ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη διαδικτυακής εφαρμογής ως μέσο επικοινωνίας και ανάδειξης μουσικών ιδεών»

Του φοιτητή
Αλέξανδρου Μελισσά Μπαλζή
Αρ. Μητρώου: 154582

Επιβλέπων
Ευκλείδης Κεραμόπουλος
Αναπληρωτής Καθηγητής

Ημερομηνία 15/1/2021

Ανάπτηξη διαδικτυακής εφαρμογής ως μέσο επικοινωνίας και ανάδειξης μουσικών ιδεών
Κωδικός Δ.Ε. 20208
Αλέξανδρος Μελισσάς Μπαλτζής
Ευκλείδης Κεραμόπουλος
16/9/2020
1/1/2021

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

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Αλέξανδρου Μελισσά Μπαλτζή που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

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

Η εργασία αυτή είναι αφιερωμένη στην οικογένειά μου και σε όλους τους ανθρώπους που, παρ'όλες τις δυσκολίες που προέκυψαν αυτή τη χρονιά, συνεχίζουν να αγωνίζονται.

Πρόλογος

Η επιλογή της παρούσας διπλωματικής εργασίας συνδέεται άμεσα με το ενδιαφέρον και την ενασχόληση που έχουν οι άνθρωποι με τη μουσική. Η μουσική αφορά σκέψεις και συναισθήματα. Η μετάδοση των ιδεών αυτών καθίσταται σημαντική για την εξέλιξη των καλλιτεχνών σε πνευματικό, ψυχικό και μουσικό επίπεδο.

Με την εφαρμογή αυτή, ανεξάρτητα από το αν ο χρήστης ειναι μουσικός ή όχι, δίνει τη δυνατότητα μετάδοσης και ανταλλαγής μουσικών ιδεών καθώς και άμεση επικοινωνία μεταξύ των χρηστών της. Με αυτόν τον τρόπο, επιτυγχάνεται η ανάπτυξη ανθρώπινων σχέσεων, εξέλιξη του μουσικού επιπέδου καθώς και αλληλεπίδραση μεταξύ μουσικών και παραγωγών.

Περίληψη

Η πτυχιακή αυτή ασχολείται με την ανάπτυξη μιας web εφαρμογής μουσικού περιεχομένου. Απευθύνεται κυρίως σε μουσικούς και παραγωγούς μουσικής αλλά και σε κάθε λάτρη της μουσικής ο οποίος θα ήθελε να ανακαλύψει καινούργια ακούσματα μέσω του διαδικτύου. Κύριος σκοπός της είναι η διευκόλυνση της επικοινωνίας και ανάδειξη των χρηστών της σε μουσικό επίπεδο.

Πιο συγκεκριμένα, η εφαρμογή δίνει τη δυνατότητα εγγραφής και σύνδεσης σε αυτή, επεξεργασία προσωπικού προφίλ,ανέβασμα εικόνας προφίλ, ανέβασμα μουσικού κομματιού σε μορφή mp3, επικοινωνία με άλλους χρήστες μέσω προσωπικών μηνυμάτων, καθώς επίσης και τη δυνατότητα να ακούσουν κομμάτια άλλων χρηστών με δυνατότητα σχολιασμού.

«Development of a web application in order that musicians communicate with each other and promote their musical ideas»

«Alexandros Melissas Mpaltzis»

Abstract

This thesis is about developing a web application with musical content. It is not only addressed to musicians and music producers but also, to every music lover who would like to discover new sounds through the internet. Its main purpose is to facilitate communication and promote its users on a musical level.

More specifically, this application gives the user the chance to register and login, edit his personal profile, upload a profile picture, upload a track through a mp3 file, communicate with other users through personal messages, listen to other user's tracks and also, comment on those tracks.

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέπων καθηγητή μου, τον κο. Ευκλείδη Κεραμόπουλο για την άψογη συνεργασία και την εμπιστοσύνη που μου έδειξε εξαρχής.

Περιεχόμενα

Πρόλογος....v
Περίληψη.... vi
Abstract vii
Ευχαριστίες viii
Περιεχόμενα ix
Κατάλογος Σχημάτων xiiii
Συντομογραφίες.... xvii
Κεφάλαιο 1ο: Εισαγωγή 1
1.1 Εισαγωγή 1
1.2 Στόχος πτυχιακής 1
1.3 Δομή πτυχιακής.... 1
1.4 Δυνατότητες εφαρμογής 2
1.4.1 Εγγραφή και σύνδεση.... 2
1.4.2 Ανέβασμα κομματιού.... 3
1.4.3 Επεξεργασία προφίλ.... 3
1.4.4 Playback κομματιού 4
1.4.5 Αναζήτηση και πλοήγηση προφίλ άλλων χρηστών.... 5
1.4.6 Αποστολή μηνύματος.... 6
1.5 Επίλογος 6
Κεφάλαιο 2ο: Βασικές τεχνολογίες....7
2.1 Εισαγωγή 7
2.2 HTML.... 7
2.3 CSS.... 7
2.4 SCSS.... 8
2.4.1 Mixins.... 9
2.4.2 Variables.... 9
2.5 Javascript.... 10
2.6 Vue JS 10
2.6.1 Components.... 10
2.6.2 Directives.... 11
2.6.3 Computed 11
2.6.4 Vuex 11
2.6.5 Vue Router 12

2.7 Axios 12
2.8 Node JS 12
2.9 Express 13
2.10 JSONWebToken 14
2.11 MongoDB 15
2.12 Mongoose 16
2.13 Validator 16
2.14 Bcrypt 17
2.15 Multer 17
2.16 Επίλογος 18
Κεφάλαιο 3ο: Έννοιες και ορισμοί 19
3.1 Εισαγωγή 19
3.2 HTTP 19
3.3 Rest API 20
3.4 Life-cycle hooks 20
3.5 Base64 21
3.6 User Experience 21
3.7 Local storage 22
3.8 JSON 22
3.9 NPM 23
3.10 Fetch API 23
3.11 JWT 23
3.12 HTMLAudioElement 24
3.13 FormData 24
3.14 Επίλογος 25
Κεφάλαιο 4ο: Σχεδιασμός εφαρμογής 25
4.1 Εισαγωγή 25
4.2 Front-End 25
4.2.1 Vue JS 25
4.2.2 SCSS 27
4.3 Back-End 28
4.3.1 Node JS 28
4.3.2 Express 28
4.3.3 MongoDB 29

4.4 Επίλογος 31
Κεφάλαιο 5ο: Υλοποίηση εφαρμογής 33
5.1 Εισαγωγή 33
5.2 Front-End 33
5.2.1 Header 33
5.2.2 Διαδρομές 34
5.2.3 Αυθεντικοποίηση 34
5.2.4 Εγγραφή 35
5.2.5 Σύνδεση 35
5.2.6 Προφίλ 36
5.2.7 Κομμάτι 36
5.2.8 Player 37
5.2.9 BottomPlayer 38
5.2.10 Σχόλιο 39
5.2.11 Σελιδοποίηση 41
5.2.12 Προστατευμένες διαδρομές 42
5.2.13 Ανέβασμα κομματιού 42
5.2.14 Μήνυμα 43
5.2.15 Συζήτηση 44
5.2.16 Αναζήτηση 44
5.2.17 Loading 44
5.2.18 Αρχική 46
5.2.19 Μεταβλητές περιβάλλοντος 47
5.3 Back-End 48
5.3.1 Σχήματα 48
5.3.2 Σχέσεις 48
5.3.3 Σχήμα χρήστη 48
5.3.4 Σχήμα κομματιού 49
5.3.5 Σχήμα σχολίου 50
5.3.6 Σχήμα προφίλ 51
5.3.7 Σχήμα μηνύματος 51
5.3.8 Σχήμα συζήτησης 52
5.3.9 Σχήμα like 53
5.3.10 Αυθεντικοποίηση 54
5.3.11 Διαδρομές 54

5.3.12 Διαδρομή εγγραφής ... 55
5.3.13 Διαδρομή σύνδεσης ... 58
5.3.14 Διαδρομή αποσύνδεσης ... 58
5.3.15 Διαδρομή ανεβάσματος κομματιού ... 59
5.3.16 Διαδρομή κομματιού ... 60
5.3.17 Διαδρομή διαγραφής κομματιού ... 60
5.3.18 Διαδρομή like κομματιού ... 61
5.3.19 Διαδρομή ενημέρωσης προφίλ ... 62
5.3.20 Διαδρομή αποστολής μηνύματος ... 62
5.3.21 Διαδρομή αποστολής κομματιού μέσω μηνύματος ... 64
5.3.22 Σελιδοποίηση ... 64
5.4 Επίλογος ... 65
Κεφάλαιο 6ο: Συμπεράσματα και βελτιώσεις ... 67
Βιβλιογραφία ... 68

Κατάλογος Σχημάτων

Σχήμα 1.1: Σύνδεση 2
Σχήμα 1.2: Εγγραφή 2
Σχήμα 1.3: Ανέβασμα κομματιού 3
Σχήμα 1.4: Επεξεργασία προφίλ 3
Σχήμα 1.5: Player 4
Σχήμα 1.6: BottomPlayer 4
Σχήμα 1.7: Αναζήτηση προφίλ 5
Σχήμα 1.8: Επίσκεψη προφίλ άλλου χρήστη 5
Σχήμα 1.9: Αποστολή μηνύματος 6
Σχήμα 2.1: Κύρια διαφορά σύνταξης SCSS και CSS 9
Σχήμα 2.2: Απεικόνιση λειτουργίας του Vuex 12
Σχήμα 2.3: Εσωτερικό workflow Node JS 13
Σχήμα 2.4: Express Route με Middleware 14
Σχήμα 2.5: Διαδικασία Authentication χρησιμοποιώντας JWT 15
Σχήμα 2.6: Vue JS και Node JS με Mongoose 16
Σχήμα 2.7: Hashed κωδικός 17
Σχήμα 3.1: Λειτουργία HTTP Πρωτοκόλλου 19
Σχήμα 3.2: Περιγραφή λειτουργίας Rest API 20
Σχήμα 3.3: Διαδικασία rendering ενός component 21
Σχήμα 3.4: Local Storage 22
Σχήμα 3.5: Αναπαράσταση μορφής JSON 23
Σχήμα 4.1: Setup για Vue 26
Σχήμα 4.2: vue.config.js 26
Σχήμα 4.3: Setup για SCSS 27
Σχήμα 4.4: Setup για Node JS 28
Σχήμα 4.5: Αρχείο index.js με Express 29
Σχήμα 4.6: Cors Middleware 29
Σχήμα 4.7: MongoDB-Σύνδεση σε localhost 30
Σχήμα 4.8: Σύνδεση στη βάση 30
Σχήμα 5.1: Header.vue 33
Σχήμα 5.2: Routes 34
Σχήμα 5.3: Register action στο store 35
Σχήμα 5.4: Register μέθοδος στο Register.vue 35
Σχήμα 5.5: Μέθοδος openModal 36
Σχήμα 5.6: Μέθοδος likeUnlikeSong 37
Σχήμα 5.7: Like και Unlike Song Actions 37
Σχήμα 5.8: Div με data-source του κομματιού 38
Σχήμα 5.9: Μέθοδος updateBar για την επεξεργασία έντασης του κομματιού 38
Σχήμα 5.10: Getters και state για το παιζόμενο κομμάτι 39
Σχήμα 5.11: Comment.vue 40
Σχήμα 5.12: Μέθοδος για τη δημιουργία σχολίου 40
Σχήμα 5.13: Mutation για τη δημιουργία σχολίου 41
Σχήμα 5.14: Action για τη δημιουργίου σχολίου 41
Σχήμα 5.15: Μέθοδος getPage για το set της τρέχουσας σελίδας 41

Σχήμα 5.16: Protected Routes Middleware 42
Σχήμα 5.17: Μέθοδοι getFile και upload για το ανέβασμα του κομματιού 43
Σχήμα 5.18: Message.vue 43
Σχήμα 5.19: Μέθοδος sendMessage 44
Σχήμα 5.20: Loader.vue 45
Σχήμα 5.21: SCSS για τον κύκλο-loading 45
Σχήμα 5.22: Keyframes για το animation του Loader 46
Σχήμα 5.23: Index.vue 46
Σχήμα 5.24: Δημιουργία prototype για το URL του API. 47
Σχήμα 5.25: Μέθοδοι virtuals για relations 48
Σχήμα 5.26: User Schema 49
Σχήμα 5.27: Song Schema 50
Σχήμα 5.28: Comment Schema 50
Σχήμα 5.29: Profile Schema 51
Σχήμα 5.30: Message Schema 52
Σχήμα 5.31: Conversation Schema 53
Σχήμα 5.32: Authentication Middleware 53
Σχήμα 5.33: Like Schema 54
Σχήμα 5.34: Υπόδειγμα χρήσης routes μέσω Express 55
Σχήμα 5.35: Register Route 56
Σχήμα 5.36: Μέθοδος checkDuplicates 56
Σχήμα 5.37: Password hashing 57
Σχήμα 5.38: Μέθοδος generateAuthJwt 57
Σχήμα 5.39: Μέθοδος για την απόκρυψη κωδικού και jwts 57
Σχήμα 5.40: Login Route 58
Σχήμα 5.41: Μέθοδος checkCredentials 58
Σχήμα 5.42: Logout Route 59
Σχήμα 5.43: Route για το ανέβασμα κομματιού με χρήση της βιβλιοθήκης Multer 59
Σχήμα 5.44: Ρυθμίσεις για τη χρήση του Multer 60
Σχήμα 5.45: Get Song Route 60
Σχήμα 5.46: Delete Song Route 61
Σχήμα 5.47: Like Song Route 61
Σχήμα 5.48: Update Profile Route 63
Σχήμα 5.49: Send Message Route - Περίπτωση 1η 63
Σχήμα 5.50: Send Message Route - Περίπτωση 2η 63
Σχήμα 5.51: Αποστολή μηνύματος με κομμάτι 64
Σχήμα 5.52: Παράδειγμα χρήσης Pagination 64

Κατάλογος Πινάκων

Πίνακας 2.1: Προτεραιότητες κανόνων CSS 16
Πίνακας 2.2: Διαφορές μεταξύ SQL και NoSQL 24

Συντομογραφίες

SPA	Single Page Application
API	Application Programming Interface
JWT	JSON Web Token
UX	User Experience

Κεφάλαιο 1:Εισαγωγή

1.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναφέρουμε το στόχο και τη δομή της πτυχιακής αυτής. Επίσης, θα αναλυθούν οι δυνατότητες της συγκεκριμένης εφαρμογής.

1.2 Στόχος πτυχιακής

Στόχος της πτυχιακής αυτής είναι η δημιουργία μιας web εφαρμογής μουσικού περιεχομένου. Απευθύνεται, όχι μόνο σε χρήστες οι οποίοι ασχολούνται επαγγελματικά ή ερασιτεχνικά με τη μουσική, αλλά και σε χρήστες ενδιαφερόμενους για μουσικά ακούσματα και κοινωνικοποίηση. Πιο συγκεκριμένα, η εφαρμογή αυτή δίνει τη δυνατότητα στους χρήστες να εισέλθουν στον κόσμο της μουσικής κάνοντας εγγραφή, να δημιουργήσουν το προφίλ τους και να επικοινωνήσουν με χρήστες από όλον τον κόσμο. Παρ’ολ’αυτά, οι χρήστες δεν υποχρεούνται να εγγραφούν για να έχουν πρόσβαση στο μουσικό περιεχόμενο της εφαρμογής. Έχουν τη δυνατότητα να πλοηγηθούν στα κομμάτια των καλλιτεχνών ελεύθερα και απεριόριστα.

1.3 Δομή πτυχιακής

Η πτυχιακή αυτή αποτελείται από 6 κεφάλαια. Κάθε κεφάλαιο αποτελεί βήμα για την κατανόηση και ολοκλήρωση της εφαρμογής.

Κεφάλαιο 1: Αποτελεί το κεφάλαιο της εισαγωγής. Σε αυτό το κεφάλαιο αναλύεται ο απώτερος σκοπός, η δομή της πτυχιακής και οι δυνατότητες της εφαρμογής η οποία ολοκληρώθηκε στα πλαίσια αυτής της πτυχιακής.

Κεφάλαιο 2: Σε αυτό το κεφάλαιο αναλύονται όλες οι τεχνολογίες οι οποίες χρησιμοποιούνται για την υλοποίηση της εφαρμογής. Αναλύεται η λειτουργικότητα και ο ρόλος κάθε τεχνολογίας που έχει σε αυτή καθώς επίσης και οι λόγοι για τους οποίους έγινε η επιλογή τους.

Κεφάλαιο 3: Αναφέρονται και αναλύονται ορισμοί οι οποίοι είναι κρίσιμοι για την κατανόηση και την εμπέδωση της υλοποίησης και λειτουργικότητας της εφαρμογής.

Κεφάλαιο 4: Στο συγκεκριμένο κεφάλαιο, αναλύονται τα βήματα με τα οποία γίνεται η προετοιμασία της εφαρμογής πριν την υλοποίησής της. Αναφέρονται τρόποι για την καλύτερη διάταξη φακέλων και την εύρυθμη λειτουργία συγκεκριμένων τεχνολογιών όπως Vue και MongoDB.

Κεφάλαιο 5: Περιλαμβάνει την υλοποίηση της εφαρμογής η οποία χωρίζεται στο front-end και στο back-end. Όπως θα αναφέρουμε και παρακάτω, θεωρείται ότι όταν αναφερόμαστε στο front-end, το back-end θα έχει ολοκληρωθεί και όταν γίνεται αναφορά στο back-end, το front-end θα έχει ολοκληρωθεί αντίστοιχα. Εξηγούνται αναλυτικά όλες οι λειτουργίες (αυθεντικοποίηση, εγγραφή, σύνδεση, διαδρομές, σχήματα, σχέσεις κλπ.) και τα βήματα που χρειάστηκαν για την ολοκλήρωση τους με τη βοήθεια σχημάτων.

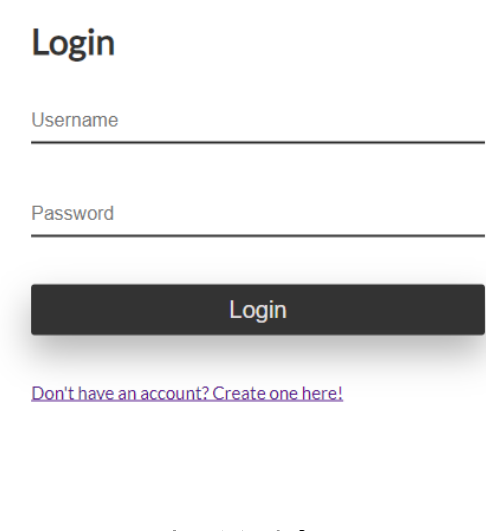
Κεφάλαιο 6: Στο κεφάλαιο αυτό αναφέρονται τα συμπεράσματα και οι προτάσεις βελτίωσης για την πτυχιακή αυτή.

