήσεις που χρησιμοποιούνται σε περισσότερα σημεία του κώδικα, επομένως είναι σε ξεχωριστό φάκελο.

Οπτικά η δομή των αρχείων είναι η εξής



Εικόνα 78: Δομή αρχείων πελάτη

4.4.3 Φόρμες και Αίτηματα Πελάτη

Η εφαρμογή αποτελείται από πολλές φόρμες και αιτήματα από τον πελάτη προς τον εξυπηρετητή (δημιουργία λογαριασμου, διπλωματικής, commit, αρχείων κτλπ.). Επομένως, αξίζει να αναπτύξουμε ένα από αυτά τα αιτήματα μαζί με την φόρμα του, για να γίνει πιο κατανοητό πως μεταφέρονται τα δεδομένα από τον πελάτη στον εξυπηρετητή.

Χαρακτηριστικό παράδειγμα είναι η φόρμα δημιουργίας commit. Για αρχή, έχει δημιουργηθεί ένα component CustomInput, προκειμένου να υπάρχει μία σταθερότητα στην εφαρμογή σχετικά με τις εισόδους δεδομένων. Το component αυτό παίρνει ως props την μεταβλητή στην οποία αναφέρεται, το label, το πιθανό λάθος σε περίπτωση που αποτύχει

κάποιος από τους κανόνες validation, το placeholder, κάποιες ρυθμίσεις σχετικά με το είδος των δεδομένων, καθώς και το μέγεθος του input.

```tsx
const CustomInput: React.FC<InputProps> = ({register,name,error,label,wrapperClass,placeholder,options,large,...rest}) => {
  const theme = useTheme();
  const colors = token(theme.palette.mode);

  return (
    <>
      <div className="flex flex-col gap-2 mt-2 w-full">
        <label>
          <Typography
            variant="h5"
            color={error ? colors.redAccent[400] : colors.gray[100]}
          >{error ? error : label? label:name.toLowerCase()}</Typography>
        </label>
        {!large ? (
          <input
            {...rest}
            placeholder={placeholder}
            {...register(name, options)}
            className={`p-2  outline-none rounded-md text-gray-800 ${
              error ? "border-2 border-red-500" : ""
            }`}
            style={{ border: colors.primary[400] }}
          />
        ) : (
          <textarea
            cols="100"
            {...rest}
            placeholder={placeholder}
            {...register(name, options)}
            className={`p-2  outline-none rounded-md text-gray-800 ${
              error ? "border-2 border-red-500" : ""
            }`}
            style={{ border: colors.primary[400] }}
          />
        )}
      </div>
```

Εικόνα 79: CustomInput

Με παρόμοια λογική έχει δημιουργεί και ένα component που λέγεται FormLayout, το οποίο κάνει map τα children που του έχουν δοθεί και δημιουργεί μία φόρμα. Ως props δέχεται τα παιδιά, την συμβολοσειρά που θα δείχνει το κουμπί (πχ Submit, Create, Edit), την συνάρτηση που θα τρέξει όταν ολοκληρωθεί η φόρμα, καθώς και το register προκειμένου να ψάξει για τυχόν validation errors.

```
const FormLayout: React.FC<Props> = ({
  defaultValues,
  buttonLabel,
  children,
  onSubmit,
  handleSubmit,
  register,
  ...rest
}) => {
  const theme = useTheme();
  const colors = token(theme.palette.mode);

  return (
    <form onSubmit={handleSubmit(onSubmit)} {...rest}>
      <div>
        {Array.isArray(children)
          ? children.map((child) => {
              return child.props.name
                ? React.createElement(child.type, {
                    ...{
                      ...child.props,
                      register,
                      key: child.props.name,
                    },
                  })
                : child;
            })
          : children}
      </div>
      {buttonLabel ? (
```

Εικόνα 80: FormLayout

```
        <Button
          type="submit"
          sx={{
            paddingY: 2,
            paddingX: 3,
            marginX: "auto",
            display: "block",
            background: colors.greenAccent[600],
            textTransform: "uppercase",
          }}
          className={`rounded-md  px-12 mt-8 hover:bg-gray-700 transition-all duration-500`}
          onClick={() => {}}
        >
          <Typography
            variant="h5"
            color={colors.gray[100]}
            textTransform="uppercase"
          >
            {buttonLabel}
          </Typography>
        </Button>
      ) : null}
    </form>
  );
};
```

Εικόνα 81: FormLayout 2

Με την χρήση αυτών των δύο component, δημιουργούνται όλες οι φόρμες της εφαρμογής. Χαρακτηριστικό παράδειγμα είναι η φόρμα δημιουργία commit, η οποία είναι ένα modal, επομένως χρειάζεται τιμή για να ξέρει πότε ανοίγει καθώς και ένα set state για να μπορέσει να κλείσει. Επιπλέον, έχει ένα state για να κρατάει τα αρχεία και να τα ενημερώνει προτού τα στείλει στον εξυπηρετητή. Ο κώδικας για την επίτευξη της φόρμας είναι ο εξής:

```
export default function CreateCommit({ isOpen, setIsOpen, showSnack }: Props) {
  const { user } = useSelector((state: state) => state);
  const [file, setFile] = useState<File[]>([]);

  const {
    register,
    handleSubmit,
    reset,
    formState: { errors },
  } = useForm();

  const onSubmit = (data: any) => {
    const formData = new FormData();
    console.log(data.description);
    formData.append("title", data.title);
    formData.append("description", data.description);
    formData.append("user", JSON.stringify(user.id));

    if (file.length > 0) {
      for (let f of file) {
        console.log(f);
        formData.append("files", f);
      }
    } else {
      formData.append("files", file[0]);
    }

    createCommit(formData)
      .then((res) => {
        setFile([]);
```

Εικόνα 82: CreateCommit

```
        reset();
        setIsOpen(false);

        showSnack("Commit was created!", true);
      })
      .catch((error) => {
        setFile([]);
        reset();
        setIsOpen(false);
        showSnack(error.response.message, false);
      });
  };

  return (
    <Modal setIsOpen={setIsOpen} isOpen={isOpen} title="Commit">
      <FormLayout
        buttonLabel="New Commit"
        register={register}
        handleSubmit={handleSubmit}
        onSubmit={onSubmit}
        className=" mx-auto py-2 mt-4 w-full gap-12"
      >
        <div className="flex flex-col justify-center items-center lg:px-64">
        <CustomInput
          name="title"
          type="text"
          placeholder="Enter commit title"
```

Εικόνα 83: CreateCommit 2

```
        error={errors.title?.message}
        autoFocus
        register={register}
        options={{ required: "Title required." }}
      />

      <CustomInput
        name="description"
        type="text"
        placeholder="Description"
        error={errors.description?.message}
        autoFocus
        large={true}
        register={register}
        options={{ required: "Description required." }}
      />
      <CustomInput
        name="Files"
        type="file"
        id="input-file-upload"
        multiple={true}
        placeholder="Enter description"
        autoFocus
        register={register}
      />
      </div>
      <div className=" flex flex-col justify-center items-center mb-12">
        <DragDropFile file={file} setFile={setFile} />
      </div>
```