Τέλος αναφέρονται όλοι οι πόροι από τους οποίους αντλήθηκαν πληροφορίες για τη δημιουργία της πτυχιακής αυτής με τη βοήθεια της βιβλιογραφίας.

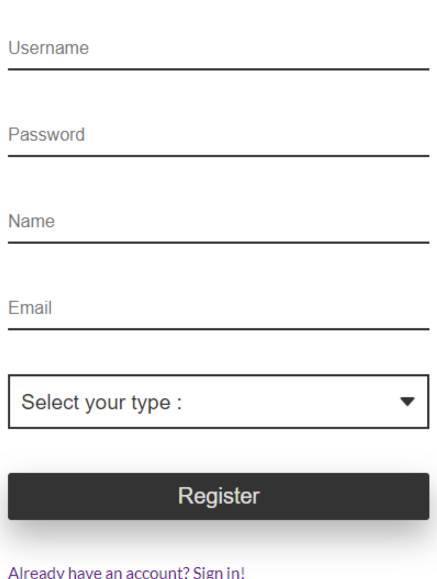
1.4 Δυνατότητες εφαρμογής

1.4.1 Εγγραφή και σύνδεση

Ο χρήστης έχει την δυνατότητα να εγγραφεί (Σχήμα 1.2) και να συνδεθεί (Σχήμα 1.1) στην εφαρμογή έτσι ώστε να ξεκλειδωθούν δυνατότητες όπως ανέβασμα κομματιού, αποστολή μηνύματος, δημιουργία σχολίου και επεξεργασία προφίλ.



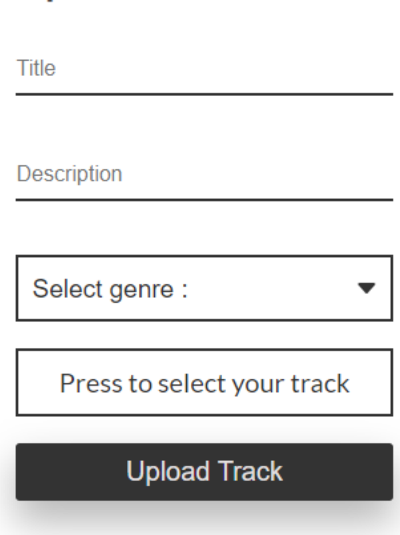
Σχήμα 1.1 Σύνδεση



Σχήμα 1.2 Εγγραφή

1.4.2 Ανέβασμα κομματιού

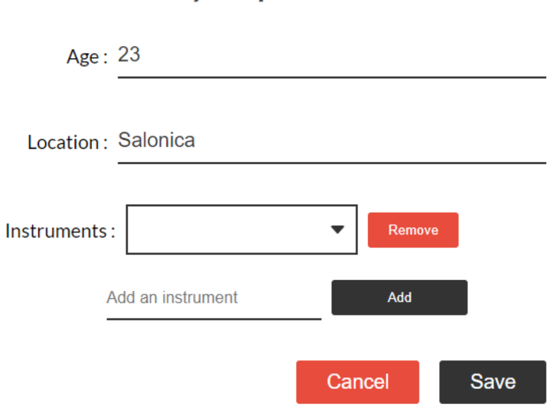
Ο χρήστης, αφού συνδεθεί στο λογαριασμό του, έχει τη δυνατότητα να ανεβάσει το δικό του mp3 κομμάτι συμπληρώνοντας τη φόρμα που φαίνεται στο σχήμα 1.3.



Σχήμα 1.3 Ανέβασμα κομματιού

1.4.3 Επεξεργασία Προφίλ

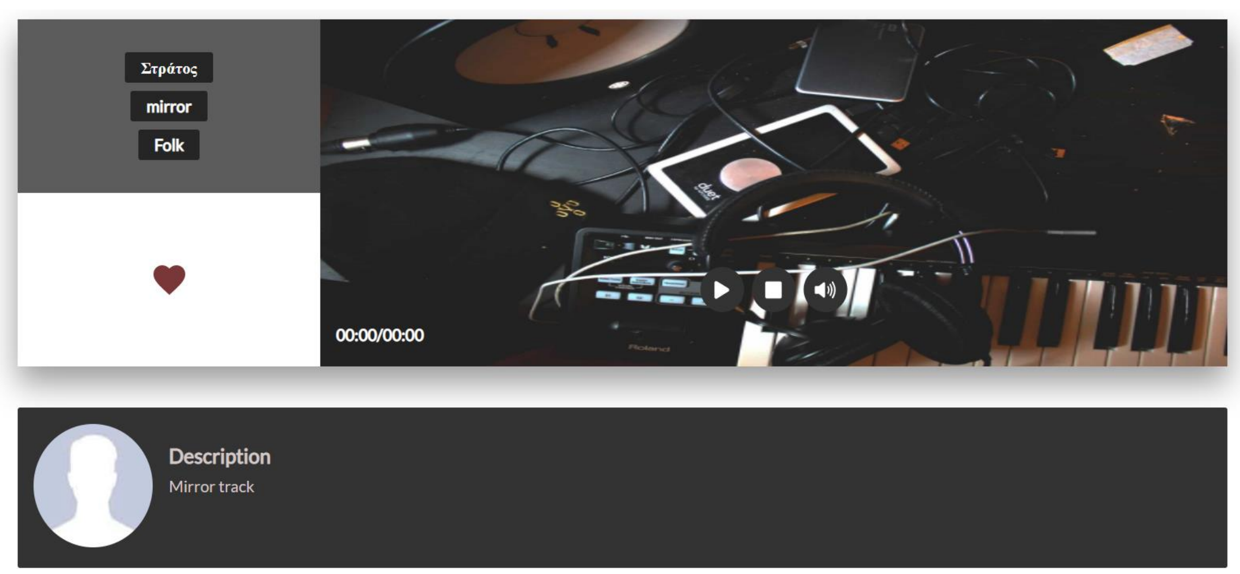
Δίνεται η δυνατότητα στο χρήστη, αφού συνδεθεί, να επεξεργαστεί το προφίλ που θα είναι ορατό στους άλλους χρήστες. Η επεξεργασία προφίλ περιλαμβάνει την αλλαγή φωτογραφίας, ηλικίας, τοποθεσίας και προσθήκη οργάνων (Σχήμα 1.4).



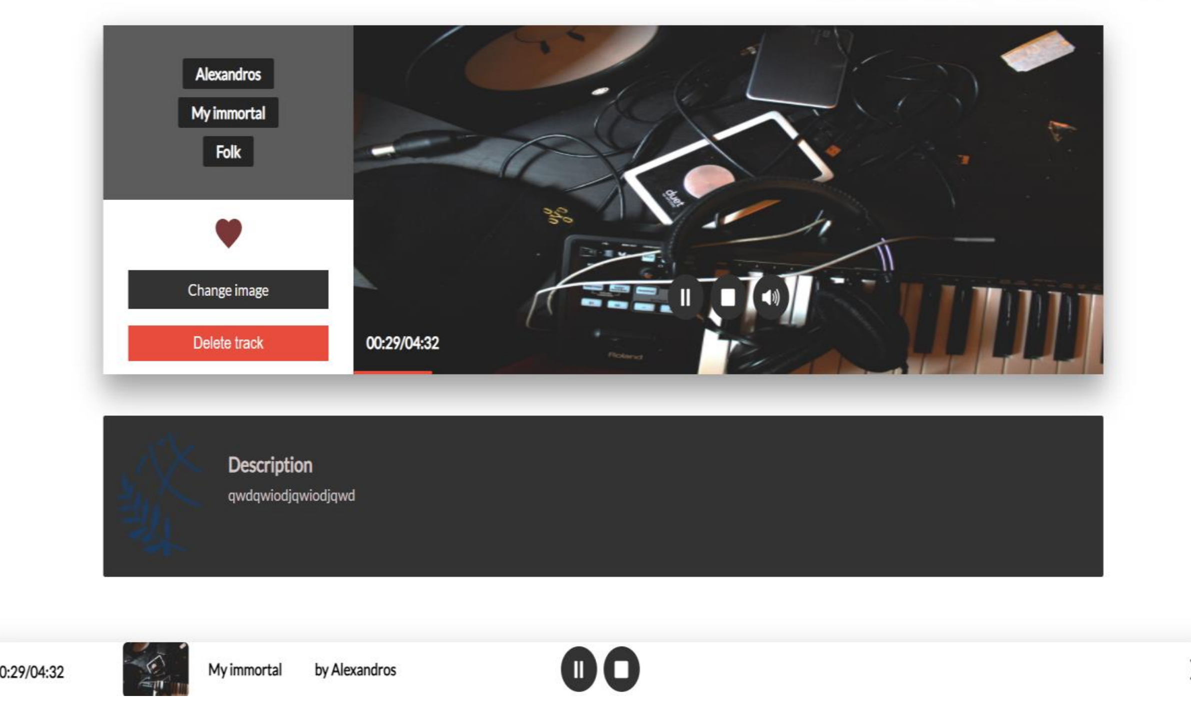
Σχήμα 1.4 Επεξεργασία προφίλ

1.4.4 Playback κομματιού

Ο χρήστης μπορεί να ακούσει κάποιο κομμάτι μέσω του player (Σχήμα 1.5) καθώς και να πλοηγηθεί ακούγοντας το κομμάτι ταυτόχρονα χωρίς να σταματήσει (Σχήμα 1.6).



Σχήμα 1.5 Player

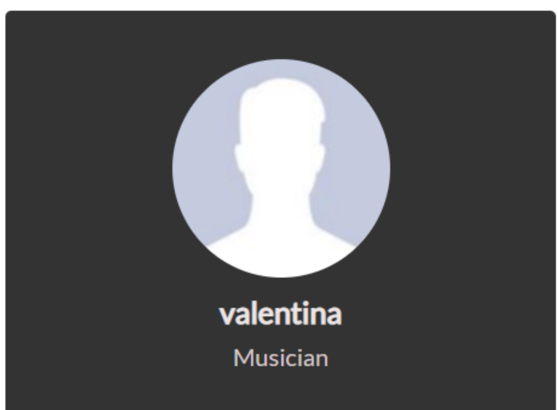


Σχήμα 1.6 BottomPlayer

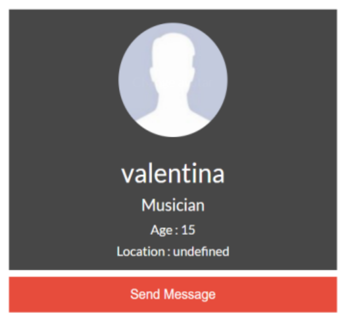
1.4.5 Αναζήτηση και πλοήγηση προφίλ άλλων χρηστών

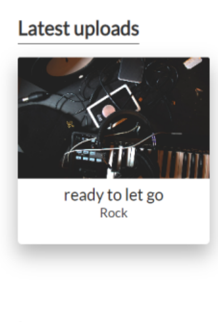
Ο χρήστης έχει τη δυνατότητα να αναζητήσει άλλους χρήστες με το όνομά τους (Σχήμα 1.7) καθώς και να επισκεφθεί το προφίλ τους (Σχήμα 1.8).

Search by name



Σχήμα 1.7 Αναζήτηση προφίλ





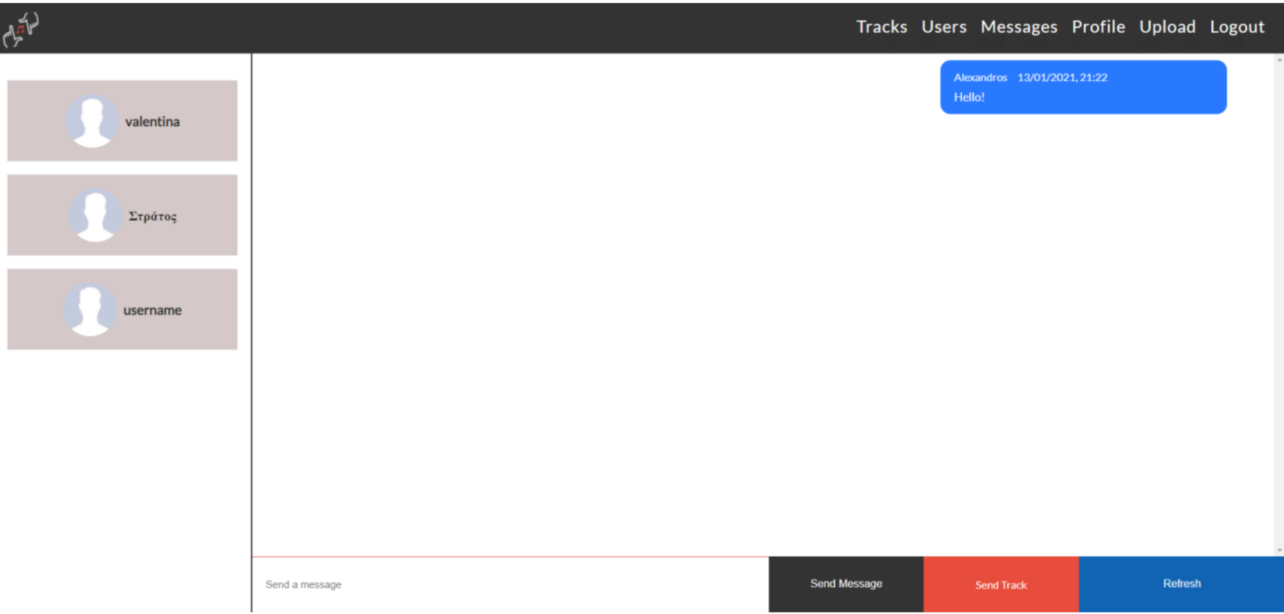
Latest comments

No comments from this user yet

Σχήμα 1.8 Επίσκεψη προφίλ άλλου χρήστη

1.4.6 Αποστολή μηνύματος

Παρέχεται η δυνατότητα αποστολή μηνύματος από χρήστη σε χρήστη. Επίσης, υπάρχει η δυνατότητα αποστολής κομματιού σε μορφή ιδιωτικού μηνύματος (Σχήμα 1.9).



Σχήμα 1.9 Αποστολή μηνύματος

1.5 Επίλογος

Σε αυτό το κεφάλαιο αναλύθηκε κυρίως η δομή και ο στόχος της πτυχιακής. Παράλληλα, έγινε αναφορά των δυνατοτήτων της εφαρμογής με τη βοήθεια των σχημάτων.

Κεφάλαιο 2:Βασικές τεχνολογίες

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναφέρουμε τις βασικές τεχνολογίες που χρησιμοποιούνται στην εφαρμογή.

2.2 HTML

Η HTML είναι η κύρια γλώσσα σήμανσης για τις web εφαρμογές και τις ιστοσελίδες και τα στοιχεία της είναι τα βασικά δομικά στοιχεία των ιστοσελίδων.

Ο σκοπός ενός web browser είναι να διαβάζει τα έγγραφα HTML και να τα συνθέσει σε σελίδες που μπορεί κάποιος να διαβάσει. Στον browser, δεν εμφανίζονται οι ετικέτες HTML αυτούσια αλλά χρησιμοποιούνται για να παρουσιάσουν το περιεχόμενο της σελίδας.

Η HTML5 κυκλοφόρησε για πρώτη φορά στις 22 Ιανουαρίου 2008. Στόχοι της ήταν η βελτίωση της γλώσσας με την υποστήριξη των πιο πρόσφατων πολυμέσων και άλλων νέων λειτουργιών, να γίνει κατανοητή από τον άνθρωπο, τους υπολογιστές και συσκευές όπως προγράμματα περιήγησης στο Web, προγράμματα ανάλυσης κ.λπ. και να παραμείνει συμβατό με παλαιότερο λογισμικό. Η HTML5 συνδυάζει, όχι μόνο την HTML 4 αλλά και τη HTML, XHTML 1 και DOM Level 2.

Βελτιώνει τη σήμανση για έγγραφα και εισάγει διεπαφές σήμανσης και προγραμματισμού εφαρμογών (API) για σύνθετες εφαρμογές ιστού. Για τους παραπάνω λόγους, η HTML5 είναι επίσης κατάλληλη για εφαρμογές πολλαπλών πλατφορμών για κινητά, επειδή διαθέτει δυνατότητες σχεδιασμένες για τις συσκευές χαμηλής ισχύος.

Περιλαμβάνονται πολλές νέες συντακτικές δυνατότητες. Για την εισαγωγή πολυμέσων και γραφικού περιεχομένου, προστέθηκαν τα νέα στοιχεία «video», «audio», «canvas» και «svg». Για τον εμπλουτισμό του σημασιολογικού περιεχομένου των εγγράφων, προστίθενται νέα στοιχεία, όπως «main», «article», «header» και «nav». Παρουσιάστηκαν νέα χαρακτηριστικά, ορισμένα στοιχεία και χαρακτηριστικά αφαιρέθηκαν και άλλα όπως «a» και «menu» άλλαξαν. Το API και το μοντέλο αντικειμένου εγγράφου (DOM) αποτελούν πλέον θεμελιώδη μέρη της HTML5 [1].

2.3 CSS

Η CSS (Cascading Style Sheets) είναι μια γλώσσα που χρησιμοποιείται για τον έλεγχο της εμφάνισης ενός εγγράφου που έχει γραφτεί με μια γλώσσα σήμανσης όπως η HTML.

Η γλώσσα αυτή έχει σχεδιαστεί για να επιτρέπει το διαχωρισμό της παρουσίασης και του περιεχομένου, συμπεριλαμβάνοντας τη διάταξη, τα χρώματα και τις γραμματοσειρές. Αυτός ο διαχωρισμός μπορεί να βελτιώσει την προσβασιμότητα του περιεχομένου, να παρέχει περισσότερη ευελιξία και έλεγχο στην προδιαγραφή των χαρακτηριστικών παρουσίασης, να επιτρέψει σε πολλές ιστοσελίδες να μοιράζονται τη μορφοποίηση καθορίζοντας το σχετικό CSS σε ένα ξεχωριστό αρχείο .css που μειώνει την πολυπλοκότητα και την επανάληψη στο δομικό περιεχόμενο καθώς και το αρχείο .css που θα αποθηκευτεί στην κρυφή μνήμη για να βελτιώσει την ταχύτητα φόρτωσης της σελίδας μεταξύ των σελίδων που μοιράζονται το αρχείο και τη μορφοποίησή του.

Ο διαχωρισμός της μορφοποίησης και του περιεχομένου καθιστά επίσης εφικτή την παρουσίαση της ίδιας σελίδας σήμανσης σε διαφορετικά στυλ για διαφορετικές μεθόδους απόδοσης, όπως στην οθόνη υπολογιστή, οθόνη tablet και οθόνη κινητού [2 - 4].

Το όνομά της προέρχεται από το καθορισμένο σχήμα προτεραιότητας για να προσδιοριστεί ποιος κανόνας στυλ ισχύει εάν πολλοί κανόνες αντιστοιχούν σε ένα συγκεκριμένο στοιχείο (Πίνακας 2.1).

Προτεραιότητα	**Πηγή Κανόνα CSS**	**Περιγραφή**
1	Importance	Το !important annotation
2	Inline	Κανόνας ο οποίος προσδιορίζεται μέσα στο HTML Element
3	Media Query	Ο κανόνας ισχύει για συγκεκριμένες συσκευές
4	Custom κανόνες	Κανόνες οι οποίοι έχουν οριστεί από το χρήστη και περιλαμβάνουν έναν ή πολλούς css κανόνες
5	Selector Specificity	To id (#) υπερισχύει της κλάσης (.)
6	Σειρά κανόνων	Ο τελευταίος κανόνας υπερτερεί σε σχέση με τους κανόνες βρίσκονται από πάνω του
7	Κληρονομικότητα	Αν κάποιο rule δεν έχει οριστεί, τότε κληρονομείται από το γονέα element
8	CSS κανόνες στο HTML αρχείο	Κανόνες οι οποίοι έχουν οριστεί μέσα στο element «style»
9	Browser Default	Κανόνες οι οποίοι ορίζονται by default από τους browser

Πίνακας 2.1 Προτεραιότητες κανόνων CSS

2.4 SCSS

Η SCSS (Sassy CSS) είναι μια ειδική σύνταξη η οποία προέρχεται απο τη SASS. Η SASS είναι μια γλώσσα τύπου preprocessor (αλλαγή στη σύνταξη του CSS) η οποία γίνεται compile πάλι σε CSS. Η σύνταξη του SCSS ειναι παρόμοια με αυτή του CSS. Κύρια διαφορά είναι οτι έχουμε τη δυνατότητα να έχουμε εμφωλευμένο κώδικα (Σχήμα 2.1). Αυτή η τεχνική χρησιμοποιείται για να υπάρχει μια συγκεκριμένη αλληλουχία ανάμεσα στη HTML και στην SCSS. Επίσης, με αυτόν τον τρόπο έχουμε ένα πιο scalable και ευανάγνωστο κώδικα [5].

Η SASS, ουσιαστικά, επεκτείνει τη CSS παρέχοντας διάφορους μηχανισμούς οι οποίοι είναι διαθέσιμοι σε πιο παραδοσιακές γλώσσες προγραμματισμού, ιδιαίτερα σε αντικειμενοστρεφείς γλώσσες, αλλά δεν είναι διαθέσιμοι στη CSS. Όταν γίνεται compile, δημιουργείται ένα CSS αρχείο με κανόνες CSS.

Τα αρχεία που χρησιμοποιούν SCSS λαμβάνουν τις επεκτάσεις .scss.

SCSS

```scss
section {
    height: 100px;
    width: 100px;

    .class-one {
        height: 50px;
        width: 50px;

        .button {
            color: #074e68;
        }
    }
}
```

CSS

```css
section {
    height: 100px;
    width: 100px;
}

section .class-one {
    height: 50px;
    width: 50px;
}

section .class-one .button {
    color: #074e68;
}
```

Σχήμα 2.1 Κύρια διαφορά σύνταξης SCSS και CSS

2.4.1 Mixins

Τα mixins είναι μια λειτουργία την οποία μας προσφέρει η γλώσσα της SASS. Με τα mixins δηλωνουμε διάφορα css properties τα οποία μπορούν να επαναχρησιμοποιηθούν στον κώδικά μας. Διευκολύνουν την αποφυγή χρήσης μη σημασιολογικών τάξεων και τη διανομή συλλογών στυλ σε βιβλιοθήκες. Για να το χρησιμοποιήσουμε, γράφουμε «@include» ακολουθώντας με το όνομα του mixin.

Το όνομα mixin μπορεί να είναι οποιοδήποτε αναγνωριστικό Sass και μπορεί να περιέχει οποιαδήποτε δήλωση εκτός από τις δηλώσεις που χρησιμοποιούνται από την ίδια τη γλώσσα. Μπορούν να χρησιμοποιηθούν για να ενσωματώσουν στυλ που μπορούν να τοποθετηθούν σε έναν κανόνα. Μπορούν να περιέχουν δικούς τους κανόνες που μπορούν να τοποθετηθούν σε άλλους κανόνες ή να συμπεριληφθούν στο ανώτερο επίπεδο του φύλλου στυλ ή μπορούν απλώς να χρησιμεύσουν για την τροποποίηση μεταβλητών [6].

2.4.2 Variables

Τα variables είναι μεταβλητές οι οποίες περιέχουν ένα CSS property. Μετά τη δήλωσή τους σε ένα αρχείο, μπορούμε να τις χρησιμοποιήσουμε σε όλα τα αρχεία .scss. Όταν χρειαστεί, κάποια στιγμή, να αλλάξουμε το συγκεκριμένο property, τότε θα το αλλάξουμε από το δηλωμένο variable με αποτέλεσμα να εφαρμοστεί σε όλα τα σημεία στα οποία έχουμε δηλώσει το συγκεκριμένο variable [7]. Παρά την απλότητά τους, είναι ένα από τα πιο χρήσιμα εργαλεία που φέρνει στο τραπέζι η Sass. Οι μεταβλητές καθιστούν δυνατή τη μείωση της επανάληψης, την πραγματοποίηση σύνθετων μαθηματικών, τη διαμόρφωση βιβλιοθηκών και πολλά άλλα.

Πιο συγκεκριμένα, εκχωρούμε έναν κανόνα σε ένα όνομα που ξεκινά με $ και, στη συνέχεια, μπορούμε να αναφερθούμε σε αυτό το συγκεκριμένο όνομα αντί για την τιμή που εισχωρήσαμε. Οι μεταβλητές μπορούν να δηλωθούν οπουδήποτε θέλουμε.

2.5 Javascript

Η JavaScript είναι μια από τις κύριες τεχνολογίες του World Wide Web. Αποτελεί μέρος της υλοποίησης των browsers, ώστε τα σενάρια από την πλευρά του πελάτη (client-side scripts) να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του εγγράφου που εμφανίζεται [8], [9].

Οι μηχανές JavaScript χρησιμοποιήθηκαν αρχικά μόνο σε προγράμματα περιήγησης ιστού, αλλά τώρα ενσωματώνονται σε ορισμένους διακομιστές, συνήθως μέσω του Node.js. Είναι επίσης ενσωματωμένα σε μια ποικιλία εφαρμογών που έχουν δημιουργηθεί με πλαίσια όπως το Electron και το Cordova.

Αποτελείται από λειτουργικό και επιτακτικό προγραμματισμό που βασίζεται σε events. Διαθέτει διεπαφές προγραμματισμού εφαρμογών (API) για εργασία με κείμενο, ημερομηνίες, κανονικές εκφράσεις, τυπικές δομές δεδομένων και το μοντέλο αντικειμένου εγγράφου (DOM). Ωστόσο, η ίδια η γλώσσα δεν περιλαμβάνει είσοδο / έξοδο (I / O), όπως εγκαταστάσεις δικτύωσης, αποθήκευσης ή γραφικών, καθώς ο browser παρέχει αυτά τα API [9].

Η Javascript τα τελευταία χρόνια έχει εξελλιχθεί ραγδαία προσθέτοντας ολοένα και περισσότερες λειτουργίες όπως αντικειμενοστρέφεια που σχετίζεται με άλλες γλώσσες προγραμματισμού, promises, async/await κλπ.

Παρόλο που υπάρχουν ομοιότητες μεταξύ JavaScript και Java, συμπεριλαμβανομένου του ονόματος γλώσσας, της σύνταξης και των αντίστοιχων τυπικών βιβλιοθηκών, οι δύο γλώσσες είναι διακριτές και διαφέρουν σε μεγάλο βαθμό στο σχεδιασμό.

2.6 Vue JS

Το Vue Js είναι ένα front-end javascript framework το οποίο μας βοηθάει να αναπτύσσουμε scalable εφαμοργές τις οποίες θα μπορούμε να διευρύνουμε με ευκολία. Μέσω της Vue, μπορούμε να φτιάξουμε Single Page Applications, web εφαμοργές οι οποίες δεν χρειάζεται να γίνει ανανέωση στον browser καθώς η Javascript ειναι υπεύθυνη για το re-rendering της σελίδας [10], [11].

Διαθέτει μια σταδιακά προσαρμόσιμη αρχιτεκτονική που εστιάζει στη απόδοση και τη σύνθεση των components. Η βασική βιβλιοθήκη εστιάζεται μόνο στο οπτικό επίπεδο. Προηγμένες δυνατότητες που απαιτούνται για σύνθετες εφαρμογές, όπως routing, state management και εργαλεία κατασκευής προσφέρονται μέσω επίσημων βιβλιοθηκών και πακέτων.

2.6.1 Components

Τα components αποτελούν ένα σημαντικό κομμάτι της Vue. Είναι μικρά μεμονωμένα κομμάτια κώδικα τα οποία αποτελούνται από 3 μέρη. Το template, το script και το style. Το template είναι η HTML που χρησιμοποιείται στο component, το script είναι η Javascript και το style είναι το CSS. Με τη βοήθεια των components, η εφαρμογή μας σπάει σε μικρά κομμάτια κώδικα τα οποία μπορούμε να επαναχρησιμοποιήσουμε σε όλη την εφαρμογή μας [12].

2.6.2 Directives

Τα directives έχουν την μορφή των HTML attributes και είναι ειδικά tags της Vue τα οποία όταν μπαίνουν μεσα σε HTML elements, έχουν κάποια επίδραση πάνω σε αυτά [13].

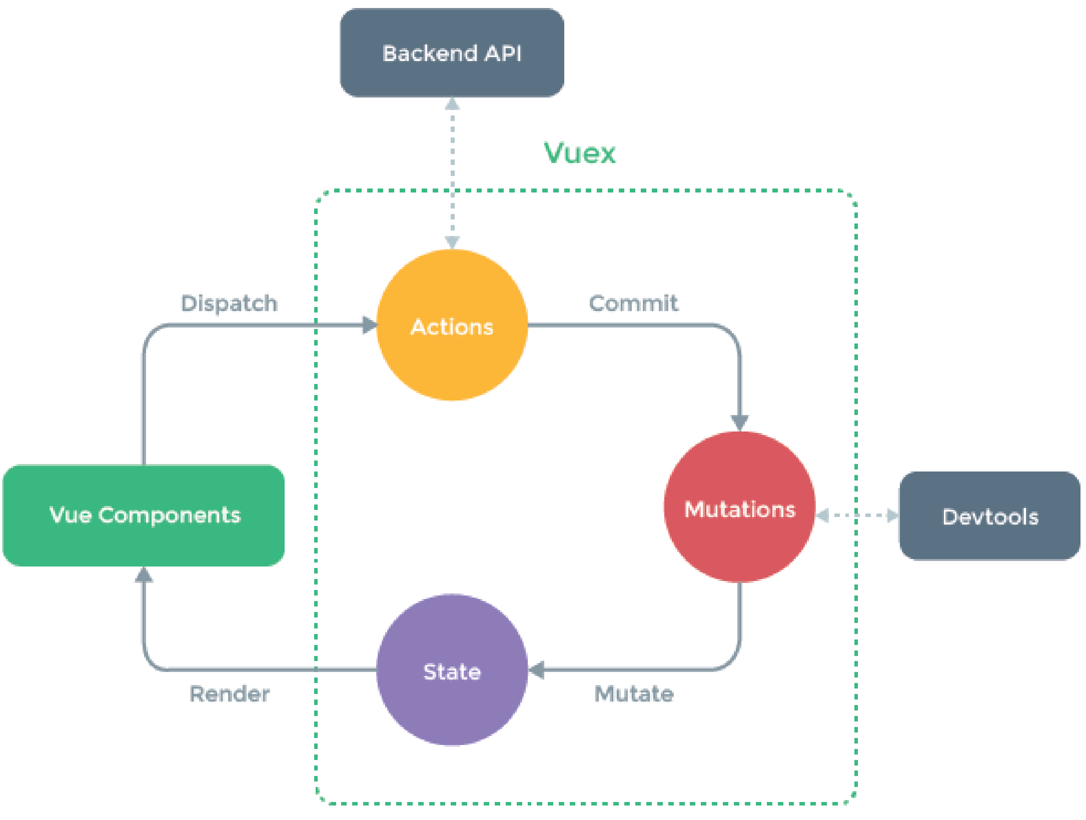
2.6.3 Computed

Οι computed μέθοδοι είναι ειδικές μέθοδοι της Vue οι οποίες υπολογίζουν μια μεταβλητή. Εάν αυτή η μεταβλητή αλλάξει οπουδήποτε, τότε και η computed μέθοδος θα επιστρέψει το ανανεωμένο value της μεταβλητής χρησιμοποιώντας το reactivity της Vue. Ουσιαστικά, στη συγκεκριμένη εφαρμογή, συνήθως μας βοηθάνε να εντοπίσουμε τις αλλαγές που γίνονται στο store με σκοπό την αυτόματη ανανέωση των δεδομένων του store που χρησιμοποιούνται στο component.

2.6.4 Vuex

Το Vuex είναι μια native βιβλιοθήκη της Vue με τη βοήθεια της οποίας έχουμε τη δυνατότητα να έχουμε ενα κεντρικό «store» για όλα τα components έτσι ώστε ένα component να μπορεί να μοιράζεται δεδομένα με ένα άλλο και να επεξεργάζεται κοινές μεταβλητές που υπάρχουν στο store. Γενικά, χρησιμοποιείται με σκοπό μια εφαρμογή να γίνει μεγάλη και επεκτάσιμη [14].

Πιο συγκεκριμένα αποτελείται από το state, actions και mutations. Ένα component καλεί ένα action το οποίο εκτελεί κάποιο HTTP request προς τον server. Μόλις πάρει την απάντηση, κάνει ένα mutation το οποίο κάνει mutate δηλαδή αλλάζει το store. Στη συνέχεια, το state κάνει rendering τα δεδομένα στα Vue Components (Σχήμα 2.2).



Σχήμα 2.2 Απεικόνιση λειτουργίας του Vuex

2.6.5 Vue Router

Το Vue-Router είναι αλλή μια native βιβλιοθήκη της Vue. Επειδή η εφαρμογή μας είναι **SPA** τότε τις διαδρομές δεν τα διαχειρίζεται κάποιος server αλλά η ίδια η Vue μέσω του Vue-Router [15].

2.7 Axios

Έιναι ένας **HTTP** Client για τον browser και για Node JS ο οποίος μας βοηθάει να διαχειριζόμαστε καλύτερα την επικοινωνία client-server. Περιλαμβάνει Javascript promises για να επεξεργαζόμαστε δεδομένα ασύγχρονα.

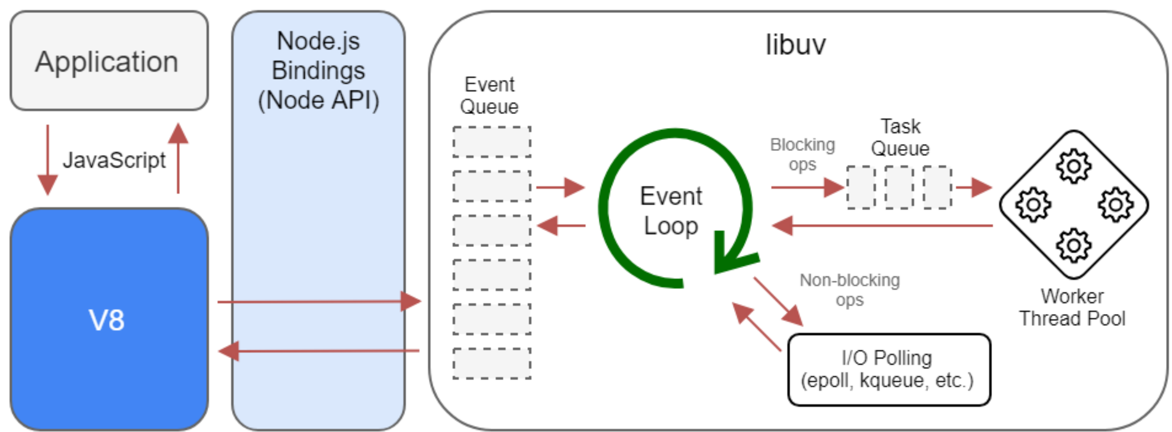
2.8 Node JS

H Node JS είναι ένα open-source, server-side περιβάλλον το οποίο τρέχει έξω από τον browser. Η Node μας επιτρέπει να δημιουργήσουμε web servers/APIs. Περιλαμβάνει built-in modules για I / O συστήματος αρχείων, δικτύωση (DNS, HTTP, TCP, TLS / SSL ή UDP), δυαδικά δεδομένα (buffer), συναρτήσεις κρυπτογραφίας, ροές δεδομένων και άλλες βασικές λειτουργίες. Είναι γνωστή για το μεγάλο οικοσύστημά της [16]. Η γλώσσα που χρησιμοποιεί είναι η Javascript και είναι ο κύριος λόγος που την επιλέξαμε για την εφαρμογή μας.

Όπως βλέπουμε στο σχήμα 2.3, η Node JS λειτουργεί σε single-threaded event loop, χρησιμοποιώντας κλήσεις εισόδου/εξόδου χωρίς αποκλεισμό, επιτρέποντάς του να υποστηρίζει δεκάδες χιλιάδες ταυτόχρονες συνδέσεις χωρίς να επιβαρύνεται με το κόστος της αλλαγής περιβάλλοντος του thread. Ο σχεδιασμός της κοινής χρήσης ενός thread μεταξύ όλων των αιτημάτων που χρησιμοποιούν το μοτίβο προορίζεται για τη δημιουργία ταυτόχρονων εφαρμογών, όπου οποιαδήποτε λειτουργία που εκτελεί I / O πρέπει να χρησιμοποιεί μια επανάκληση. Για να φιλοξενήσει το event loop με ένα thread, η Node.js χρησιμοποιεί τη βιβλιοθήκη libuv - η οποία, με τη σειρά της, χρησιμοποιεί μια ομάδα από threads σταθερού μεγέθους που χειρίζεται ορισμένες από τις ασύγχρονες λειτουργίες εισόδου / εξόδου [17].