Εικόνα 84: CreateCommit3

Στην ουσία, υπάρχει η συνάρτηση onSubmit, που μαζεύει τα δεδομένα προτού καλέσει την createCommit για να γίνει το αίτημα στον εξυπηρετητή. Επιπλέον γίνεται χρήση των component που αναφέρθηκαν προηγούμενως, για να υπάρχει μία σταθερότητα στην φόρμα. Αξίζει να αναφερθούμε λίγο και στην createCommit, για να δούμε πως ο πελάτης επικοινωνεί με τον εξυπηρετητή. Αρχικά μιλάμε για μία συνάρτηση που επιστρέφει ένα Promise, καθώς είναι ασύγχρονη. Έχει δημιουργηθεί ένα axios instance, προκειμένου τα αιτήματα να γίνονται πιο εύκολα και να υπάρχει μία συνοχή στον κώδικα. Το instance αυτό κρατάει το url του εξυπηρετητή καθώς και την πόρτα, και μερικά headers προκειμένου να μην αποτύχει το αίτημα λόγω λανθασμένης μορφής των δεδομένων.

```
1  import axios from "axios";
2
3  export const instance = axios.create({
4    baseURL: "http://127.0.0.1:4000/api",
5    headers: {
6      "Content-Type": "application/json",
7      timeout: 0,
8    },
9  });
10
```

Εικόνα 85: Axios instance

Στην συνέχεια έχει δημιουργηθεί ένα enum, προκειμένου να έχουμε συγκεντρωμένα όλα τα endpoints του εξυπηρετητή

```
export enum ApiKind {
  LOGIN = "/auth/signin",
  LOGINGOOGLE = "/auth/signin/google",
  CREATEUSER = "/auth/signup",
  CREATECOMMENT = "/comment/create/",
  CREATETHESIS = "/project/create",
  UPDATETHESIS = "/project/update",
  GETCOMMITS = "/commit/",
  GETCOMMENTS = "/comment/",
  CREATECOMMIT = "/commit/create/",
  DELETECOMMIT = "/commit/",
  GETTHESIS = "/project/",
  CREATEUSEROPEN = "/auth/signup/user",
  GETALLUSERS = "/users/all",
  DELETEUSER = "/users",
  UPDATEROLEUSER = "/users",
  CREATETASK = "/task/create/",
  IMAGE = "http://127.0.0.1:8000/",
}
```

Εικόνα 86: Enum endpoints

Τέλος αφού έχουμε όλα τα επιμέρους στοιχεία το αίτημα πραγματοποιείται με τον εξής κώδικα.

```
export const createCommit = function (data: any) {
  console.log(data);
  const state = store.getState();

  return new Promise<string>((resolve, reject) => {
    if (state.thesis) {
      instance({
        url: ApiKind.CREATECOMMIT + state.thesis.id,
        method: "POST",
        data: data,
        headers: {
          "x-access-token": state.user.token,
          "Content-Type": "multipart/form-data",
        },
      })
        .then((res) => {
          console.log(res);
          store.dispatch({
            type: commitActionKind.CREATECOMMIT,
            payload: res.data.commit,
          });
          resolve(res.data.message);
        })
        .catch((error) => {
          reject(error);
        });
    }
  });
};
```

Εικόνα 87: createCommit ασύγχρονη συνάρτηση

Σε περίπτωση επιτυχίας του αιτήματος γίνεται resolve το promise με το μήνυμα που μας γυρίζει ο εξυπηρετητής. Σε αντίθετη περίπτωση γίνεται reject και παίρνουμε ως μήνυμα το

λάθος που επιστρέφει ο εξυπηρετητής. Το αίτημα που αποστέλλεται στον εξυπηρετητή είναι το εξής.

```
----------------------------922988580721235436124943014 8
Content-Disposition: form-data; name="title"

test
----------------------------922988580721235436124943014 8
Content-Disposition: form-data; name="description"

test
----------------------------922988580721235436124943014 8
Content-Disposition: form-data; name="user"

18
----------------------------922988580721235436124943014 8
Content-Disposition: form-data; name="files"; filename="6.png"
Content-Type: image/png
```

Εικόνα 88: Δεδομένα αιτήματος

Και απάντηση του εξυπηρετητή είναι το καινούργιο commit, που δημιουργήθηκε με επιτυχία

Εικόνα 89: Απάντηση αιτήματος

4.4.4 Αυθεντικόποιηση Πελάτη

Σε προηγούμενο κεφάλαιο αναλύθηκε πως ο εξυπηρετής διαχειρίζεται την αυθεντικοποίηση, ωστόσο πρέπει να αναφερθούμε και στην μεριά του πελάτη και στον τρόπο που κρύβονται routes, για να μην υπάρχει μη αυθεντικοποιήμενη πρόσβαση σε πόρους που δεν πρέπει. Η τεχνική που χρησιμοποιηθήκε για την επίτευξη αυτού του στόχου, είναι των private και public routes. Πιο συγκεκριμένα, υπάρχουν routes που δεν υπάρχει έλεγχος και υπάρχουν και αυτά που απαιτούν να είσαι αυθεντικοποιημένος χρήστης προκειμένου να έχει πρόσβαση.

```
<BrowserRouter>
  <Routes>
    <Route element={<PrivateRoute />}>
      <Route element={<Home />} path="/" />
      <Route element={<AddUser />} path="/adduser" />
      <Route element={<Users />} path="/users" />
      <Route element={<CreateThesis />} path="/thesis" />
      <Route element={<Commits />} path="/commits" />
      <Route element={<Settings />} path="/settings" />
      <Route element={<Mail />} path="/mail" />
      <Route element={<Statistics/>} path="/statistics"/>
      <Route element={<Calendar/>} path="/calendar"/>
    </Route>

    <Route element={<PublicRoute />}>
      <Route path="/login" element={<Login />} />
      <Route path="/register" element={<Register />} />
    </Route>
    <Route path="/login" element={<Login />} />
```

Τα public routes, κοιτάτε στο state (θα αναλυθεί σε επόμενο κεφάλαιο) αν ο χρήστης είναι αυθεντικοποιημένος, και δρουν αναλόγως. Σε περίπτωση που είναι, τον οδηγούν στο Home Page, σε αντίθετη περίπτωση δημιουργούν τα routes “παιδιά” του

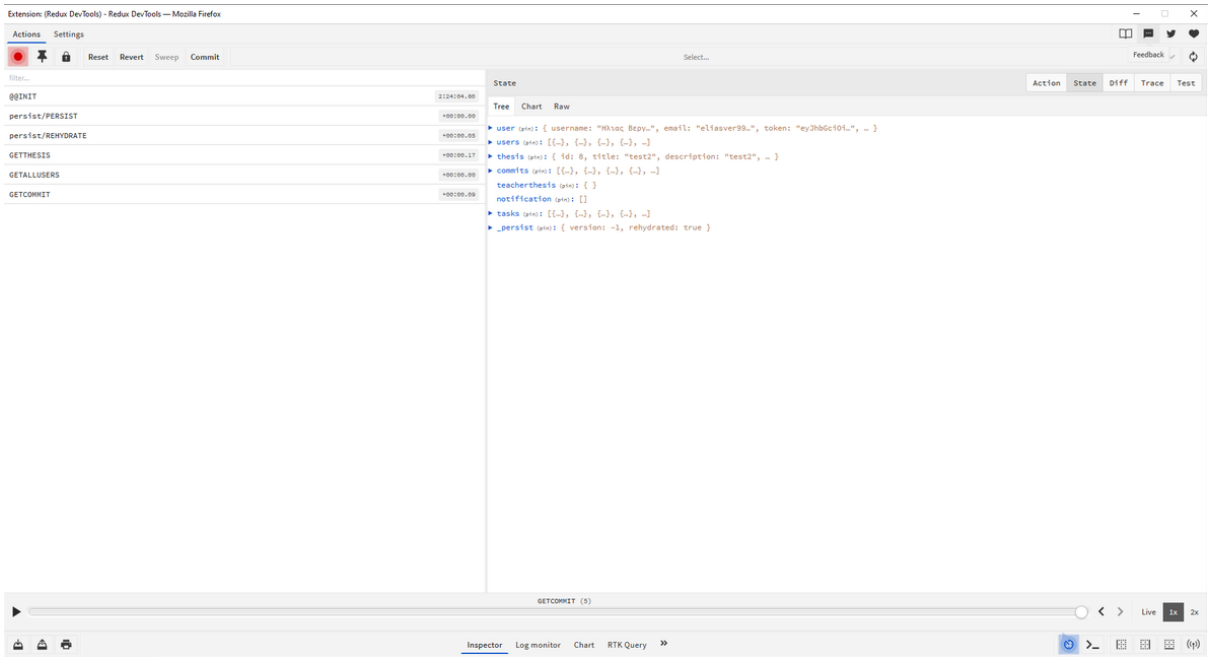
```
import React from "react";
import { useSelector } from "react-redux/es/exports";
import { Outlet, Navigate } from "react-router-dom";
import { state } from "../types/initial";

const PublicRoute = ({ children, ...rest }: any) => {
  let { auth } = useSelector((state: state) => state.user);
  return !auth ? <Outlet /> : <Navigate to="/" />;
};

export default PublicRoute;
```

Το ίδιο ακριβώς κάνει και το private route, με την διαφορά πως αν δεν είναι αυθεντικοποιημένος τον οδηγούν στο login url

```
import { useSelector } from "react-redux/es/exports";
import { Outlet, Navigate } from "react-router-dom";
import { state } from "../types/initial";

const PrivateRoute = ({ children, ...rest }: any) => {
  let { auth } = useSelector((state: state) => state.user);
  return auth ? <Outlet /> : <Navigate to="/login" />;
};

export default PrivateRoute;
```

4.4.5 State Management

Για αρχή, πρέπει να γίνει κατανοητό για ποιο λόγο είναι απαραίτητο να υπάρχει ένα state management σε μία εφαρμογή που μπορεί να φτάσει να έχει πολλά δεδομένα. Η React γνωρίζουμε πως μεταφέρει τα δεδομένα από το component πατέρα στα παιδιά του μέσω των props. Ωστόσο σε περίπτωση που θέλουμε να μεταφέρουμε σε πολλά επίπεδα κάτω τα δεδομένα αυτή η τεχνική κρύβει κινδύνους. Το πρόβλημα αυτό ονομάζεται prop drilling, και μπορεί να προκαλέσει πολλά rerenders, κάνοντας την εφαρμογή αργή, δημιουργώντας μία άσχημη αίσθηση για τον χρήστη της εφαρμογής. Το πρόβλημα αυτό μπορεί να αντιμετωπισθεί με την χρήση κάποιου state management. Παρότι η React παρέχει πρόσβαση στο local storage του φυλλομετρητή, η χρήση αυτού είναι δύσκολη για τον προγραμματιστή, αφού όλα τα δεδομένα μετατρέπονται σε μία συμβολοσειρά και δεν είναι εύκολα κατανοητά. Επομένως, η λύση που καταλήξαμε είναι αυτή του Redux. Το redux συνοπτικά έχει έναν χώρο που κρατάει τα δεδομένα, και κάθε component μπορεί να τα πάρει από εκεί. Για να γίνει πιο κατανοητό θα αναλύσουμε κάποια κομμάτια κώδικα, καθώς και το πως είναι αποθηκευμένα τα δεδομένα.



Εικόνα 90: Redux toolkit

Στο αριστερό πάνελ φαίνεται ποια ενέργεια έχει γίνει και τι δεδομένα προσθέτει στην μνήμη και δεξιά παρατηρούμε όλα τα δεδομένα της εφαρμογής. Όπως αναφέραμε αυτή η τεχνική και μειώνει τα αιτήματα που γίνονται στον εξυπηρετητή και αποφεύγουμε πιθανά

προβλήματα που θα δημιουργούσε να περνάμε τα δεδομένα μέσω των component. Πιο συγκεκριμένα, αυτή την στιγμή κάθε component στην εφαρμογή που είναι εντός του Redux Provider, έχει πρόσβαση σε αυτές τις τιμές που φαίνονται στην εικόνα.

Για να υλοποιηθεί αυτό όμως απαιτούνται κάποια συγκεκριμένα βήματα. Πρώτα πρέπει να αρχικοποιηθεί το store, μαζί με τα αρχεία που του δηλώνουν τις ενέργειες. Στην εφαρμογή αυτό υλοποιείται στο store.tsx.

```
const combReduces = combineReducers({
  user: userReducer,
  users: usersReducer,
  thesis: thesisReducer,
  commits: commitReducer,
  teacherthesis: teacherThesisReducer,
  notification: notificationReducer,
  tasks: taskReducer,
});

const persistedReducer = persistReducer(persistConfig, combReduces);
export const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware: any) =>
    getDefaultMiddleware({
      immutableCheck: false,
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
      },
    }),
});
```

Εικόνα 91: Δημιουργία redux store

Αφού δημιουργηθεί το store, πρέπει να εμφωλεύσουμε την εφαρμογή σε αυτό, προκειμένου να έχουν όλα τα components πρόσβαση στα δεδομένα. Αυτό γίνεται στο υψηλότερο επίπεδο της εφαρμογής, δηλαδή στο index.tsx

```
<Provider store={store}>
  <PersistGate loading={null} persistor={persistor}>
    <App />
  </PersistGate>
</Provider>
```

Εικόνα 92: Redux provider

Μετά θα ακολουθεί ένα παράδειγμα κάποια ενέργειας, που στην ουσία περνάει στην μνήμη τα δεδομένα ή κάνει κάποια ανανέωση εγγραφής.

```
    case thesisActionKind.CREATETHESISSTUDENT:
      state.id = payload.id;
      state.title = payload.title;
      state.description = payload.description;
      state.students = payload.students;
      state.teacher = payload.teacher;
      state.status = payload.status;
      state.semester = payload.semester;
      state.year = payload.year;
      return state;
    case thesisActionKind.LOGOUT:
      return {
        ...initialState,
      };
```

Εικόνα 93: Reducer action

Στην ουσία υπάρχει μία αρχική κατάσταση στην οποία έχουμε δώσει αρχικές τιμές και μετά βάζουμε τις τιμές ανάλογα με τα δεδομένα που θα επιστρέψει ο εξυπηρετητής στον πελάτη. Προκειμένου να καλέσουμε την ενέργεια αυτή, όταν κάνουμε το αίτημα περνάμε σε μία συνάρτηση του store το payload, δηλαδή τα δεδομένα που θέλουμε να μπούνε στην αρχική κατάσταση

```
store.dispatch({
  type: thesisActionKind.CREATETHESIS,
  payload: res.data.project,
});
resolve(res.data.message);
```

Εικόνα 94: Κλήση action

Τέλος για να έχουμε πρόσβαση σε αυτά τα δεδομένα, αρκεί σε κάθε component να καλέσουμε την συνάρτηση useSelector, και να πάρουμε όποια δεδομένα χρειαζόμαστε.