Ένα μειονέκτημα αυτής της προσέγγισης με ένα thread είναι ότι η Node.js δεν επιτρέπει την απότομη αύξηση του αριθμού των πυρήνων CPU του μηχανήματος που εκτελεί χωρίς τη χρήση πρόσθετης μονάδας, όπως κάποιο cluster ή StrongLoop Process Manager. Ωστόσο, οι προγραμματιστές μπορούν να αυξήσουν τον προεπιλεγμένο αριθμό threads στην βιβλιοθήκη libuv. Το λειτουργικό σύστημα διακομιστή είναι πιθανό να διανείμει αυτά τα threads σε πολλούς πυρήνες [18].

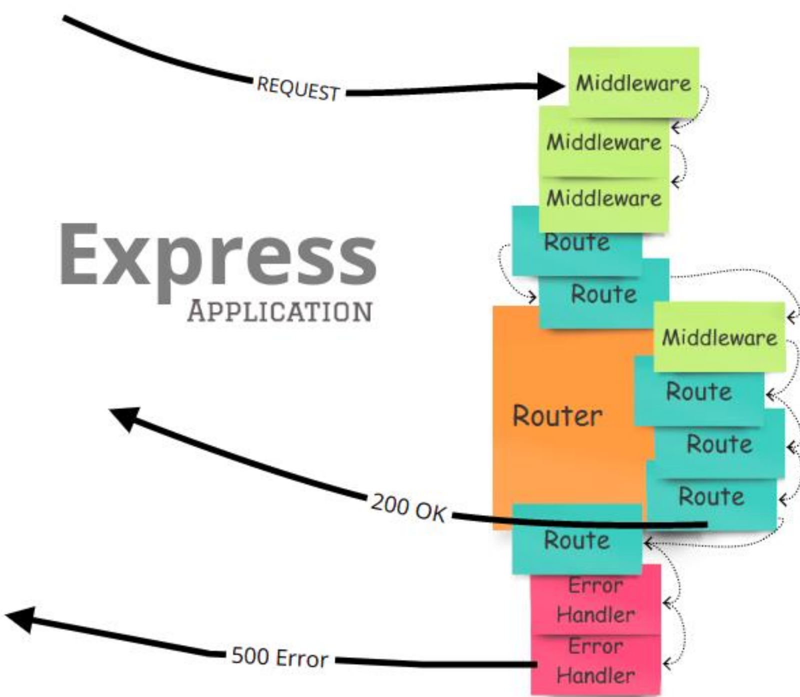
Ένα άλλο πρόβλημα είναι ότι οι υπολογισμοί μεγάλης διάρκειας και άλλες εργασίες που συνδέονται με την CPU παγώνουν ολόκληρο το event loop μέχρι την ολοκλήρωσή τους [19].



Σχήμα 2.3 Εσωτερικό workflow Node JS

2.9 Express

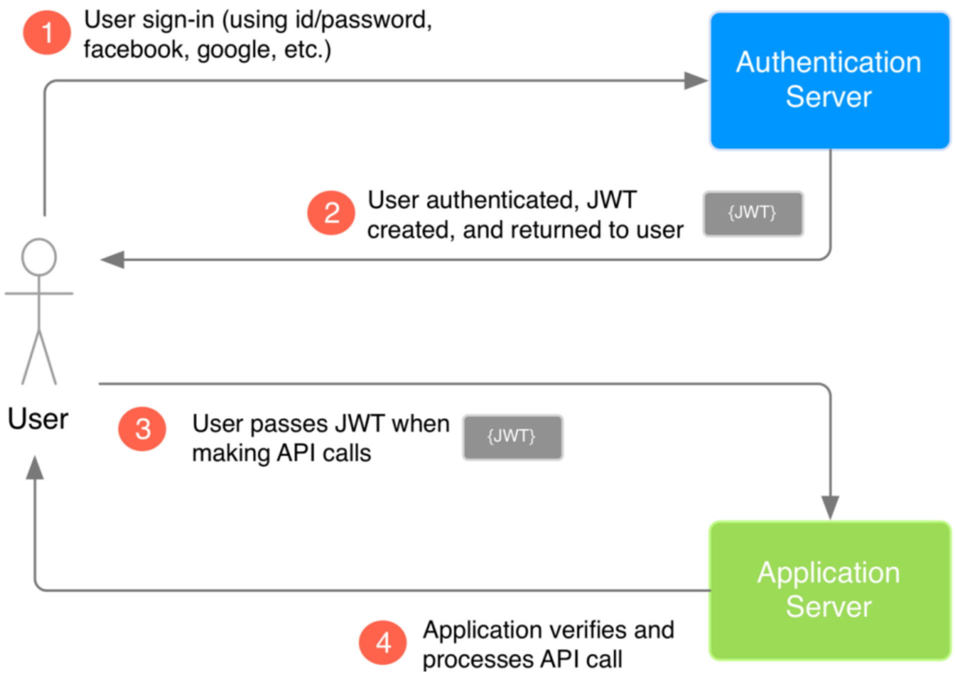
Το Express είναι ένα ευέλικτο web-framework της Node.js που παρέχει ένα μεγάλο σύνολο δυνατοτήτων για την ανάπτυξη εφαρμογών ιστού και κινητών. Διευκολύνει την ταχεία ανάπτυξη εφαρμογών Web της Node JS. Επιτρέπει τη χρήση ειδικών middleware για την καλύτερη και αποδοτικότερη μεταχείριση των HTTP requests (Σχήμα 2.4). Προσφέρει τη δυνατότητα routing για τη δημιουργία routes που εξαρτώνται από τη μέθοδο και το url. Τέλος, δίνει τη δυνατότητα του rendering HTML σελίδων με δυναμικές μεταβλητές.



Σχήμα 1.4 Express Route με Middleware

2.10 JSONWebToken

Η βιβλιοθήκη JSONWebToken μας βοηθάει να υλοποιήσουμε την αυθεντικοποίηση με **JWT.** Σύμφωνα με το σχήμα 2.5, ο χρήστης συνδέεται στο API με τους κωδικούς του. Με τη μέθοδο sign, υπογράφουμε με ένα string (υπογραφή) και δημιουργούμε το jwt. Στέλνουμε το jwt πίσω στον χρήστη και συνήθως ο client τοποθετεί το jwt στο Local Storage. Ο χρήστης εκτελεί μια κλήση API. Τότε χρησιμοποιούμε τη μέθοδο verify για να βεβαιωθούμε ότι το token είναι έγκυρο.



Σχήμα 2.5 Διαδικασία Authentication χρησιμοποιώντας JWT

2.11 MongoDB

Η MongoDB είναι μια cross-platform document-oriented βάση δεδομένων η οποία χαρακτηρίζεται ως NoSQL (Not only SQL) βάση δεδομένων. Χρησιμοποιεί **JSON**-like documents. Η κύρια διαφορά της με τις σχεσιακές βάσεις δεδομένων είναι στην ευκολία διαχείρισης και εφαρμογής αλλαγών στη βάση [20-23].

Οι βάσεις δεδομένων NoSQL παρέχουν έναν μηχανισμό για την αποθήκευση και ανάκτηση δεδομένων που διαμορφώνεται με άλλα μέσα σε σχέση με τα relations που χρησιμοποιούνται σε σχεσιακές βάσεις δεδομένων. Τέτοιες βάσεις δεδομένων υπήρχαν από τα τέλη της δεκαετίας του 1960, αλλά το όνομα "NoSQL" επινοήθηκε μόνο στις αρχές του 21ου αιώνα.

Οι βάσεις δεδομένων NoSQL χρησιμοποιούνται ολοένα και περισσότερο για μεγάλα δεδομένα και εφαρμογές ιστού σε πραγματικό χρόνο. Τα συστήματα NoSQL ονομάζονται επίσης μερικές φορές "Not Only SQL" για να τονίσουν ότι μπορούν να υποστηρίξουν γλώσσες ερωτήσεων που μοιάζουν με SQL.

Τα κίνητρα για αυτήν την προσέγγιση περιλαμβάνουν: απλότητα σχεδιασμού, απλούστερη "οριζόντια" κλιμάκωση σε συστάδες μηχανών (που αποτελεί πρόβλημα για σχεσιακές βάσεις δεδομένων), ακριβέστερος έλεγχος της διαθεσιμότητας και περιορισμός της ασυμφωνίας αντικειμενικής σχέσης αντίστασης. Οι δομές δεδομένων που χρησιμοποιούνται από βάσεις δεδομένων NoSQL (π.χ. ζεύγος κλειδιού-τιμής, ευρεία στήλη, γράφημα ή έγγραφο) είναι διαφορετικές από αυτές που χρησιμοποιούνται από προεπιλογή σε σχεσιακές βάσεις δεδομένων, κάνοντας κάποιες λειτουργίες γρηγορότερες στη βάση NoSQL (Πίνακας 2.2). Η συγκεκριμένη καταλληλότητα μιας δεδομένης βάσης δεδομένων NoSQL εξαρτάται από το πρόβλημα που πρέπει να λύσει. Μερικές φορές οι δομές

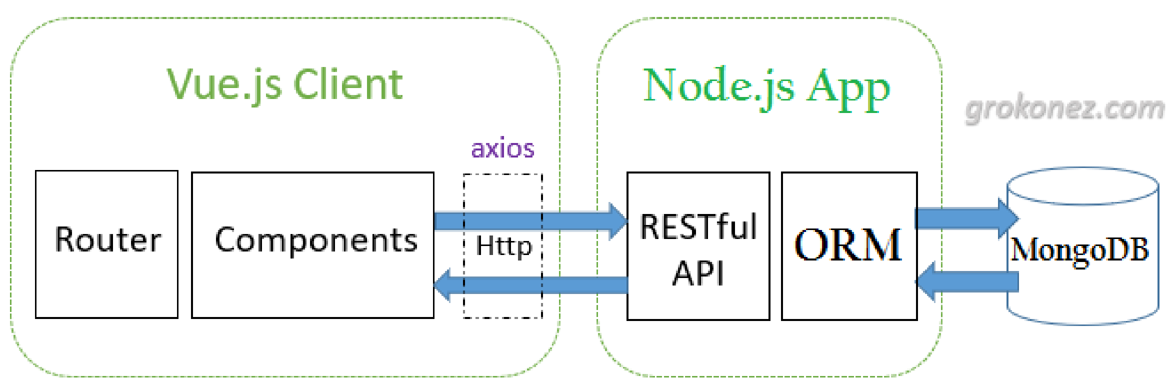
δεδομένων που χρησιμοποιούνται από βάσεις δεδομένων NoSQL θεωρούνται «πιο ευέλικτες» από τους σχετικούς πίνακες βάσεων δεδομένων.

	SQL	NoSQL
Data Model	Πίνακες με σειρές και στήλες	JSON Documents
Κύριος Σκοπός	Γενικός	Πολλά δεδομένα με απλά queries
Schemas	Αυστηρή δομή	Ευέλικτη δομή
Scaling	Οριζόντια (αυξάνεται με μεγαλύτερο server)	Κάθετα (αυξάνεται με πολλούς servers – clusters)
Transactions	Υποστηρίζουν	Οι περισσότερες δεν υποστηρίζουν – κάποιες όμως (MongoDB) υποστηρίζουν transactions
Joins	Είναι απαραίτητα	Συνήθως δεν είναι απαραίτητα
Data to Object Mapping (για κώδικα και γλώσσες προγραμματισμού)	Απαιτείται κάποιο ORM	Συνήθως δεν απαιτείται. Τα documents της MongoDB ταυτίζονται άμεσα με τα πιο γνωστά data structures.

Πίνακας 2.2 Διαφορές μεταξύ SQL και NoSQL

2.12 Mongoose

Η Mongoose είναι μια npm βιβλιοθήκη για το Object Modelling της MongoDB (κουτί ORM στο σχήμα 2.6). Αποτελεί πρώτη επιλογή για την διαχείριση της βάσης στη Javascript.



Σχήμα 2.6 Vue Js και Node JS με Mongoose

2.13 Validator

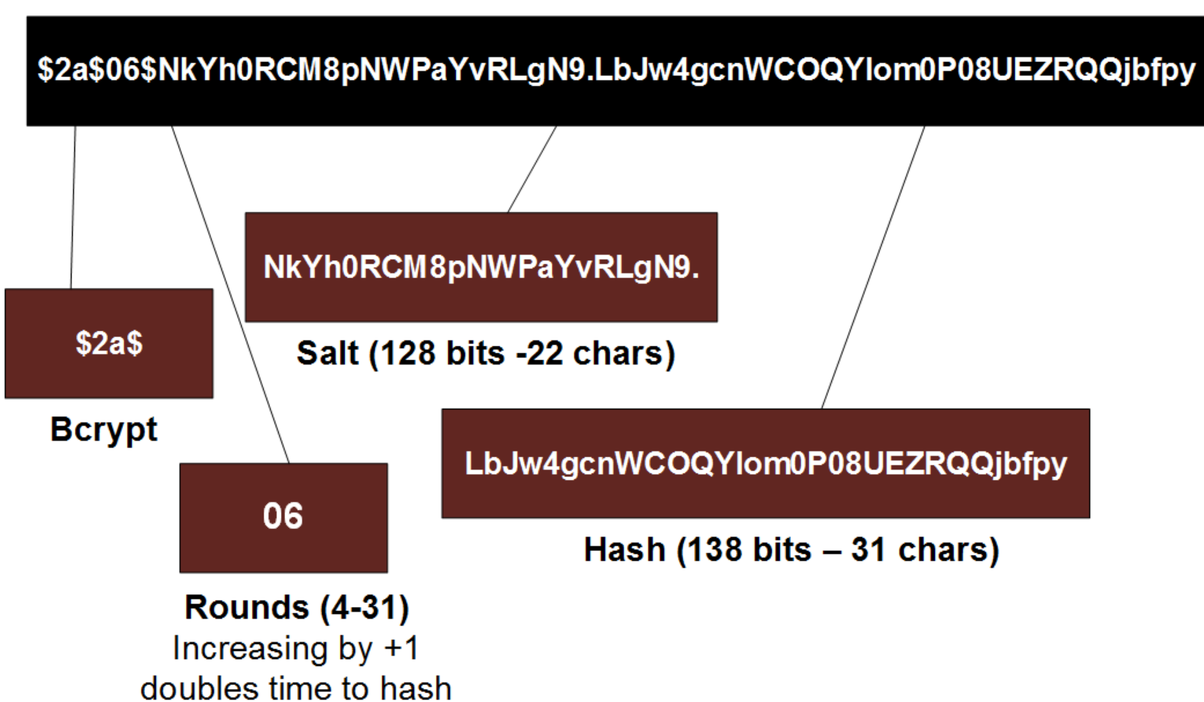
Ο Validator είναι μια βιβλιοθήκη η οποία μας βοηθάει να κάνουμε validate τα δεδομένα που έρχονται και φεύγουν από το back-end δίχως να εξαρτιώμαστε από regular expressions.

2.14 Bcrypt

Η Bcrypt είναι μία npm βιβλιοθήκη η οποία μας βοηθάει να προστατέψουμε τους κωδικούς που αποθηκεύονται στη βάση με hashing μεθόδους. Ουσιαστικά, κρυπτογραφεί και αποκρυπτογραφεί τους κωδικούς όταν τους χρειαζόμαστε για μεγαλύτερη ασφάλεια σε περίπτωση εισβολής στη βάση.

Χρησιμοποιεί τη password-hashing μέθοδο η οποία αναπτύχθηκε από τους Niels Provos και David Mezieres και εφαρμόζει τα λεγόμενα «salts» τα οποία είναι επίπεδα αποκρυπτογράφησης χρησιμοποιώντας τον συγκεκριμένο αλγόριθμο. Όσο πιο περίπλοκο είναι το salt, τόσο πιο αργή είναι η κρυπτογράφηση με το αντάλλαγμα ότι γίνεται πιο ασφαλής.

Το string που δημιουργείται αποτελείται από τη μέθοδο κρυπτογράφησης, τον αριθμό των γύρων (rounds) που χρησιμοποιήθηκαν για την κρυπτογράφηση, το salt και το hash (Σχήμα 2.7).



Σχήμα 2.7 Hashed κωδικός

2.15 Multer

Η Multer είναι μια npm βιβλιοθήκη που αναλαμβάνει φόρμες τύπου multipart/form-data δηλαδή αρχεία, μουσική, βίντεο στο back-end. Μέσω του multer, δημιουργούμε ένα ειδικό middleware το οποίο τοποθετείται αμέσως μετά το request του route μας. Μέσω αυτού παίρνουμε το αρχείο που έχει σταλθεί με τη μορφή buffer και το επεξεργαζόμαστε όπως επιθυμούμε.

2.16 Επίλογος

Στο συγκεκριμένο κεφάλαιο αναφέραμε τις πιο σημαντικές τεχνολογίες της εφαρμογής. Η HTML, CSS, Javascript οι οποίες είναι κύριες τεχνολογίες για τη λειτουργία του Web. Χρησιμοποιούμε SCSS για το καλύτερη σύνταξη και κάποια σημαντικά χαρακτηριστικά που μας προσφέρει.

Για τη χρήση **SPA**, έχουμε τη Vue, ένα Javascript Framework το οποίο έρχεται με το Vue Router και Vuex, packages τα οποία έχουν καθοριστικό ρόλο στη λειτουργία της εφαρμογής. Κλείνοντας, θα χρησιμοποιήσουμε τη Node, ενα πασίγνωστο Runtime περιβάλλον σε Javascript καθώς και MongoDB για την ευκολία και για το γεγονός ότι χρησιμοποιεί JSON Documents.

Κεφάλαιο 3: Έννοιες και ορισμοί

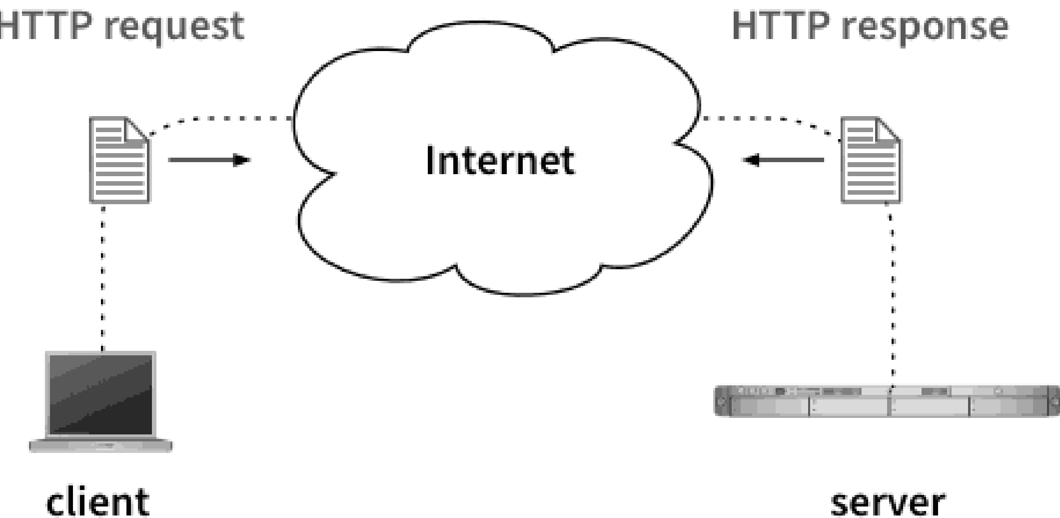
3.1 Εισαγωγή

Σε αυτό το κεφάλαιο αναφέρονται διάφορες έννοιες και ορισμοί οι οποίοι είναι απαραίτητοι για τη κατανόηση της φιλοσοφίας και της λειτουργίας, όχι μόνο της συγκεκριμένης εφαρμογής, αλλά και ολόκληρου του web development. Οι ορισμοί αυτοί αναφέρονται στα επόμενα και προηγούμενα κεφάλαια και είναι μαρκαρισμένοι με bold στην αρχική τους αναφορά.