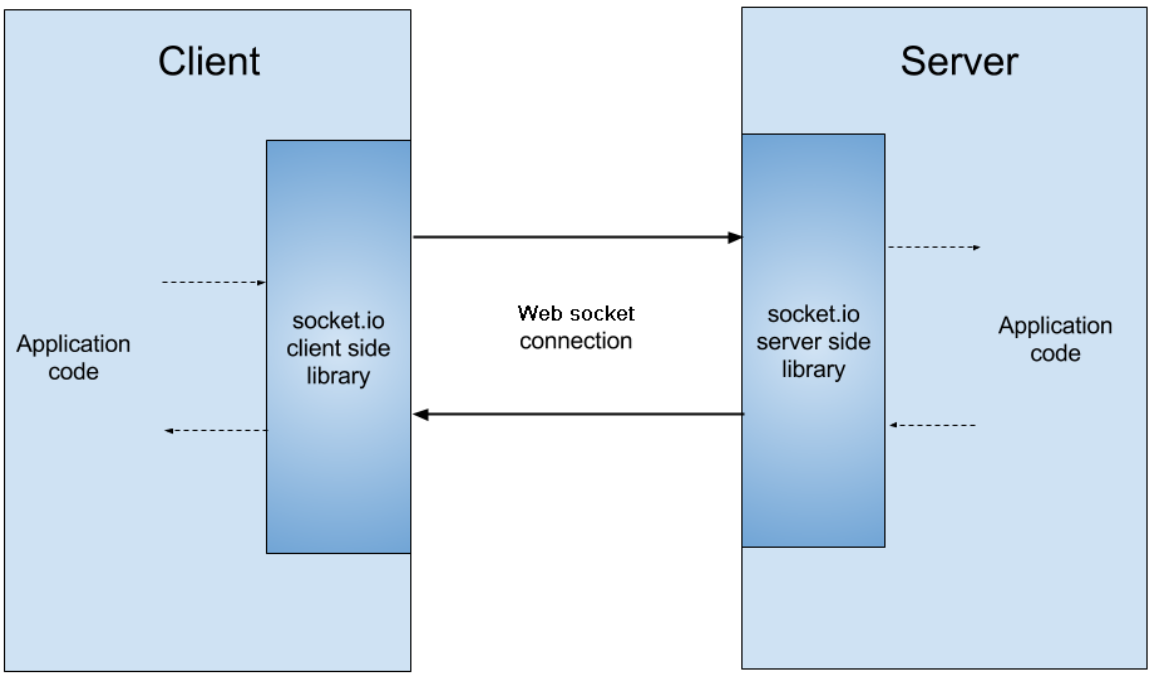
```
const { users } = useSelector((state: state) => state);
```

Εικόνα 95: Επιλογή δεδομένων redux

Στο συγκεκριμένο παράδειγμα παίρνουμε τα δεδομένα μέσα στους χρήστες.

4.5 Λειτουργία ζωντανής συνομιλίας

Αφού αναλύθηκαν κάποιες από τις βασικές λειτουργίες τόσο του πελάτη όσο και του εξυπηρετητή, τώρα πρέπει να μιλήσουμε για την λειτουργία της ζωντανής συνομιλίας. Η λειτουργία αυτή αποτελεί ένα ξεχωριστό υποκεφάλαιο, αφού ο τρόπος που λειτουργεί δεν είναι η κλασσική αρχιτεκτονική εξυπηρετητή πελάτη. Όπως αναλύσαμε και σε προηγούμενο κεφάλαιο, ο πελάτης κάνει αιτήματα στον εξυπηρετητή και εκείνος του στέλνει την απάντηση. Τι συμβαίνει όμως σε περίπτωση που θέλουμε να επικοινωνήσουν δύο πελάτες μεταξύ τους. Αυτό είναι αδύνατο να συμβεί με την αρχιτεκτονική πελάτη-εξυπηρετητή, αφού ο πελάτης «ακούει» μόνο σε αιτήματα που έχει κάνει ο ίδιος και περιμένει απάντηση. Με άλλα λόγια δεν μπορεί να ξέρει πότε το εξυπηρετητής θα του στείλει μόνος του μήνυμα για να ξεκινήσει μία συνομιλία. Αυτό το πρόβλημα λύνεται με την εισαγωγή του όρου socket.



Εικόνα 96: Μοντέλο επικοινωνίας με socket

Η βασική ιδέα είναι η δημιουργία μίας σύνδεσης μέσω web socket, και μέσω αυτού στέλνονται μηνύματα και προς τις δύο κατευθύνσεις. Στην εφαρμογή αυτή όμως δεν χρησιμοποιούμε μόνο σύνδεση ένας προς ένα, αλλά δημιουργείται ένα δωμάτιο συνδέσης, έτσι ώστε να μπορεί να επικοινωνεί ο καθηγητής με τους πιθανούς δύο φοιτητές που έχουν μία διπλωματική.
Το σημείο του κώδικα που υλοποιεί αυτό το κομμάτι στον εξυπηρετητή είναι

```
const io = new Server(server, {
  cors: {
    origin: ["http://localhost:3000", "http://localhost:4000"],
    methods: "GET,POST,PUT,DELETE,OPTIONS",
    credentials: true,
  },
});
const PORT = process.env.APP_PORT;
```

Εικόνα 97: Αρχικοποίηση Server

Στο σημείο αυτό αρχικοποιείται το είδος σύνδεσης μαζί με τα url που μπορεί να δεχτεί, (πόρτα 3000 είναι ο πελάτης και πόρτα 4000 ο εξυπηρετητής). Επιπλέον ορίζουμε τις μεθόδους που θα δέχεται, καθώς και αν χρειάζεται αυθεντικοποίηση για την επικοινωνία.

```
io.on("connection", (socket) => {
  socket.on("join_room", (data) => {
    const { username, room } = data;
    console.log("user connected to socket!!!!!!!!!!!!");
    socket.join(room);
    let __createdtime__ = Date.now(); // Current timestamp
    // Send message to all users currently in the room, apart from the user that just joined
  });
  socket.on("send_message", (data) => {
    const { message, username, room, __createdtime__ } = data;
    io.in(room).emit("receive_message", data); // Send to all users in room, including sender
    socket.broadcast.in(room).emit("receive_notification", data);
  });
  socket.on("disconnect", () => {
    console.log("user disconnected from socket !!!!!!!!!!!!");
    socket.disconnect();
  });
  socket.on("message", (data) => {
    socket.to(data.room).emit("messageResponse", data);
    // io.emit("messageResponse", data);
  });
});
```

Εικόνα 98: Socker Listener και συναρτήσεις

Οι μέθοδοι που χρησιμοποιούνται είναι τρεις. Αρχικά είναι η είσοδος σε ένα δωμάτιο επικοινωνίας. Μετά είναι η αποστολή μηνύματος, που γίνεται μόνο μεταξύ χρηστών που βρίσκονται στο ίδιο δωμάτιο, και τέλος είναι η αποσύνδεση από το δωμάτιο.

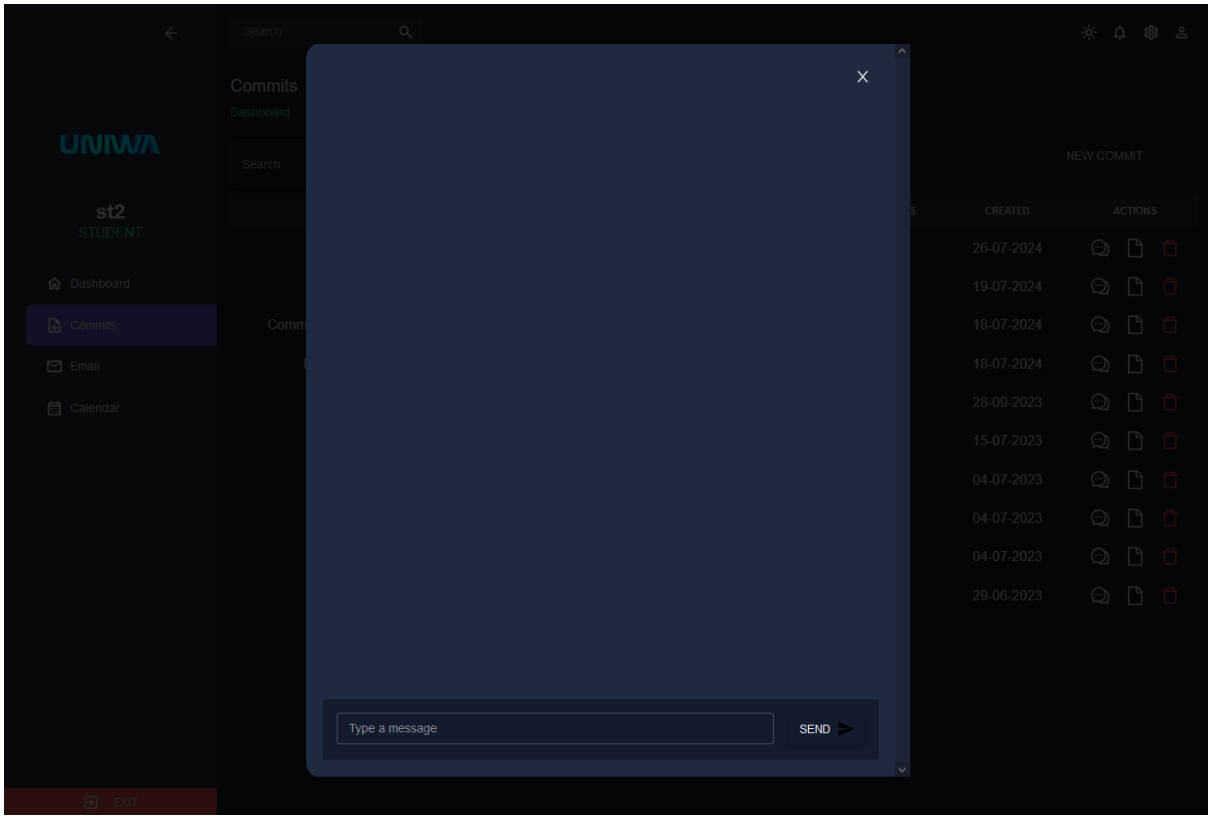
Στην πλευρά του πελάτη, η σύνδεση γίνεται όταν μπει να δει τις καταχωρίσεις του, όπου και δημιουργούνται δωμάτια ίσα με τις καταχωρίσεις. Ο κώδικας που υλοποιεί την δημιουργία αλλά και την σύνδεση είναι ο εξής

```
React.useEffect(() => {
  const ENDPOINT = "http://localhost:4000";
  setSocket(socketIOClient(ENDPOINT));
  setLoading(true);
}, []);
React.useEffect(() => {
  if (socket) {
    socket.emit("join_room", { username: user.username, room: commit.id });
  }
}, [loading, commit.id]);
React.useEffect(() => {
  if (socket) {
    socket.on("receive_message", (data: any) => {

      if (!open && data.username != user.username) {
        setMessages((state: any) => [
          ...state,
          {
            message: data.message,
            username: data.username,
            __createdtime__: data.__createdtime__,
            read: false,
          },
        ]);
      } else {
        setMessages((state: any) => [
          ...state,
          {
            message: data.message,
            username: data.username,
            __createdtime__: data.__createdtime__,
```

Εικόνα 99: Υλοποίηση socket στον πελάτη

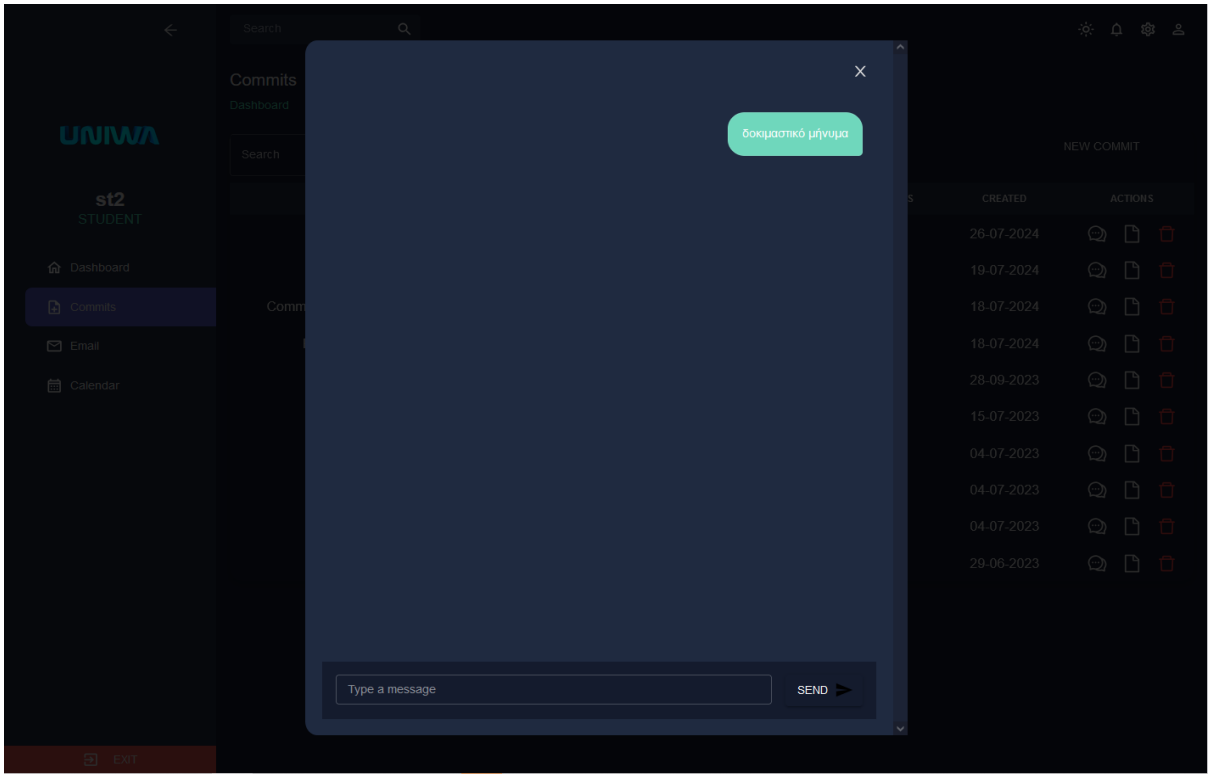
Η διαδικασία στο επίπεδο της εφαρμογής είναι η εξής:

- Αρχικά μπαίνει ο φοιτητής σε ένα δωμάτιο commit, αυτό ενεργοποιεί ένα κανάλι στον εξυπηρετητη.