3.2 HTTP

Το HTTP (Hypertext Transfer Protocol) είναι ένα πρωτόκολλο επιπέδου εφαρμογής το οποίο χρησιμοποιείται για την επικοινωνία δεδομένων στον Παγκόσμιο Ιστό (World Wide Web) [1]. Μέσω αυτού, υπάρχει άμεση επικοινωνία μεταξύ server και client με HTML έγγραφα (σχήμα 3.1).

Τα μηνύματα του πρωτοκόλλου αποτελούνται από το request και το response. Το request, το οποίο στέλνεται από τον client προς τον server, περιλαμβάνει τη μέθοδο (POST, GET, PATCH, DELETE κλπ.) η οποία περιγράφει τον τύπο του request, τους headers (Content-type: application/json) οι οποίοι κρατάνε κάποιες πολυ σημαντικές πληροφορίες και το body περιλαμβάνει τα δεδομένα τα οποία στέλνονται στον server. Το response περιγράφεται από το Status (201,200,400,404), το Status Text (Not Found), τους headers και το body όπου έχει τα δεδομένα της απάντησης [24].



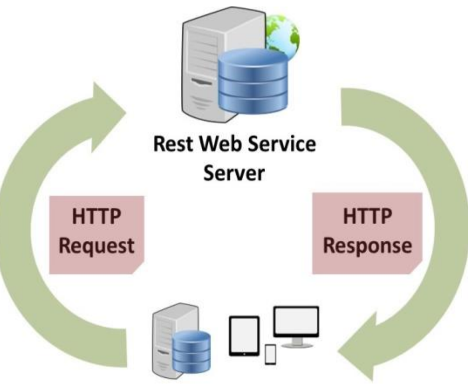
Σχήμα 3.1 Λειτουργία HTTP πρωτοκόλλου

3.3 Rest API

Το Rest API είναι μια μεθοδολογία αρχιτεκτονικής για ένα API (Application Programming Interface) η οποία χρησιμοποιεί **HTTP** requests για να έχει πρόσβαση και να χρησιμοποιήσει δεδομένα [2]. Σε ένα Rest API, αιτήματα που υποβάλλονται σε URI ενός route θα προκαλέσουν μια απάντηση με δεδομένα μορφοποιημένα σε HTML, XML, JSON ή κάποια άλλη μορφή (Σχήμα 3.2). Η απόκριση μπορεί να επιβεβαιώσει ότι έχει γίνει κάποια αλλαγή στην κατάσταση των πόρων και μπορεί να παρέχει συνδέσμους υπερκειμένου με άλλους σχετικούς πόρους. Όταν χρησιμοποιείται HTTP, όπως είναι πιο συνηθισμένο, οι διαθέσιμες λειτουργίες (μέθοδοι HTTP) είναι GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS και TRACE.

Τα «web services» ορίστηκαν για πρώτη φορά στον Παγκόσμιο Ιστό ως έγγραφα ή αρχεία που προσδιορίζονται από τις διευθύνσεις URL τους. Παρ’ολ’αυτά, σήμερα έχουν έναν πολύ πιο γενικό και αφηρημένο ορισμό που περιλαμβάνει κάθε πράγμα, οντότητα ή ενέργεια που μπορεί να αναγνωριστεί, να ονομαστεί, να αντιμετωπιστεί, να αντιμετωπιστεί ή να εκτελεστεί, με οποιονδήποτε τρόπο, στον Ιστό [2].

Χρησιμοποιώντας ένα πρωτόκολλο χωρίς στάνταρ και τυπικές λειτουργίες, τα Rest API στοχεύουν στη γρήγορη απόδοση, την αξιοπιστία και την ικανότητα ανάπτυξης, επαναχρησιμοποιώντας στοιχεία που μπορούν να διαχειριστούν και να ενημερωθούν χωρίς να επηρεάσουν τα σύστηματα στο σύνολό του [2].



Σχήμα 3.2 Περιγραφή λειτουργίας Rest API

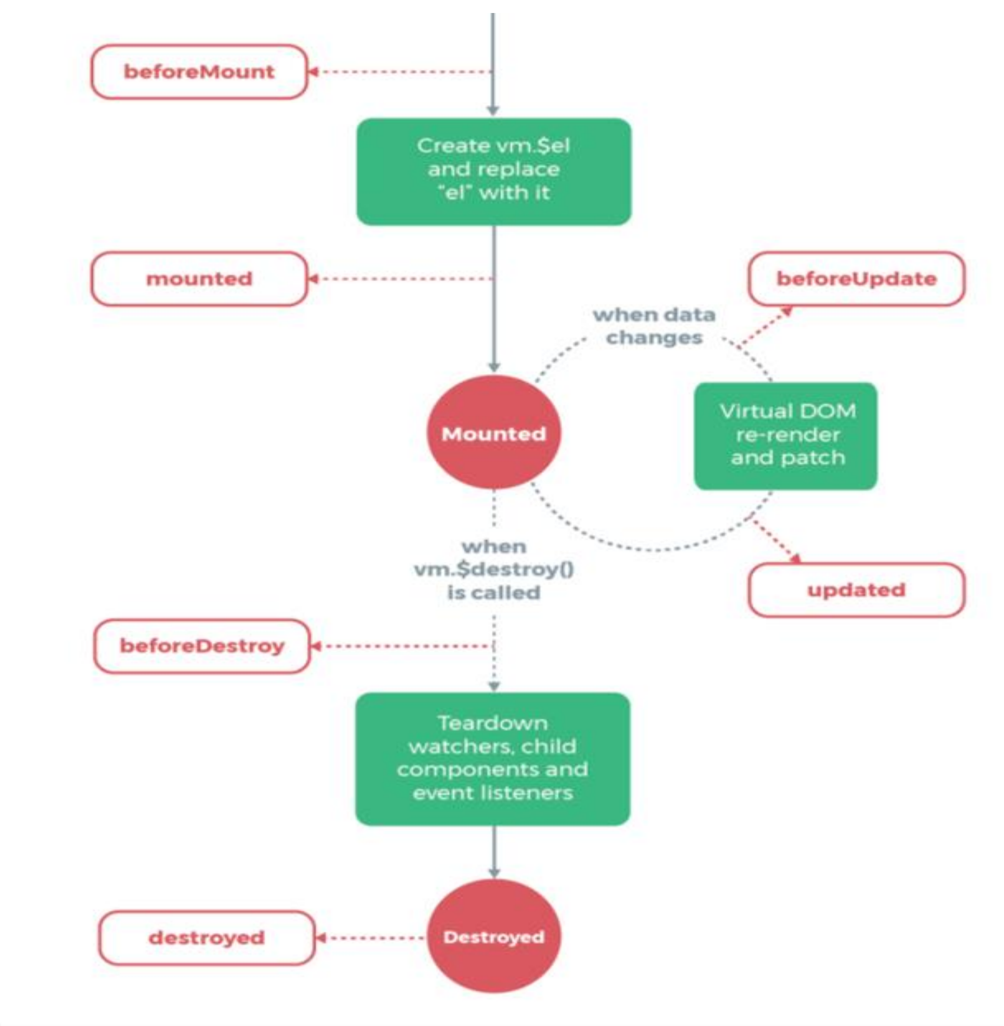
3.4 Life-cycle hooks

Τα life-cycle hooks είναι διάφορα γεγονότα τα οποία συμβαίνουν πίσω από το παρασκήνιο κατά τη διάρκεια του rendering ενός component. Όπως φαίνεται και στο σχήμα 3.3 παρακάτω, καλούνται οι μέθοδοι beforeMount, mounted πριν την εισαγωγή του component στο DOM. Πριν γίνει κάποιο update στο component, καλείται η beforeUpdate και updated, πριν καταστραφεί το component, η beforeDestroy και αφού καταστραφεί η destroyed. Τα life-cycle hooks είναι πολύ χρήσιμα εάν επιθυμούμε να γίνει κάτι πριν,μετά ή κατα τη διάρκεια του rendering του component [26].

3.5 Base64

Το Base64 είναι μια ομάδα σχημάτων κωδικοποίησης δυαδικού σε κείμενο που αντιπροσωπεύουν δυαδικά δεδομένα (πιο συγκεκριμένα μια ακολουθία byte 8-bit) σε μορφή συμβολοσειράς ASCII.

Κάθε μη τελικό ψηφίο Base64 αντιπροσωπεύει ακριβώς 6 bit δεδομένων. Επομένως, τρία ψηφία 8 bit (δηλαδή, συνολικά 24 bit) μπορούν να αναπαρασταθούν με τέσσερα ψηφία Base64 6 bit [27].



Σχήμα 3.3 Διαδικασία rendering ενός component

3.6 User Experience

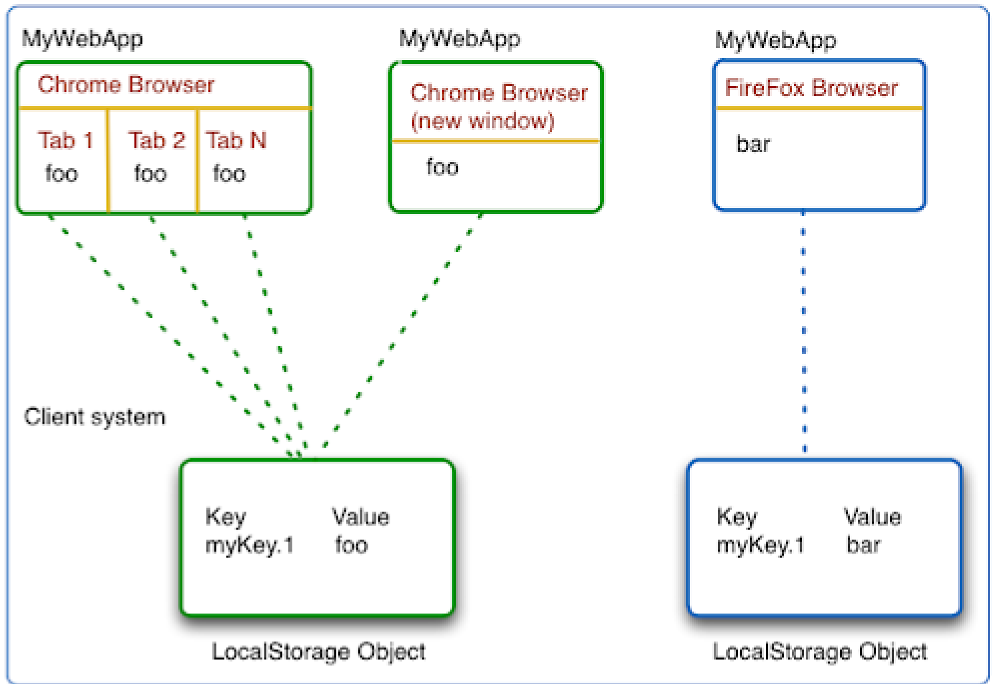
Η εμπειρία χρήστη (UX ή UE) είναι τα συναισθήματα και οι στάσεις ενός ατόμου σχετικά με τη χρήση ενός συγκεκριμένου προϊόντος, συστήματος ή υπηρεσίας. Περιλαμβάνει τις πρακτικές, βιωματικές, συναισθηματικές, σημαντικές και πολύτιμες πτυχές της αλληλεπίδρασης ανθρώπου-υπολογιστή.

Επιπλέον, περιλαμβάνει τις αντιλήψεις ενός ατόμου για πτυχές του συστήματος όπως χρησιμότητα, ευκολία χρήσης και αποτελεσματικότητα. Η εμπειρία του χρήστη μπορεί να έχει υποκειμενικό

χαρακτήρα στο βαθμό που αφορά την ατομική αντίληψη και σκέψη σχετικά με ένα προϊόν ή ένα σύστημα.

3.7 Local storage

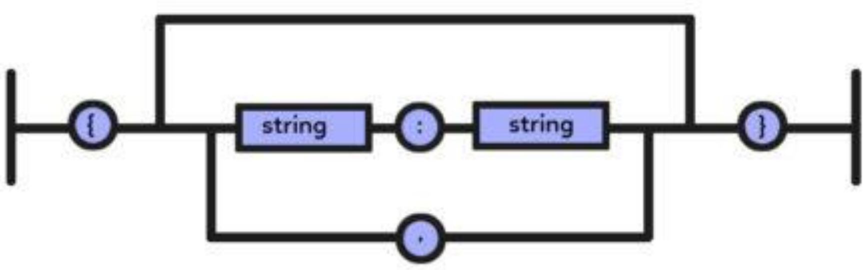
Το Local Storage είναι μια μνήμη η οποία βρίσκεται στον browser η οποία χρησιμοποιείται για να αποθηκεύει διάφορες πληροφορίες (Σχήμα 3.4). Κύριο χαρακτηριστικό της είναι ότι οι πληροφορίες που κρατάει δεν λήγουν και διαγράφονται μόνο όταν είναι επιθυμητό. Επίσης, το local storage δεν επηρεάζεται από κάποιο session με αποτέλεσμα, όταν κλείσει ο browser ή η σελίδα, να μένει αμετάβλητο.



Σχήμα 3.4 Local Storage

3.8 JSON

JavaScript Object Notation ή JSON είναι ένας τύπος δεδομένων ο οποίος χρησιμοποιεί κείμενο για τη μετάδοση αντικειμένων δεδομένων που αποτελούνται από ζεύγη χαρακτηριστικών-τιμών και τύπου δεδομένων συστοιχιών. Πρόκειται για ένα πολύ κοινό τύπο δεδομένων που χρησιμοποιείται για την ασύγχρονη επικοινωνία client-server [27]. Η μορφή του αποτελείται από brackets τα οποία περιέχουν key-value pairs τιμών σε μορφή string οπως φαίνεται και στο σχήμα 3.5.



Σχήμα 3.5 Αναπαράσταση μορφής JSON

3.9 NPM

Ο NPM ή αλλιώς Node Package Manager είναι ο προεπιλεγμένος package manager της Node JS. Αποτελείται από έναν command line client και μια online βάση η οποία περιέχει δημόσια αλλά και ιδιωτικά πακέτα (βιβλιοθήκες). Η πρόσβαση σε αυτά τα πακέτα επιτυγχάνεται μέσω του client και τα διαθέσιμα πακέτα μπορούν να βρεθούν στην επίσημη ιστοσελίδα [28].

3.10 Fetch API

Το Fetch API είναι ένα API που δημιουργήθηκε από τη Mozilla και χρησιμοποιεί το XMLHttpRequest της AJAX. Σκοπός του είναι η ευκολότερη μεταχείριση των web requests και responses ανάμεσα σε client και server [29].

3.11 JWT

Το JWT ή αλλιώς JSON Web Token είναι ένα κρυπτογραφημένο string. Τα διακριτικά υπογράφονται χρησιμοποιώντας ένα ιδιωτικό μυστικό ή ένα δημόσιο-ιδιωτικό κλειδί. Για παράδειγμα, ένας server θα μπορούσε να δημιουργήσει ένα διακριτικό που έχει την αξίωση "συνδεδεμένος ως διαχειριστής" και να το παρέχει σε έναν client. Ο client θα μπορούσε τότε να χρησιμοποιήσει αυτό το διακριτικό για να αποδείξει ότι έχει συνδεθεί ως διαχειριστής.

Τα διακριτικά μπορούν να υπογραφούν από το ιδιωτικό κλειδί ενός μέρους (συνήθως του server), ώστε αυτό το μέρος να μπορεί στη συνέχεια να επαληθεύσει ότι το διακριτικό είναι έγκυρο. Εάν το άλλο μέρος, με κάποια κατάλληλα και αξιόπιστα μέσα, έχει κατοχή του αντίστοιχου δημόσιου κλειδιού, είναι επίσης σε θέση να επαληθεύσουν τη νομιμότητα του διακριτικού. Τα διακριτικά έχουν σχεδιαστεί για να είναι συμπαγή, ασφαλή για URL και μπορούν να χρησιμοποιηθούν ειδικά σε ένα σύστημα σύνδεσης με ένα πρόγραμμα περιήγησης ιστού [30].

3.12 HTMLAudioElement

Το HTMLAudioElement είναι ένα interface το οποίο δίνει πρόσβαση στα properties των Audio HTML elements. Μας δίνει τη δυνατότητα να επεξεργαστούμε αυτά τα properties με διάφορες μεθόδους. Δημιουργούμε ενα HTMLAudioElement αρχικοποιώντας τον constructor του Audio(). Οι πιο χρήσιμες μέθοδοι είναι η play και η pause [31].

3.13 FormData

Η κλάση FormData παρέχει έναν τρόπο για την εύκολη κατασκευή ενός συνόλου ζευγών κλειδιών / τιμών που αντιπροσωπεύουν πεδία φόρμας και τις τιμές τους, τα οποία μπορούν να σταλούν με http requests. Συνήθως χρησιμοποιείται για την αποστολή διάφορων αρχείων με μορφή mp3, wav, jpg, png μαζί με τον header multipart/form-data [32].

3.14 Επίλογος

Ανακεφαλαιώνοντας, σε αυτό το κεφάλαιο, είδαμε κάποιους ορισμούς οι οποίοι αναφέρονται στα επόμενα κεφάλαια όπως JSON, HTTP, Rest API. Η κατανόησή τους είναι κρίσιμη για την εμπέδωση διάφορων λειτουργιών που εφαρμόζονται στην εφαρμογή αυτή.