Εικόνα 100: Δημιουργία δωματίου UI

- Αφού γίνει η ενεργοποίηση του δωματίου, ο φοιτητής μπορεί να στείλει μήνυμα, αυτό το μήνυμα, δεν μένει μόνο στο δωμάτιο αλλά περνίεται και στην βάση δεδομένων.

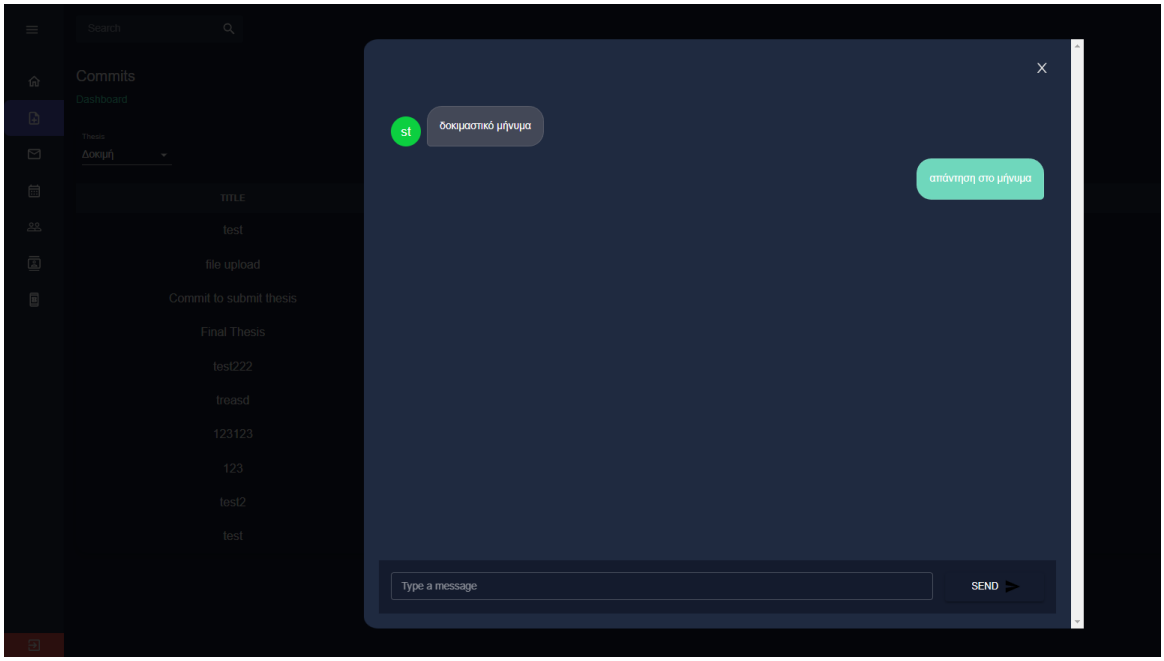


Εικόνα 101: Αποστολή μηνύματος UI

- Στην συνέχεια ο διδάσκοντας της συγκεκριμένης διπλωματικής ενημερώνεται για την ύπαρξη ενός καινούργιου μηνύματος.

Εικόνα 102: Ειδοποίηση νέου μηνύματος UI

- Μετά μπορεί να απάντησει και ο διδάσκοντας, συνεχίζοντας με τον τρόπο αυτό την επικοινωνία.



Εικόνα 103: Απάντηση στο μήνυμα

4.6 Ανάρτηση Αρχείων

Άλλη μία κομβική υπηρεσία της εφαρμογής είναι η ανάρτηση αρχείων. Είναι ξεκάθαρο πως μία τέτοια εφαρμογή χωρίς την δυνατότητα ο χρήστης να ανεβάζει αρχεία στον server δεν θα είχε κανένα απολύτως νόημα. Αυτή η λειτουργία επιτυγχάνεται και από τις δύο πλευρές (πελάτη-εξυπηρετητή). Αρχικά ο πελάτης στέλνει ένα αίτημα με τα αρχεία σε μορφή filedata

```
if (file.length > 0) {
  for (let f of file) {
    console.log(f);
    formData.append("files", f);
  }
} else {
  formData.append("files", file[0]);
}
```

Εικόνα 104: Formdata file handling

Έτσι δημιουργείται ένα αίτημα της μορφής:

```
----------------------------384946009026783397173299408788
Content-Disposition: form-data; name="title"

file upload
----------------------------384946009026783397173299408788
Content-Disposition: form-data; name="description"

file upload
----------------------------384946009026783397173299408788
Content-Disposition: form-data; name="user"

18
----------------------------384946009026783397173299408788
Content-Disposition: form-data; name="files"; filename="package.json.png"
Content-Type: image/png
```

Εικόνα 105: Αίτημα ανάρτηση αρχείων

Στην συνέχεια ο εξυπηρετητής είναι αυτός που θα περάσει το αρχείο τόσο στην βάση, χρησιμοποιώντας ως filename το path, για να μπορεί μετά ο πελάτης να το καλεί, όσο και στα αρχεία του. Αυτό επιτυγχάνεται με το εξής script:

```
const savePath = path.join(
  "public/files/",
  `${project.id}/`,
  `${commit.id}`
);
fs.mkdirSync(savePath, { recursive: true }, (err) => {
  if (err) res.status(500).send({ message: err.message });
});
let arr = [];
if (images.length) {
  for (let index = 0; index < images.length; index++) {
    images[index].mv(savePath + "/" + images[index].name,(err) => {
      if (err) {res.status(500).send({ message: err.message });}
    }
    );
    const object = {
      fileName: images[index].name,
      filePath: savePath,
      type: images[index].mimetype.split("/")[1],
    };
    arr.push(object);
  }
```

Εικόνα 106: Υλοποιήση εξυπηρετητή αρχείων

Στην ουσία, ο εξυπηρετητής δημιουργεί πρώτα το commit, και με βάση το commit και το project id δημιουργεί έναν φάκελο που θα αποθηκευτούν τα αρχεία. Θεωρητικά αυτό το βήμα θα μπορούσε να απουσιάζει και να αποκτά ένα τυχαίο όνομα κάθε αρχείο και να είναι όλα στον ίδιο φάκελο. Ωστόσο, αυτή η τεχνική προσφέρει μία καλύτερη οργάνωση στα αρχεία, σε περίπτωση απώλειας κάποιου αρχείου. Στην συνέχεια κάνει μία επανάληψη που μεταφέρει τα αρχεία μέσα στον φάκελο και δημιουργεί μία εγγραφή στον πίνακα. Τέλος ο