Κεφάλαιο 4: Σχεδιασμός εφαρμογής

4.1 Εισαγωγή

Λόγω του υψηλού βαθμού πολυπλοκότητας της εφαρμογής, η εφαρμογή χωρίστηκε σε 2 ξεχωριστά και μεμονωμένα κομμάτια κώδικα, στο front-end και στο back-end δίνοντάς μας τη δυνατότητα να διαχωρίσουμε τη λογική του σέρβερ με τη λογική του browser. Με αυτόν τον τρόπο, η εφαρμογή γίνεται πιο επεκτάσιμη λόγω της ευκολίας εφαρμογής αλλαγών στον κώδικα. Πιο αναλυτικά, το front-end δηλαδή ο browser θα «καταναλώνει» το **Rest API** που θα έχουμε φτιάξει στον σέρβερ μας ενεργοποιώντας διάφορα routes που θα έχουμε ορίσει στο back-end. Το Vuex που αναφέραμε πιο πάνω θα παίξει καθοριστικό ρόλο καθώς θα είναι υπεύθυνο για την αποστολή αλλά και την ανάκτηση δεδομένων από το API.

4.2 Front-End

4.2.1 Vue JS

Έχοντας εγκαταστήσει τη Vue, δημιουργούμε ένα καινούργιο boilerplate (έτοιμα αρχεία που μας προσφέρει η Vue) μέσω του command prompt με την εντολή vue create. Έχουν πλέον δημιουργηθεί οι φάκελοι που βλέπουμε στο σχήμα 4.1.

Το αρχείο App.vue θα περιλαμβάνει όλα τα main components που θα χρησιμοποιήσουμε αλλά και κώδικα ο οποίος θα χρησιμοποιηθεί σε όλη την εφαρμογή.

Το αρχείο main.js χρησιμοποιείται για να εκκινήσουμε κάποιες πολυ σημαντικές βιβλιοθήκες όπως το Vuex,Vue-Router κλπ.

Στον φάκελο:

- Views: θα περιέχονται όλα τα .vue αρχεία τα οποία θα είναι τα components που φαίνονται στην εφαρμογή μας.
- Store: θα περιέχονται όλα τα αρχεία που θα έχουν να κάνουν με τον Vuex τα οποία θα συγκροτούν το state μας.
- Router: θα περιέχονται όλα τα αρχεία με routes.
- Components: θα περιέχονται τα components τα οποία τις περισσότερες φορές θα ειναι επαναχρησιμοποιήσημα.
- Assets: στα assets θα συμπεριλαμβάνονται όλες οι εικόνες.

Τον φάκελο public θα τον χρησιμοποιήσουμε όταν θα κάνουμε την Vue εφαρμογή μας build, δηλαδή compile σε Javascript.

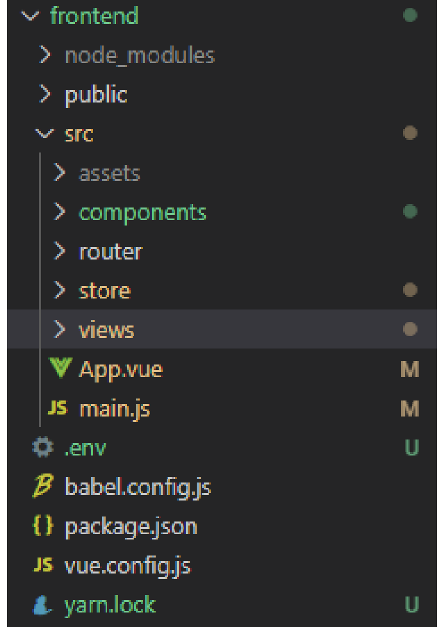
Το αρχείο babel.config.js περίεχει κάποιες ρυθμίσεις για τη μετατροπή της Vue σε κώδικα Javascript.

Το αρχείο vue.config.js περιλαμβάνει ρυθμίσεις της Vue. Στη συγκεκριμένη περίπτωση κάνει load το _main.scss αρχείο (Σχήμα 4.2).

Μέσω των αρχείων package.json και yarn.lock, διαχειριζόμαστε τα modules μας. Περιλαμβάνουν όλα τα modules και dependencies που έχει κατεβάσει το project μας. Αποτελούν μέρος του Package Manager που χρησιμοποιούμε για το project μας (NPM,Yarn).

Το αρχείο .env το δημιουργήσαμε εμείς ετσί ώστε να κρατάμε σημαντικές μεταβλητές όπως το url του API οι οποίες θα χρησιμοποιούνται μέσα στον κώδικα. Με αυτον τρόπο, όταν και αν αλλάξει το url το μόνο που θα κάνουμε είναι να αλλάξουμε τη μεταβλητή σε αυτό το αρχείο με το επιθυμητό μας url.

Τέλος, στον φάκελο node_modules περιέχονται όλα τα packages που χρειάζεται η Vue για να λειτουργήσει καθώς και όλες οι βιβλιοθήκες που προσθέτουμε.



Σχήμα 4.1 Setup για Vue

```
var path = require('path')
module.exports = {
    outputDir : path.resolve(__dirname,'../backend/public'),
    css : {
        loaderOptions : {
            sass : {
                prependData : `@import "@/scss/_main.scss";`
            }

        }
    },

}
```

Σχήμα 4.2 vue.config.js

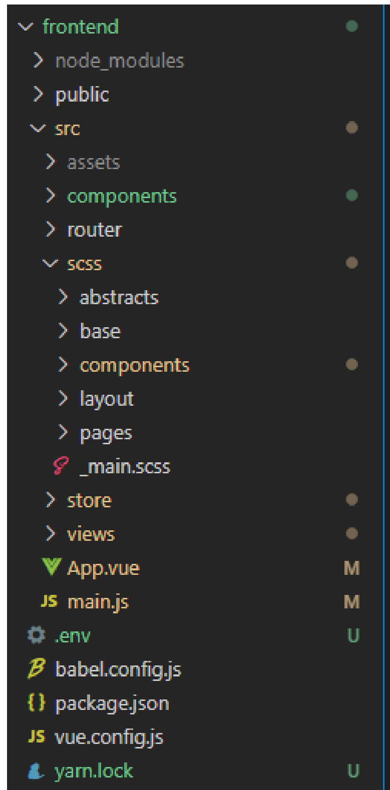
4.2.2 SCSS

Στη συνέχεια δημιουργούμε έναν φάκελο και θα τον ονομάσουμε «scss». Μέσα σε αυτόν τον φάκελο θα έχουμε όλα τα αρχεία του SCSS. Μέσα στον φάκελο που δημιουργήσαμε, δημιουργούμε τους φακέλους abstracts, base, components, layout, pages και το αρχείο main.scss.

Στον φάκελο:

- Abstracts: θα τοποθετήσουμε όλα τα variables και τα mixins.
- Base: θα τοποθετήσουμε τα animations, την τυπογραφία , τα utilities κλπ.
- Components: θα βάλουμε ολα τα αρχεία τα οποία συμβαδίζουν με την λογική των components της Vue.
- Layout: θα συμπεριλάβουμε τα SCSS αρχεία τα οποία θα σχετίζονται με το layout της σελίδας (header, footer, content)
- Pages: θα περιέχονται τα αρχεία SCSS που σχετίζονται με τα views της Vue.

Τέλος, στο αρχείο main.scss θα κάνουμε import μόνο όσα αρχεία θέλουμε να εφαρμόζονται σε όλη την εφαρμογή. Τα υπόλοιπα θα τα κάνουμε import σε vue components ξεχωριστά και μεμονωμένα. Το τελικό αποτέλεσμα φαίνεται στο σχήμα 4.3.



Σχήμα 4.3 Setup για SCSS

4.3 Back-End

4.3.1 Node JS

Έχοντας εγκατεστημένη τη Node , δημιουργούμε έναν φάκελο backend. Μέσα σε αύτον τον φάκελο, δημιουργούμε εναν ακόμα φάκελο και τον ονομάζουμε src. Αυτός ο φάκελος θα περιέχει όλο τον κώδικα του σέρβερ μας. Μέσα στον src, δημιουργούμε τους φακέλους assets, config, db, middleware, models, routers και το αρχείο index.js.

Στον φάκελο :

Routers : περιλαμβάνονται όλα τα routes του API

Models : περιλαμβάνονται όλα τα μοντέλα τα οποία επικοινωνούν με τη βάση

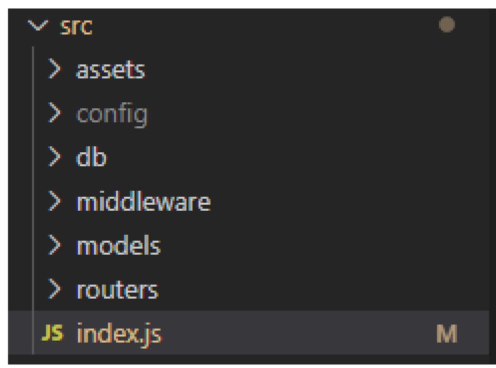
Middleware : περιλαμβάνονται βοηθητικές μέθοδοι οι οποίοι συνήθως χρησιμοποιούνται πριν ή μετα από ενα request ή response. Π.χ. αυθεντικοποίηση

Db : περιέχει τα αρχεία τα οποία απαιτούνται για τη σύνδεσή μας στη βάση

Config : περιέχει όλες τις ρυθμίσεις του server

Assets : περιέχει static περιεχόμενο όπως εικόνες

Το τελικό αποτέλεσμα φαίνεται στο σχήμα 4.4.



Σχήμα 4.4 Setup για Node JS

4.3.2 Express

Για να δημιουργήσουμε το API θα χρειαστούμε μια open-source βιβλιοθήκη της Node η οποία λέγεται Express. Έχοντας εγκαταστήσει την Express, δημιουργούμε το σέρβερ μας στο αρχείο index.js, όπως φαίνεται στο σχήμα 4.6. Απαραίτητη προϋπόθεση για να δουλέψει το API είναι κάθε φόρα που γίνεται ένα request από τον client, ο σέρβερ να κάνει fallback στην index.html της Vue. Αυτό είναι απαραίτητο, γιατί όπως είπαμε, η Vue είναι αυτή που διαχειρίζεται όλα τα routes της σελίδας εφόσον φτιάχνουμε μια **SPA** εφαρμογή. Επίσης, προσθέτουμε ένα ειδικό middleware που λέγεται cors το οποίο διαχειρίζεται όλα τα security errors που προκαλούνται ενώ δουλεύουμε σε localhost (Σχήμα 4.5).

```
const cors = require('cors')
app.use(cors())
app.options('*', cors())
```

Σχήμα 4.5 Cors Middleware

```
const express = require('express')
const path = require('path')


const app = express()


app.use(history({
    disableDotRule: true,
    verbose: true,
    htmlAcceptHeaders: ['text/html', 'application/xhtml+xml']
}))

if(process.env.NODE_ENV === 'production') {
    app.use(express.static(__dirname + '/../public/'))



    app.get('/.*/', (req, res) => {
        res.sendFile(__dirname + '/../public/index.html');
    });

} else {
    app.use(express.static(path.join(__dirname,'../frontend/public')))
    app.get('/.*/', (req, res) => {
        res.sendFile(path.join(__dirname, '../frontend/public/index.html'));
    });


}

const port = process.env.PORT



app.listen(port, () => {
    console.log(`Server is up on : ${port}`)
})
```

Σχήμα 4.6 Αρχείο index.js με Express

4.3.3 MongoDB

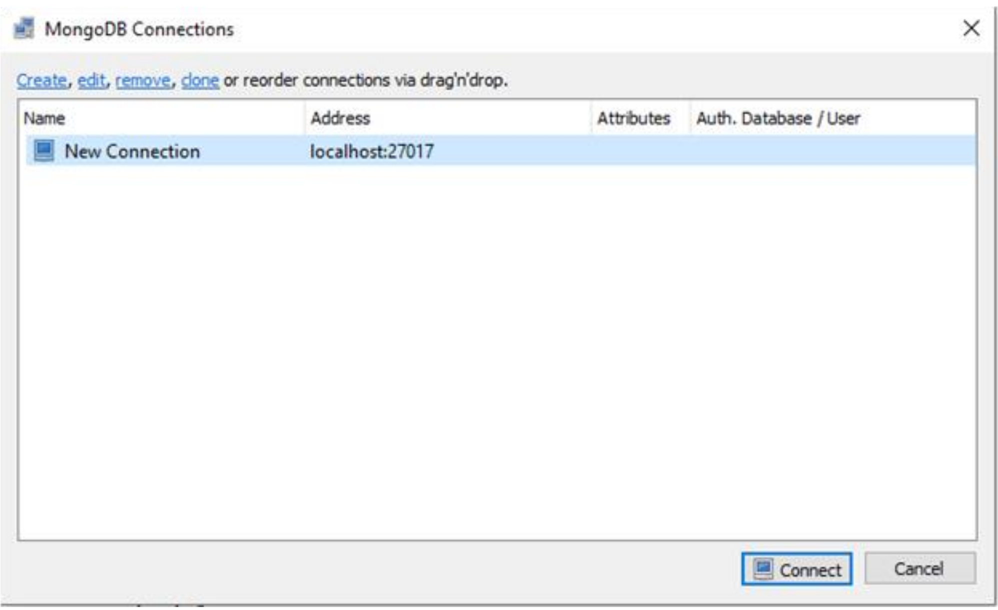
Εγκαθιστούμε την MongoDB και το GUI της (Robo 3T) και φτιάχνουμε μια σύνδεση στον localhost όπως βλέπουμε στο σχήμα 4.7.

Για να συνδέσουμε τη βάση με τη Node, θα χρησιμοποιήσουμε ένα package το οποίο λέγεται Mongoose. Το mongoose είναι το κατάλληλο εργαλείο για να δημιουργήσουμε σχήματα έτσι ώστε να επικοινωνούμε άμεσα με τη βάση.

Αφού το εγκαταστήσουμε δημιουργούμε τη σύνδεση (σχήμα 4.8). Σαν παράμετρο του δίνουμε τη δυναμική environment μεταβλητή την οποία την φτιάχνουμε στον φάκελο config και περιλαμβάνει το url της MongoDB βάσης μαζί με τα ακόλουθα options:

1. useNewUrlParser: συνθέτει το connection string για τη σύνδεση της βάσης
2. useUnifiedTopology: σε περίπτωση αποσύνδεσης από τη βάση, γίνεται προσπάθεια για επανασύνδεση
3. createIndex: για ταχύτερες αναζητήσεις

Ονομάζουμε το αρχείο mongoose.js και το τοποθετούμε μέσα στον φάκελο db.



Σχήμα 4.7 MongoDB-Σύνδεση σε localhost

```
const mongoose = require('mongoose')

mongoose.connect(process.env.MONGODB_URL,{
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex : true
})
```

Σχήμα 4.8 Σύνδεση στη βάση

4.4 Επίλογος

Σε αυτό το κεφάλαιο κάναμε την προετοιμασία για την υλοποίηση της εφαρμογής. Ξεκινήσαμε με το front-end όπου δημιουργήσαμε τους κατάλληλους φακέλους και τα κατάλληλα αρχεία για τη σωστή δομή της εφαρμογής. Έπειτα, εφαρμόσαμε την ίδια τακτική και στο back-end. Τέλος, συνδεθήκαμε στη βάση.

Κεφάλαιο 5: Υλοποίηση εφαρμογής

5.1 Εισαγωγή

Λόγω της δομής της εφαρμογής και της λογικής που χρησιμοποιεί, η υλοποίηση διαχωρίζεται σε δύο κομμάτια, στο front-end και στο back-end. Παρ'όλ'αυτά, όταν αναφερόμαστε σε ενέργειες που γίνονται στο frontend, θα θεωρήσουμε ότι το backend είναι έτοιμο ανεξάρτητα με τη σειρά που αναφέρονται οι παράγραφοι. Όταν δουλεύουμε στο backend, το frontend θα είναι έτοιμο αντίστοιχα.

5.2 Front-End

5.2.1 Header

Δημιουργούμε το αρχείο Header.vue μέσα στον φάκελο Views. Μεσα στα tags του template φτιάχνουμε το Navbar το οποίο αποτελείται από τα links : Tracks, Users, Login, Sign up, Messages, Profile, Upload, Logout. Σε κάθε link, αντί για το element «a», έχουμε το ειδικό element «router-link» το οποίο προέρχεται από τη Vue. Στα router-links, δίνουμε το route που θα δημιουργησουμε στη συνέχεια με το attribute «to». Το συγκεκριμένο attribute είναι παρόμοιο με το attribute «href» της HTML. Τα routes που δίνουμε μεσα σε αυτό το attribute συντονίζονται με το vue-router της Vue. Έτσι, τα links που φτιάξαμε θα μας ανακατευθύνουν σε συγκεκριμένα routes τα οποία θα ορίσουμε στον router. Το τελικό αποτέλεσμα φαίνεται στο σχήμα 5.1.

```
<template>
<div >
        <nav class="navigation" :class="[$route.name==='Index' ? 'navigation_transparent margin-top-small' : 'navigation_dark']">
            <router-link to="/"><img class="navigation_logo" src="../assets/images/logo-light.png" /></router-link>
            <ul class="navigation_list">
                <li class="navigation_item navigation_item-tracks"><router-link to="/tracks" class="navigation_link">Tracks</router-link></li>
                <li class="navigation_item navigation_item-musicians"><router-link to="/users" class="navigation_link">Users</router-link></li>
                <li class="navigation_item"><router-link to="/login" class="navigation_link navigation_link-login">Login</router-link></li>
                <li class="navigation_item"><router-link to="/register" class="navigation_link navigation_link-register">Sign up</router-link></li>
                <li class="navigation_item"><router-link to="/register" class="navigation_link">Sign up</router-link></li>
                <li class="navigation_item"><router-link to="/message" class="navigation_link">Messages</router-link></li>
                <li class="navigation_item"><router-link  class="navigation_link">Profile</router-link></li>
                <li class="navigation_item"><router-link to="/upload" class="navigation_link">Upload</router-link></li>
                <li class="navigation_item"><router-link to="/logout" class="navigation_link">Logout</router-link></li>
            </ul>
        </nav>
</div>
</template>
```

Σχήμα 5.1 Header.vue

5.2.2 Διαδρομές

Στη συνέχεια, κατευθυνόμαστε στο αρχείο index.js στον φάκελο router. Σε αυτόν τον φάκελο, δημιουργούμε τα routes τα οποία είναι ένας πίνακας με objects. Σε κάθε object έχουμε το path,name,component. Το path είναι το url του route, το name είναι το όνομα του route το οποίο θα μας χρησιμεύσει στη συνέχεια και το component είναι αυτό που θέλουμε να ορίσουμε για το συγκεκριμένο route το οποίο το κάνουμε import στο πάνω μέρος του αρχείου. Δημιουργούμε ολα τα routes που χρειαζόμαστε. Ένα δείγμα από διάφορα routes φαίνεται στο σχήμα 5.2.

```
const routes = [
{
  path: '/',
  name: 'Index',
  component: Index
},
{
  path: '/tracks',
  name: 'Home',
  component: Home
},
{
  path: '/register',
  name: 'Register',
  component: Register
},
{
  path: '/login',
  name: 'Login',
  component: Login
},
```

Σχήμα 5.2 Routes

5.2.3 Αυθεντικοποίηση

Δημιουργούμε ένα αρχείο auth.js μέσα στον φάκελο store. Το αρχείο θα είναι ένα από τα πολλά modules του store και όλα μαζί θα συνθέτουν το state μας. Έπειτα το κάνουμε εισαγωγή στο κύριο αρχείο του store και το δηλώνουμε σαν module.

Θεωρώντας ότι το API είναι έτοιμο, φτιάχνουμε 3 actions τα οποία θα είναι το Register, Login, Logout. Κάθε ένα από αυτά τα actions θα περιέχει ένα api call το οποίο θα το εκτελέσουμε με μια βιβλιοθήκη που αναφέραμε, το Axios. Το Axios χρησιμοποιεί εσωτερικά το **Fetch API**. Αυτά τα API calls θα «χτυπούν» στα αντιστοιχα register,login,logout routes του API μας [33].

Όπως παρατηρούμε στο σχήμα 5.3, όταν κάνουμε register, παίρνουμε το **JWT** από το API το οποίο θα το αποθηκεύουμε στο **Local Storage** του browser. Έπειτα, θα βάζουμε αυτόματα στους headers του axios αυτό το token έτσι ώστε να μην χρειάζεται να στέλνουμε το jwt κάθε φορά που κάνουμε ένα request.

Αποθηκεύουμε το jwt και στο state μας το οποίο θα μας βοηθήσει στη συνέχεια όταν θα θέλουμε να ξέρουμε πότε κάποιος χρήστης είναι συνδεδεμένος και πότε όχι (Protected Routes). Δημιουργούμε τον

getter isAuthenticated οποίος επιστρέφει true ή false ανάλογα με το αν υπάρχει το jwt στο state ή όχι. Όταν γίνεται login, η διαδικασία είναι ακριβώς ίδια. Στο logout, διαγράφουμε το jwt από το local storage και από το state.

```
actions: {
    REGISTER : ({commit}, payload) => {

        return Axios.post('/register',payload).then(({data, status}) => {
            if(status===201){
              localStorage.setItem('user-token',data.token)
              Axios.defaults.headers.common['Authorization'] = `Bearer ${data.token}`
              commit('setToken',data.token)
            }

        }).catch((error) => {
          commit('setError',error.response.data.error)
        })
    },
```

Σχήμα 5.3 Register action στο store

5.2.4 Εγγραφή

Δημιουργούμε το αρχείο Register.vue στον φάκελο views. Αυτό το component θα αποτελείται από inputs τα οποία θα είναι το username, password, name, email και το αν ο χρήστης είναι μουσικός ή παραγωγός. Δημιουργούμε την μέθοδο register η οποία καλείται οτάν πατήσουμε το κουμπί Register. Η μέθοδος register δεν κάνει τίποτα άλλο παρά να καλεί το αντίστοιχο action που έχουμε φτιάξει στο store. Εάν η εγγραφή είναι επιτυχής, δηλαδή το status του request είναι 201 τότε ανακατευθύνουμε το χρήστη στο homepage μας (Σχήμα 5.4).

```
methods:{
  register() {
    this.$store.dispatch("REGISTER",this.form).then((success) => {
      Object.keys(this.form).forEach(key => this.form[key] = '')
      this.$router.push('/tracks')
    })
  }
}
```

Σχήμα 5.4 Register μέθοδος στο Register.vue

5.2.5 Σύνδεση

Με παρόμοιο τρόπο φτιάχνουμε τη σελίδα του login. Αυτή τη φορά, τα inputs θα είναι μονο το username και το password για τη σύνδεση του χρήστη. Με την ίδια λογική, φτιάχνουμε τη μέθοδο login η οποία θα καλεί την αντίστοιχη μέθοδο στο store. Αν το response που μας δίνει ο server είναι επιτυχές τότε ανακατευθύνουμε το χρήστη, αλλιώς, δείχνουμε το error.

5.2.6 Προφίλ

Στο προφίλ του χρήστη θα περιλαμβάνεται η φωτογραφία του, πληροφορίες όπως το όνομα του και η ηλικία του, τα κομμάτια που έχει ανεβάσει καθώς και τα σχόλια που έχει κάνει. Επίσης, ο χρήστης θα έχει τη δυνατότητα να αλλάξει την εικόνα προφίλ του και τις πληροφορίες του. Έαν επισκεφτεί ένα προφίλ άλλου χρήστη τότε μπορεί να στείλει μήνυμα.

Όταν φορτώνεται το συγκεκριμένο component, θα καλείται ένα action το οποίο θα παίρνει τα δεδομένα που χρειαζόμαστε για το προφίλ από το API. Το παραπάνω εφαρμόζεται με τη χρήση **Lifecycle Hooks** της Vue. Με αυτόν τον τρόπο, καλώντας με τη μέθοδο mounted το αντίστοιχο action για το προφίλ που έχουμε φτιάξει στο store, παίρνουμε τα δεδομένα του συγκεκριμένου προφίλ και τα παρουσιάζουμε στο template του component. Σημαντική είναι η χρήση του attribute v-if της Vue όπου το rendering ενός HTML element γίνεται μετά από κάποια κατάσταση,στη παρούσα περίπτωση, αφού έχουμε πάρει τα δεδομένα που επιθυμούμε.

Υπάρχουν δύο περιπτώσεις στις οποίες ανοίγει ένας διάλογος (modal). Στην πρώτη περίπτωση, ο χρήστης επεξεργάζεται το προφίλ του. Μόλις πατήσει το κουμπί Edit Profile, καλούμε τη μέθοδο openModal στέλνωντας παραμετρικά το string «edit». Έαν πατήσει το κουμπί Send Message τότε στέλνουμε παραμετρικά το string «send». Στη μέθοδο openModal (σχήμα 5.5), ανάλογα με τη παράμετρο που περάστηκε στη μέθοδο, εμφανίζουμε και τα κατάλληλα inputs.

```
openModal(modal) {
    const modal_container = document.querySelector('.modal_container')
    if(modal == 'send') {
    const modal_box = document.querySelector('.modal_box-send')
    modal_container.style.display = 'flex'
    modal_box.style.display = 'flex'
    }
    const modal_box = document.querySelector('.modal_box-edit')
    modal_container.style.display = 'flex'
    modal_box.style.display = 'flex'
},
```

Σχήμα 5.5 Μέθοδος openModal

5.2.7 Κομμάτι

Το Song component είναι το μεγαλύτερο component της εφαρμογής. Περιλαμβάνει τις πληροφορίες του χρήστη που ανέβασε το κομμάτι, τη δυνατότητα του like/unlike, την αλλαγή φωτογραφίας κομματιού και το διαγραφή του κομματιού (εάν ο συνδεδεμένος χρήστης είναι αυτός που ανέβασε το κομμάτι). Επίσης, περιέχει τον player και τα σχόλια που είναι ξεχωριστά, μικρότερα components.

Στη μέθοδο lifecycle hook created καλούμε το action getSong για να πάρουμε το json που έχει όλες τις πληροφορίες του κομματιού και το action checkLike για να ελέγξουμε αν ο συνδεδεμένος χρήστης έχει κάνει like στο συγκεκριμένο κομμάτι. Μόλις ο χρήστης πατήσει το εικονίδιο της καρδιάς τότε καλούμε τη μέθοδο likeUnlikeSong (σχήμα 5.6) όπου καλεί το κατάλληλο action (σχήμα 5.7) ανάλογα με το αν ο χρήστης έκανε like ή unlike.

```
likeUnlikeSong (){
    if(!this.getLiked){
        this.$store.dispatch('LIKE_SONG',this.$route.params.id)
    } else {
        this.$store.dispatch('UNLIKE_SONG',this.$route.params.id)
    }
},
```

Σχήμα 5.6 Μέθοδος likeUnlikeSong

```
LIKE_SONG : ({commit},payload) => {
    return Axios.post(`/song/${payload}/like`).then(({status}) => {
        if(status===200) {
            commit('setLike',true)
        }
    })
},
UNLIKE_SONG : ({commit},payload) => {
    return Axios.post(`/song/${payload}/unlike`).then(({status}) => {
        if(status===200) {
            commit('setLike',false)
        }
    })
},
```

Σχήμα 5.7 Like και Unlike song actions

5.2.8 Player

O player θα είναι ένα ξεχωριστό component το οποίο διαχειρίζεται το playback του κομματιού. Πριν φορτωθεί η σελίδα με το κομμάτι, προσθέτουμε ένα div το οποίο θα έχει ένα attribute που θα το ονομάσουμε "data-source" ή όπως αλλιώς επιθυμούμε. Όπως φαίνεται στο σχήμα 5.8, το attribute αυτό θα περιέχει το route το οποίο έχουμε ετοιμάσει στο backend το οποίο παίρνει το mp3 αρχείο που αντιστοιχεί στο id του κομματιού.

Όταν ο χρήστης πατήσει το play, θα δημιουργήσουμε ένα **HTMLAudioElement** το οποίο θα αποθηκεύει όλες τις χρησιμες πληροφορίες του κομματιού όπως διάρκεια, αν είναι σταματημένο ή όχι. Το συγκεκριμένο element το αποθηκεύουμε στο store διότι θα μας χρειαστεί στη συνέχεια για να φτιάξουμε το BottomPlayer component.

Για την ένταση, ακολουθείται η εξής διαδικασία: στον slider της έντασης τοποθετούμε τον event listener onMouseDown. Όταν γίνει trigger, καλείται η μέθοδος updateBar (Σχήμα 5.9) η οποία δέχεται παραμετρικά το event. Υπολογίζουμε το ποσοστό της απόστασης που έχει ο slider από την αρχή του div και ανάλογα με αυτό, αλλάζουμε την ένταση του Audio element στο store και το width του κόκκινο χρώματος της έντασης.

Πιο αναλυτικά, πρώτα υπολογίζουμε την απόσταση που έχει το div της έντασης από την υπόλοιπη σελίδα με τη βοήθεια της μεθόδου getBoundingClientRect και του property scollLeft του body και την αποθηκεύουμε στη μεταβλητή left. Στη συνέχεια, αφαιρούμε τη θέση του slider (event.clientX) με τη μεταβλητή left για να πάρουμε το position του slider ως προς το div της έντασης. Βρίσκουμε το ποσοστό της απόστασης αν πολλαπλασιάσουμε το position επί εκατό και το διαιρέσουμε με το πλάτος της έντασης.

```
<div
  @click.capture="playSong"
  v-bind:data-source="$api + '/song/' + $route.params.id + '/songFile'"
  class="player-play player-button"
>
```

Σχήμα 5.8 Div με data-source του κομματιού

```
updateBar(event) {
  const volume = document.querySelector(".volume_slider-con");
  const eInner = document.querySelector(".volume_slider");
  let percentage;
  let volumeRect = volume.getBoundingClientRect();
  let left = volumeRect.left + document.body.scrollLeft;
  let position = event.clientX - left;
  percentage = (100 * position) / volume.clientWidth;

  if (percentage > 100) {
    percentage = 100;
  }
  if (percentage < 0) {
    percentage = 0;
  }

  //update volume bar and audio volume
  eInner.style.width = percentage + "%";
  this.getPlayingSong.volume = percentage / 100;
},
```

Σχήμα 5.9 Μέθοδος updateBar για την επεξεργασία της έντασης του κομματιού

5.2.9 BottomPlayer

Το component BottomPlayer θα δίνει τη δυνατότητα στο χρήστη να ακούσει ένα κομμάτι ακόμα και οταν φύγει από τη σελίδα του κομματιού. Με αυτόν τον τρόπο, βελτιώνεται το UX (**User Experience**) και ταυτόχρονα, δίνεται η ελευθερία στο χρήστη να περιηγηθεί στην ιστοσελίδα ακούγοντας κάποιο κομμάτι.

Για την υλοποίηση του component, θα εκμεταλλευτούμε το Audio element το οποίο υπάρχει ήδη στο store μαζί με όλα τα χρήσιμα mutations, actions, getters (σχήμα 5.10). Δημιουργούμε ένα div το οποίο θα περιέχει τη φωτογραφία του κομματιού, τον τίτλο, τον χρήστη που το ανέβασε και τα playback buttons. Όταν ο χρήστης πατάει play,pause ή stop, θα καλούμε τα κατάλληλα mutations με σκοπό το συγχρονισμό του Player και του BottomPlayer.

```
const playingSong = {
    state : {
        playingSong : null,
        playingSongObject : {},
        minutesDuration: "00",
        secondsDuration: "00",
        minutesCurrent: "00",
        secondsCurrent: "00",
    },
    getters : {
        getPlayingSong : (state) => {
            return state.playingSong
        },
        getPlayingSongPaused : (state) => {
            if(state.playingSong != null){
                return state.playingSong.paused
            } else {
                return true
            }
        },
        getPlayingSongObject : (state) => {
            return state.playingSongObject
        },
        getMinutesDuration : (state) => {
            return state.minutesDuration
        },
        getSecondsDuration : (state) => {
            return state.secondsDuration
        },
        getMinutesCurrent : (state) => {
            return state.minutesCurrent
        },
        getSecondsCurrent : (state) => {
            return state.secondsCurrent
        },
    },
```

Σχήμα 5.10 Getters και state για το παιζόμενο κομμάτι

5.2.10 Σχόλιο

Το component Comment.vue θα αποτελείται από δύο κύρια divs. Σύμφωνα με το σχήμα 5.11, το πρώτο div που θα περιέχει τη φόρμα για την εισαγωγή ενός σχολίου ενώ το δεύτερο div θα περιλαμβάνει όλα τα σχόλια που δημιουργούνται. Ο χρήστης αφού εισάγει ένα σχόλιο σε κάποιο κομμάτι, θα έχει την επιλογή να το διαγράψει.

Χρησιμοποιούμε τον getter isAuthenticated/isLoggedIn για να κρύψουμε κομμάτια της HTML όταν ο χρήστης δεν είναι συνδεδεμένος. Για να προσθέσουμε ένα σχόλιο, δημιουργούμε μια μέθοδο η οποία καλεί ένα action στο store στην οποία θα παιρνάμε ως payload (παράμετρος) το id του κομματιού στο

οποίο έγινε το σχόλιο, το text του comment και την ημερομηνία που δημιουργήθηκε το σχόλιο με τον DateConstructor (σχήμα 5.12). Το action αυτό (σχήμα 5.14) κάνει το request και αν το status του response είναι 201 (created) τότε προσθέτει το σχόλιο στο state με το κατάλληλο mutation (σχήμα 5.13).

Για το delete comment, χρησιμοποιούμε την ίδια μεθολογία με την εξαίρεση ότι στο action δίνουμε μόνο το id του κομματιού και το id του σχολίου που επιθυμούμε να διαγράψουμε.

```
  <div class="insert_comment" v-if="isLoggedIn">
     <img class="insert_comment-img" v-bind:src="$api + '/profile/' + getMyProfile._id + '/avatar'" >
     <form @submit.prevent="insertComment"  class="insert_comment-form">
        <input type="text" id="input"
        v-model="comment" class="insert_comment-input"
        placeholder="Insert your comment">
        <button class="insert_comment-btn">Create</button>
     </form>
  </div>

<h2 class="none" v-else>Log in to comment!</h2>
<div class="comments">
<div v-for="comment in getComments" :key="comment._id" class="comment">
    <img class="comment_img" v-bind:src="$api + '/profile/' + comment.owner.profile[0]._id + '/avatar'" >
    <div class="comment_content">
      <div class="comment_info">
        <h2 class="comment_author">{{ comment.owner.name }}</h2>
        <h2 class="comment_date">{{  comment.createdAt }}</h2>
      </div>
    <p class="comment_text">{{ comment.comment }}</p>
    <button class="comment_delete"
     v-if="isLoggedIn && comment.owner._id === getMyProfile.owner._id"
     @click="deleteComment(comment._id)" >
     Delete
    </button>
    </div>
</div>
</div>
```

Σχήμα 5.11 Comment.vue

```
 insertComment () {
     this.$store.dispatch('INSERT_COMMENT', {
         id : this.$route.params.id,
         comment : this.comment,
         createdAt: new Date().toLocaleString('en-GB').replace(',','')
```

Σχήμα 5.12 Μέθοδος για τη δημιουργία σχολίου

```
insertComment: (state,comment) => {
    comment.createdAt = new Date(comment.createdAt)
    .toLocaleString('en-GB',{year: 'numeric', month: 'numeric', day: 'numeric',hour:'2-digit',minute: '2-digit'})

    state.comments.unshift(comment)
},
```

Σχήμα 5.13 Mutation για τη δημιουργία σχολίου

```
INSERT_COMMENT : ({commit},payload) => {
        return Axios.post(`/song/${payload.id}/comment`,{comment:payload.comment,createdAt : payload.createdAt})
         .then(({status,data}) => {
             if(status===201){
                 commit('insertComment',data)
             }
         })
},
```

Σχήμα 5.14 Action για τη δημιουργία σχολίου

5.2.11 Σελιδοποίηση

Προκειμένου να μπορούμε να χρησιμοποιήσουμε σελιδοποίηση όπου χρειάζεται, φτιάχνουμε ένα ξεχωριστό component, Pagination.vue. Το συγκεκριμένο αρχείο θα περιέχει ένα button για Previous, ένα button για next και τα buttons για την αρίθμηση των σελίδων. Παράλληλα, δημιουργούμε και ένα αρχείο pagination.js στο store. Σαν limit θα χρησιμοποιήσουμε ένα σταθερό αριθμό, το εννιά. Αυτο σημαίνει πως σε κάθε σελίδα θα περιέχονται το πολύ εννιά σχόλια. Έπειτα, δημιουργούμε την μέθοδο getPage.

Όποτε ο χρήστης πατάει ένα κουμπί του pagination, τότε περνάμε παραμετρικά το page. Κάνουμε το mutation για να ενημερώσουμε το state του currentPage, κάνουμε push τον ρούτερ στο ίδιο path αλλά σε διαφορετικό page και τέλος, καλούμε το action GET_COMMENTS για να πάρουμε τα σχόλια ανάλογα με τη σελίδα που έχουμε επιλέξει (σχήμα 5.15).

Για το next και το previous ακολουθούμε την ίδια μεθολογία με τη διαφορά ότι αντι να επιλέξουμε το page ανάλογα με τον αριθμο του κουμπιού, προσθέτουμε/αφαιρούμε το ήδη υπάρχον page.

```
getPage(page) {
    this.$store.commit('setCurrentPage',page)
    this.$router.push({path: this.$route.path, query: {
        page
    }})
    let skip = 9 * page - 9
    this.$store.commit('setSkip',skip)
    this.$store.dispatch('GET_COMMENTS',this.$route.params.id)

},
```

Σχήμα 5.15 Μέθοδος getPage για το set της τρέχουσας σελίδας

5.2.12 Προστατευμένες διαδρομές

Τα protected routes είναι διαδρομές στις οποίες δεν επιτρέπεται η πρόσβαση σε μη-συνδεδεμένους χρήστες. Στο αρχείο index.js που υπάρχει στον φάκελο router, δημιουργούμε ένα middleware, το οποίο θα εκτελείται κάθε φορά που ο χρήστης αλλάζει route, με τη χρήση του beforeEach του Vue-router. Έτσι, με τον getter isAuthenticated που έχουμε φτιάξει στο store, ελέγχουμε αν ο χρήστης είναι συνδεδεμένος. Στην περίπτωση που δεν είναι, τότε, τον ανακατευθύνουμε στην σελίδα του Login.