πίνακας κάνει bulk create στην βάση δεδομένων, και τα αρχεια γίνονται attach στο commit που ανήκουν.

```
F.bulkCreate(arr).then((files) => {
  commit.setFiles(files).then((commit) =>
    Commit.findOne({
      where: { id: commit.id },
      attributes: { exclude: ["userId", "projectId"] },
      include: [
        {
          model: User,
          as: "user",
          attributes: { exclude: ["password"] },
        },
        {
          model: Project,
          as: "project",
        },
        { model: F, as: "files" },
      ],
    }).then((c2) => res.status(200).send({ commit: c2 }))
```

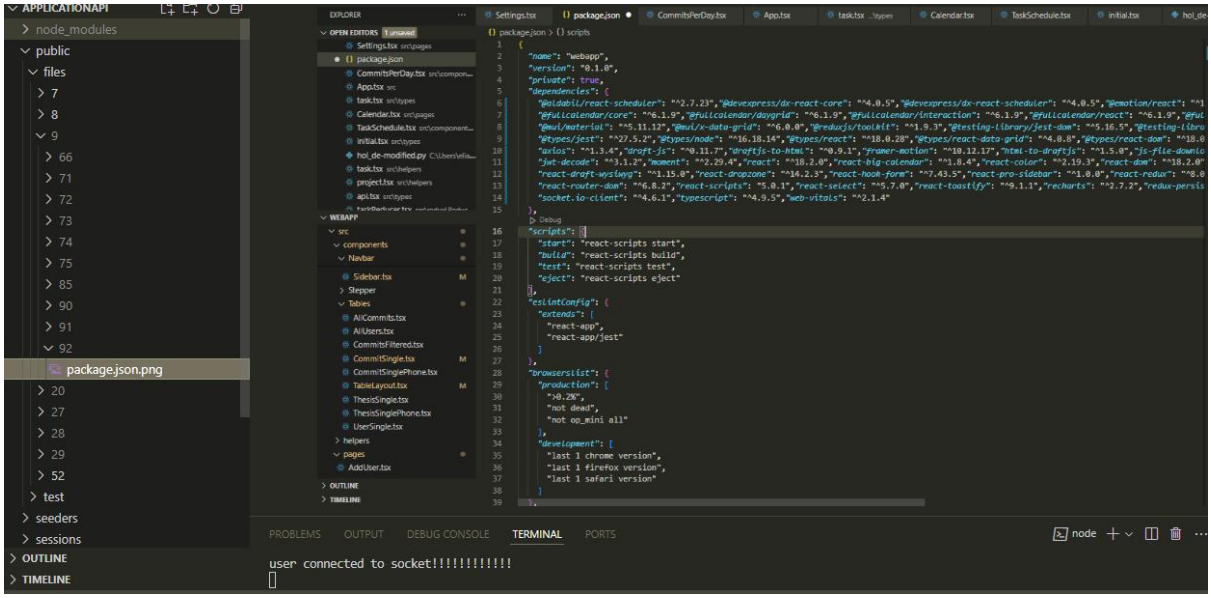
Εικόνα 107: Προσκόλληση αρχείων στο commit

Μόλις ολοκληρωθεί η διαδικασία επιστρέφεται στον πελάτη η εξής απάντηση για το αίτημα του:

commit: Object { id: 92, title: "file upload", description: "file upload", ... }
id: 92
title: "file upload"
description: "file upload"
createdAt: "2024-07-19T14:27:12.000Z"
updatedAt: "2024-07-19T14:27:12.000Z"
user: Object { id: 18, username: "st2", email: "st2@st.com", ... }
project: Object { id: 9, title: "Δοκιμή", description: "Δοκιμή", ... }
files: [{...}]
0: Object { id: 126, fileName: "package.json.png", filePath: "public\\files\\9\\92", ... }
id: 126
fileName: "package.json.png"
filePath: "public\\files\\9\\92"
type: "png"
createdAt: "2024-07-19T14:27:12.000Z"
updatedAt: "2024-07-19T14:27:12.000Z"
commitId: 92

Εικόνα 108: Απάντηση αιτήματος

Και τα αρχεία αποθηκεύονται στον public φάκελο στον εξυπηρετητή



Εικόνα 109: Αποθήκευση αρχείων

4.7 Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό έγινε αναλυτική συζήτηση σχετικά με την αρχιτεκτονική του έργου και πως αυτή υλοποιήθηκε. Στην συνέχεια αναλύονται οι λειτουργίες του εξυπηρετητή, καθώς και τα πακέτα που χρειάστηκαν για να πραγματοποιηθούν οι επιθυμητές λειτουργίες. Τέλος, αφού κάναμε το ίδιο και για τον πελάτη, με εκτενέστερη αναφορά στο state management, καταλήξαμε πως λειτουργεί η ζωντανή συνομιλία και από τις δύο πλευρές της εφαρμογής.

5. Επίλογος

Στο κεφάλαιο αυτό, γίνεται η σύνοψη όσων αναλυθήκαν στα προηγούμενα κεφάλαια, με σκοπό την κατάληξη σε συμπεράσματα. Επιπλέον, όπως κάθε σύστημα χρήζει βελτιώσεων που πρέπει να αναφερθούν, καθώς και μερικά ποσοστά που καθιστούν την εφαρμογή αυτή μία καινοτομία στην ακαδημαϊκή κοινότητα.

5.1 Σύνοψη εφαρμογής

Στα πλαίσια αυτής της διπλωματικής εργασίας, σχεδιάστηκε μία εφαρμογή με σκοπό την διευκόλυνση των φοιτητών, σε ένα από τα μεγαλύτερα προβλήματα που αντιμετωπίζουν κατά την διάρκεια των σπουδών τους.

Το σύστημα ακολουθώντας την αρχιτεκτονική του εξυπηρετητή-πελάτη, αποτελείται από την εφαρμογη του πελάτη (frontend), την εφαρμογή του εξυπηρετητή (backend) και την βάση δεδομένων. Η καινοτομία της εφαρμογής αυτής είναι η χρήση της συνομιλίας πραγματικού χρόνου, βοηθώντας στην καλύτερη επικοινωνία φοιτητή-καθηγητή για την πρόοδο. Πέρα όμως από αυτή την λειτουργία, η αρχικοποίηση της διπλωματικής καθώς και η καταγραφή της προόδου είναι κάτι το οποίο δεν υπήρχε σε κάποια άλλη παρόμοια εφαρμογή.

Στα τεχνικά κομμάτια της εφαρμογής, η χρήση της Javascript, αποτέλεσε μόνοδρομος, ειδικότερα στον εξυπηρετητή, καθώς χρειαζόμασταν μία τεχνολογία που να «σηκώνει» server και όχι να απαντάει σε αιτήματα μόνο on demand. Επιπλέον, η χρήση της React αποτέλεσε την καλύτερη επιλογή, κυρίως λόγω ταχύτητας και του απλού state management που διαθέτει.

5.2 Αδυναμίες Συστήματος και Μελλοντικές βελτιώσεις

Όπως κάθε σύστημα, έτσι και αυτό αντιμετωπίζει κάποια προβλήματα. Βασικότερο πρόβλημα είναι ότι δεν έχει δοκιμαστεί σε πραγματικές συνθήκες. Αυτό αποτελεί ένα μείζον πρόβλημα, αφού σε πραγματικές συνθήκες, μπορεί ο εξυπηρετητής να μην αντέξει το πιθανό φόρτο εργασίας που θα του ζητηθεί.

Το κυριότερο πρόβλημα είναι αυτό, ωστόσο υπάρχουν και βελτιώσεις που θα αναβαθμίσουν της εφαρμογή καθιστώντας την ικανή να αντέξει όλο το φόρτο εργασία. Μερικές από αυτές είναι:

- Αντικατάσταση του REST API με microservises. Η αλλαγή αυτή θα κάνει πιο ευέλικτο τον κώδικα, αφού κάθε microservice θα είναι υπεύθυνο για μία υλοποιήση, παρέχοντας στον κώδικα μεγαλύτερη ευκολία στην εύρεση σφαλμάτων.
- Dockerize της εφαρμογής, ώστε να τρέχει κάτω από οποιαδήποτε συνθήκη ανεξαρτήτως έκδοσης
- Unit tests, που κάθε φορά που γίνεται αλλαγή στον κώδικα, θα ελέγχουν βασικά κομμάτια του κώδικα για τυχόν σφάλματα που δημιουργήθηκαν.
- Είσοδος με microsoft 365. Αυτή η λειτουργία θα βοηθήσει τους φοιτητές να μην δημιουργήσουν νέους λογαριασμούς, αλλά να έχουν πρόσβαση με τους ακαδημαϊκούς λογαριασμούς τους.
- Λειτουργία τηλεδιάσκεψης.
- Μία τεχνική που θα μπορούσε να επιλύσει το πρόβλημα των πολλών δεδομένων θα ήταν να γίνει normalization των δεδομένων στη redux. Αυτό στην ουσία θα περιόριζε την επανάληψη πληροφορίας.

- Καλύτερη διαχείριση από πλευράς εξυπηρετητή. Δηλαδή ο εξυπηρετητής θα έπρεπε να διαχειρίζεται περισσότερους υπολογισμούς από ότι ο πελάτης

6.Βιβλιογραφία

Team, Facebook Develop. React Documentation. *React.* [Online] https://react.dev/learn.

Kalita, . 2021. Nodejs authentication. *Medium.* [Online] 14 September 2021. https://durlavkalita.medium.com/nodejs-authentication-using-bcryptjs-part-1-a8396c8cb60c.

Joshi, . 2023. Angular vs React vs Vue. *browserstack.* [Online] 11 May 2023. https://www.browserstack.com/guide/angular-vs-react-vs-vue.

S, Ravikiran A. 2024. ReactJS Tutorial: A Step-by-Step Guide To Learn React. *simplelearn.* [Online] 19 January 2024. https://www.simplilearn.com/tutorials/reactjs-tutorial.

Dimassi, . 2024. Choosing the Right Web Framework: Laravel vs. Node.js vs. Django. *medium.* [Online] 6 July 2024. https://medium.com/@dimassimehdi58/choosing-the-right-web-framework-laravel-vs-node-js-vs-django-240bdc24a07d.

Socket.IO: How it works, when to use it, and how to get started. *ably.* [Online] 2022 December 5. https://ably.com/topic/socketio.

Ighodaro, . 2023. Understanding Redux: A tutorial with examples . *logrocket.com.* [Online] 11 September 2023. https://blog.logrocket.com/understanding-redux-tutorial-examples/.

Gupta, . 12. What is REST? . *restfulapi.* [Online] 2023 December 12. https://restfulapi.net/.

Piguła, . 2023. Socket.io tutorial: Real-time communication in web development. *tsh.io.* [Online] 27 June 2023. https://tsh.io/blog/socket-io-tutorial-real-time-communication/.

Stevens, . 2022. 7 fundamental UX design principles all designers should know. *uxdesigninstitute.* [Online] 22 June 2022. https://www.uxdesigninstitute.com/blog/ux-design-principles/.

Dedigama, . 2022. How To Use Sequelize with Node.js and MySQL . *digitalocean.* [Online] 20 July 2022. https://www.digitalocean.com/community/tutorials/how-to-use-sequelize-with-node-js-and-mysql.

Krouska, Akrivi, Troussas, Christos, and Virvou, Maria. ‘A Literature Review of Social Networking-Based Learning Systems Using a Novel ISO-based Framework’. 1 Jan. 2019 : 23 – 39.

Troussas, C., Chrysafiadi, K., Virvou, M. (2018). Machine Learning and Fuzzy Logic Techniques for Personalized Tutoring of Foreign Languages. In: Penstein Rosé, C., et al. Artificial Intelligence in Education. AIED 2018. Lecture Notes in Computer Science(), vol 10948. Springer, Cham. https://doi.org/10.1007/978-3-319-93846-2_67

C. Troussas, A. Krouska and M. Virvou, "Integrating an Adjusted Conversational Agent into a Mobile-Assisted Language Learning Application," 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), Boston, MA, USA, 2017, pp. 1153-1157, doi: 10.1109/ICTAI.2017.00176.

C. Troussas, M. Virvou, and K. J. Espinosa, ―Using visualization algorithms for discovering patterns in groups of users for tutoring multiple languages through social networking, J. Netw., vol. 10, no. 12, Jan. 2016.

Kanetaki, Z., Stergiou, C., Bekas, G., Troussas, C., & Sgouropoulou, C. (2022). A Hybrid Machine Learning Model for Grade Prediction in Online Engineering Education. *International Journal of Engineering Pedagogy (iJEP), 12*(3), pp. 4–24. https://doi.org/10.3991/ijep.v12i3.23873

A. Krouska, C. Troussas and M. Virvou, "Social networks as a learning environment: Developed applications and comparative analysis," *2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)*, Larnaca, Cyprus, 2017, pp. 1-6, doi: 10.1109/IISA.2017.8316430.

Krouska, A., Troussas, C., Sgouropoulou, C. (2020). A Personalized Brain-Based Quiz Game for Improving Students' Cognitive Functions. In: Frasson, C., Bamidis, P., Vlamos, P. (eds) Brain Function Assessment in Learning. BFAL 2020. Lecture Notes in Computer Science(), vol 12462. Springer, Cham. https://doi.org/10.1007/978-3-030-60735-7_11

K. Chrysafiadi, C. Troussas and M. Virvou, "A Framework for Creating Automated Online Adaptive Tests Using Multiple-Criteria Decision Analysis," *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Miyazaki, Japan, 2018, pp. 226-231, doi: 10.1109/SMC.2018.00049.

C. Troussas, A. Krouska and M. Virvou, "Evaluation of ensemble-based sentiment classifiers for Twitter data," *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*, Chalkidiki, Greece, 2016, pp. 1-6, doi: 10.1109/IISA.2016.7785380.