Το αποτέλεσμα φαίνεται στο σχήμα 5.16.

```
router.beforeEach((to, from, next) => {
  if (to.name !== 'Login'
    && to.name !== 'Register'
    && to.name !== 'Index'
    && to.name !== 'Song'
    && to.name !== 'Home'
    && to.name !== 'Profiles'
    && to.name !== 'Profile'
    && !store.getters.isAuthenticated) {
    next({ name: 'Login' })
  } else {
    next()
  }
})
```

Σχήμα 5.16 Protected Routes Middleware

5.2.13 Ανέβασμα κομματιού

Δημιουργούμε ένα αρχείο Upload.vue το οποίο θα περιέχει τα inputs: Title,Description,Genre (dropdown),Track input (file) και το κουμπί Upload Track. Στη συνέχεια, στο file input, προσθέτουμε τον event listener onChange ο οποίος μόλις γίνεται trigger, θα καλεί τη μέθοδο getFile. Στη μέθοδο αυτή, παίρνουμε όλες τις πληροφορίες του αρχείου και τις αποθηκεύουμε στη μεταβλητή file. Στη συνέχεια, μόλις ο χρήστης πατήσει το κουμπί "Upload Track", καλούμε τη μέθοδο upload στην οποία αρχικοποιούμε τη **FormData** κλάση και την αποθηκεύουμε στη μεταβλητή form_data. Έπειτα, στη μεταβλητή αυτή, κάνουμε append όλα τα inputs σε μορφή json καθώς και τη μεταβλητή file που δημιουργήσαμε προηγουμένως. Τέλος, κάνουμε το request με το action upload στέλνωντας παραμετρικά τη μεταβλητή form_data. Το τελικό αποτέλεσμα είναι το σχήμα 5.17.

```
getFile(event){
    this.file =  event.target.files[0]
},
upload() {

    if(this.file == '') {
        let errorMessage = 'Please select a track'
        this.$store.commit('setError',errorMessage)
        return
    }
    var form_data = new FormData()

    form_data.append('song',JSON.stringify(this.form))
    form_data.append('upload',this.file)

    this.$store.dispatch('UPLOAD',form_data).then((success) => {
        this.form.title = ''
        this.form.description = ''
        this.form.genre = ''
        document.querySelector('.form_select').selected = 'Select genre : '
        this.file = ''
        this.$router.push('/tracks')
    }).catch((error) => {
        this.$store.commit('setError',error.response.data.error)
    })
}
```

Σχήμα 5.17 Μέθοδοι getFile και upload για το ανέβασμα του κομματιού

5.2.14 Μήνυμα

Στο Message.vue θα περιλαμβάνεται ένα div το οποίο θα περιέχει τα router-links δηλαδή τα άτομα με τα οποία έχει ανοίξει συνομιλία ο χρήστης. Για να εμφανίσουμε πολλά router-links, χρησιμοποιούμε το directive v-for για να εμφανίσουμε όσα άτομα υπάρχουν στις συζητήσεις. Τα άτομα θα αναδεικνύονται με την εικόνα προφίλ τους και με το όνομά τους. Ανάλογα με το αν ο χρήστης που είναι συνδεδεμένος, ξεκίνησε τη συζήτηση ή όχι, θα φιλτράρουμε και τα κατάλληλα δεδομένα. Επίσης, θα περιλαμβάνεται και ένα router view το οποίο θα αλλάζει ανάλογα με το id της συζήτησης. Με τη μέθοδο lifecycle-hook created παίρνουμε όλα τα conversations του χρήστη.

Το τελικό αποτέλεσμα φαίνεται στο σχήμα 5.18.

```
<div class="conversations">
    <div v-if="getMyProfile.owner">
        <router-link active class="conversation"
        tag="div" v-for="conversation in getConversations"
        :key="conversation._id"
        v-bind:to="'/message/conversation/' + conversation._id">
            <img class="conversation_img"
            v-bind:src="getMyProfile._id === conversation.sender.profile[0]._id ? $api + '/profile/' +
            conversation.recipient.profile[0]._id + '/avatar' : $api + '/profile/'
            + conversation.sender.profile[0]._id + '/avatar' ">
            <h2 class="conversation_name">{{getMyProfile._id === conversation.sender.profile[0]._id ?
            conversation.recipient.name : conversation.sender.name }}</h2>
        </router-link>
    </div>
</div>
```

Σχήμα 5.18 Message.vue

5.2.15 Συζήτηση

Στο Conversation.vue ξεκινάμε με ένα μεγάλο scrollable διν στο οποίο θα φαίνονται τα μηνύματα. Ένα input για την εισαγωγή μηνύματος, ένα button για την αποστολή του μηνύματος και ένα button για την αποστολή ενός κομματιού. Επίσης, ο χρήστης θα έχει τη δυνατότητα να πατήσει ανανέωση. Όταν ο χρήστης πατήσει το κουμπί Send τότε εκτελούμε τη μέθοδο sendMessage (σχήμα 5.19) και καλούμε το κατάλληλο action στέλνωντας παραμετρικά το μήνυμα και το id του χρήστη. Έπειτα, καλούμε τη scrollToEnd έτσι ώστε το scrolling του div να πάει προς την κάτω πλευρά.

Όταν ο χρήστης πατήσει την ανανέωση, καλούμε το action GET_MESSAGES και τη μέθοδο scrollToEnd.

```
 sendMessage() {
   if(this.message)
     this.$store.dispatch('SEND_MESSAGE',{
       message: this.message,
       recipient: this.to
       }).then((success) => {
       this.message = ''
       this.scrollToEnd()
     })
},
scrollToEnd() {
  const conversation = this.$refs.convo
  conversation.scrollTop = conversation.scrollHeight
},
```

Σχήμα 5.19 Μέθοδος sendMessage

5.2.16 Αναζήτηση

Για να αναζητήσουμε τους χρήστες δημιουργούμε ένα component Search.vue το οποίο θα περιέχει το input που χρειαζόμαστε. Τοποθετούμε τον event listener onKeyUp με τη μέθοδο search. Στη μέθοδο search, απλά εκτελούμε το action GET_PROFILES περνώντας παραμετρικά το όνομα του χρήστη που πληκτρολογήσαμε.

5.2.17 Loading

Υπάρχουν στιγμές όπου η σύνδεση μας στο διαδίκτυο είναι αργή ή ο server αντιμετωπίζει κάποιο overload. Γι'αυτόν ακριβώς τον λόγο οι loaders κάνουν την εμπειρία του χρήστη καλύτερη (**User Experience**). Για να εφαρμόσουμε τη λειτουργία loading σε όσα components θέλουμε, φτιάχνουμε ένα ξεχωριστό reusable component Loader.vue (σχήμα 5.20).

Δημιουργούμε το scss για τον κύκλο του loading (σχήμα 5.21) και για τα keyframes (σχήμα 5.22). Με τα keyframes ελέγχουμε το animation που θέλουμε κάνουμε. Το 0% υποδεικνύει την αρχή του animation ενώ το 100% το τέλος του.

Με v-if προϋποθέτουμε ότι ο Loader θα εμφανίζεται μόνο αν ο getter getLoading επιστρέφει true. Έπειτα με scss φτιάχνουμε έναν κύκλο που γυρίζει γύρω από τον εαυτό του συνεχώς. Παράλληλα, δημιουργούμε μια μεταβλητή loading και έναν getter getLoading στο index.js του store. Το trigger του

loading το ενεργοποιούμε με τους interceptors του axios. Στην ουσία, είναι functions που εκτελούνται πριν και μετά από ένα request όπως ακριβώς ενεργούσε το beforeEach στο Vue-Router.

```
<template>
  <div v-if="$store.getters.getLoading" class="loading_container">
    <div class="lds-ring">
      <div></div>
      <div></div>
      <div></div>
      <div></div>
    </div>
  </div>
</template>
```

Σχήμα 5.20 Loader.vue

```
.lds-ring {
  display: inline-block;
  position: relative;
  width: 80px;
  height: 80px;

  & > div {
    box-sizing: border-box;
    display: block;
    position: absolute;
    width: 64px;
    height: 64px;
    margin: 8px;
    border: 8px solid $color-gray-dark;
    border-radius: 50%;
    animation: lds-ring 1.2s cubic-bezier(0.5, 0, 0.5, 1) infinite;
    border-color: $color-gray-dark transparent transparent transparent;

    &:nth-child(1) {
      animation-delay: -0.45s;
    }

    &:nth-child(2) {
      animation-delay: -0.3s;
    }

    &:nth-child(3) {
      animation-delay: -0.15s;
    }
  }
```

Σχήμα 5.21 SCSS για τον κύκλο- loading

```
@keyframes lds-ring {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}
```

Σχήμα 5.22 Keyframes για το animation του Loader

5.2.18 Αρχική

Για να δημιουργήσουμε την αρχική μας σελίδα, δημιουργούμε τέσσερα διαφορετικά components TopSongs, UploadSection, ConnectSection και Footer. Το TopSongs θα περιέχει τα τέσσερα κομμάτια με περισσότερα likes και τα υπόλοιπα θα κάνουν την εφαρμογή πιο προσιτή προς στο χρήστη δείχνοντας του πως να τη χρησιμοποιήσει. Επίσης, ο Footer θα περιέχει links με τα οποία γίνεται navigation στην ιστοσελίδα (Σχήμα 5.23).

```
<template>
<div class="index">
  <!-- <Loader /> -->

  <div class="index-section">
      <div class="heading">
        <h2 class="heading_primary margin-bottom-medium">MusicPart</h2>
        <h5 class="heading_secondary">Share your passion.Share your music.</h5>
        <router-link to="/login" class="btn-primary btn_animated">Get Started</router-link>
      </div>
  </div>
  <TopSongs />
  <UploadSection />
  <ConnectSection />
  <Footer />
</div>
</template>
```

Σχήμα 5.23 Index.vue

5.2.19 Μεταβλητές περιβάλλοντος

Μεταβλητές περιβάλλοντος ονομάζονται οι μεταβλητές οι οποίες κατέχουν χρήσιμες πληροφορίες όπως το url του api όπου πρέπει να παραμένουν ασφαλείς. Έτσι, αντί να έχουμε το url του api γραμμένο στον κώδικα δημιουργούμε ένα .env αρχείο και το τοποθετούμε εκεί. Έπειτα στο main.js της Vue, χρησιμοποιούμε τη μεταβλητή με αυτόν τον τρόπο (σχήμα 5.24). Έτσι, αν το url αλλάξει κάποια στιγμή, δεν θα χρειαστεί να το αλλάξουμε σε όλα τα αρχεία παρά μόνο στο αρχείο .env.

```
Vue.prototype.$api = process.env.VUE_APP_API
```

Σχήμα 5.24 Δημιουργία prototype για το URL του API

5.3 Back-End

5.3.1 Σχήματα

Για να δημιουργήσουμε τα σχήματα, χρησιμοποιούμε το mongoose. Τα μοντέλα που θα ορίσουμε είναι τα εξής: Comment, Conversation, Like, Message, Profile, Song, User. Δημιουργούμε ξεχωριστά αρχεία για κάθε μοντέλο μέσα στον φάκελο models. Κάθε αρχείο θα αποτελείται από το σχήμα το οποίο περιλαμβάνει πεδία του σχήματος που έχουμε ορίσει καθώς επίσης και σχέσεις μεταξύ των μοντέλων [34].

Σε αυτό το σημείο είναι απαραίτητο να ξεχωρίσουμε τις έννοιες «μοντέλο» και «σχήμα». Το σχήμα είναι η δομή του document στη βάση. Το μοντέλο είναι η κλάση με την οποία αλληλεπιδρούμε και κάνουμε αλλαγές στο αντίστοιχο σχήμα που βρίσκεται στη βάση.

5.3.2 Σχέσεις

Πολλές φορές θα χρειαστεί να αναφέρουμε ένα μοντέλο μέσα σε ένα άλλο μοντέλο δηλώνοντας κάποια σχέση μεταξύ τους. Αν και η MongoDB δεν αποτελεί σχεσιακή βάση δεδομένων, χρησιμοποιεί τα document references για να υποδείξει σχέσεις μεταξύ των μοντέλων.

Πιο συγκεκριμένα, μέσω του mongoose, δηλώνουμε μια virtual μέθοδο στο σχήμα που θέλουμε η οποία θα αποτελείται από τα πεδία ref (το μοντέλο στο οποίο αναφέρεται, refference), το localField το οποίο είναι το τοπικό πεδίο του μοντέλου και το foreignField το οποίο είναι το ξένο πεδίο του μοντέλου στο οποίο βασίζεται η σχέση αυτή (σχήμα 5.25) [35].

```
userSchema.virtual('profile',{
    ref: 'Profile',
    localField: '_id',
    foreignField: 'owner'
})

userSchema.virtual('comments',{
    ref: 'Comment',
    localField: '_id',
    foreignField: 'owner',
})
```

Σχήμα 5.25 Μέθοδοι virtuals για relations

5.3.3 Σχήμα χρήστη

Βασικά πεδία του user είναι το username, password, email, name, type (μουσικός ή παραγωγός) και τα jwts τα οποία θα αποθηκεύονται στη βάση. Στο username και στο password βάζουμε το property trim: true για να μην ληφθούν υπόψη backslashes και κενά. Για το validation του email, χρησιμοποιούμε τη βιβλιοθήκη validator μέσα στη μέθοδο validate της Mongoose. Τέλος, κάνουμε export το σχήμα για τη μελλοντική του χρήση. Παρατηρούμε το αποτέλεσμα στο σχήμα 5.26.

```
const userSchema = new mongoose.Schema({
    username: {
        type: String,
        required: true,
        trim: true
    },
    password: {
        type: String,
        required: true,
        trim: true,
        minlength: 8,
    },
    name : {
        type: String,
        required: true
    },
    type: {
        type: String,
        required: true
    },
    email: {
        type : String,
        required: true,
        unique: true,
        validate(value){
            if(!validator.default.isEmail(value)){
                throw new Error('Email is invalid')
            }
        }
    },
```

Σχήμα 5.26 User Schema

5.3.4 Σχήμα κομματιού

Τα πεδία του κομματιού θα αποτελούνται από τον τίτλο, την περιγραφή, το genre, τον αριθμό των likes, την εικόνα του κομματιού, το αρχείο και το χρήστη που ανέβασε το κομμάτι, τον owner. Όπως παρατηρούμε στο σχήμα 5.27, τα likes θα έχουν σαν προεπιλογή 0, η εικόνα θα είναι τύπου Buffer και θα έχει ως προεπιλογή το path μιας εικόνας που βρίσκεται στο server, το αρχείο θα είναι τύπου Buffer και ο owner θα έχει τύπο ObjectID και θα αποτελεί σχέση ως προς τον User που έχουμε φτιάξει. Επειδή το πότε δημιουργήθηκε το κομμάτι είναι χρήσιμο για την εφαρμογή, αντί να βάλουμε ένα πεδίο createdAt και να το γεμίζουμε εμείς, προσθέτουμε την επιλογή timestamps: true.

Επίσης, θέτουμε τις επιλογές virtuals σε true για να ενεργοποιούμε τις σχέσεις αυτόματα κάθε φορά που παίρνουμε ή κάνουμε insert ένα κομμάτι στη βάση.

```
const songSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    genre: {
        type: String,
        required: true
    },
    likes : {
        type: Number,
        default: 0
    },
    image : {
        type: Buffer,
        default: fs.readFileSync(path.resolve((__dirname), '../assets/img/song.jpg'))
    },
    file: {
        type: Buffer,
        required: true
    },
    owner: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User'
    }
},{timestamps:true},{toJSON: {virtuals: true},toObject: {virtuals:true}})
```

Σχήμα 5.27 Song Schema

5.3.5 Σχήμα σχολίου

Τα σχόλια θα αποτελούνται από το κείμενο του σχολίου, το κομμάτι στο οποίο έγινε το σχόλιο και το χρήστη που έκανε το σχόλιο. Όπως και στο Song, έτσι και εδώ χρησιμοποιούμε τα timestamps για να δημιουργηθούν αυτόματα τα dates δημιουργίας των σχολίων (Σχήμα 5.28).

```
const commentSchema = new mongoose.Schema({
    song : {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'Song'
    },
    comment : {
        type: String,
        required: true
    },
    owner: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User'
    }
},{timestamps:true})
```

Σχήμα 5.28 Comment Schema

5.3.6 Σχήμα προφίλ

Το προφίλ θα περιλαμβάνει την ηλικία του χρήστη, την τοποθεσία, τα όργανά του, το avatar και το πεδίο owner το οποίο θα είναι reference στον User. Σύμφωνα με το σχήμα 5.29, τα instruments θα είναι ένα array με strings και η προεπιλεγμένη τιμή του avatar θα περιέχει το path της προεπιλεγμένης εικόνας που θα έχει ο χρήστης.

```
const profileSchema = new mongoose.Schema({
    age : {
        type : Number,
        trim: true
    },
    city: {
        type: String,
        default: ''
    },
    instruments: [{
        instrument : {
            type: String
        }
    }],
    avatar : {
        type: Buffer,
        default: fs.readFileSync(path.resolve((__dirname), '../assets/img/avatar.jpg'))
    },
    owner: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User'
    }
})
```

Σχήμα 5.29 Profile Schema

5.3.7 Σχήμα μηνύματος

Το σχήμα μηνύματος (σχήμα 5.30) θα αποτελείται από τα πεδία: sender (User), message, file, recipient (User) και conversation (Conversation). Περιλαμβάνουμε το πεδίο file για να μπορέσουμε να στείλουμε ένα κομμάτι στην ιδιωτική συνομιλία μας.

```
const messageSchema = new mongoose.Schema({
    sender : {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User'
    },
    message: {
        type: String
    },
    file : {
        type: Buffer
    },
    recipient : {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User'
    },
    conversation : {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'Conversation'
    }
},{timestamps:true})
```

Σχήμα 5.30 Message Schema

5.3.8 Σχήμα συζήτησης

Όταν δημιουργείται μια συζήτηση μεταξύ δύο χρηστών, πρέπει να ξέρουμε ποιος είναι ο αποστολέας και ποιος ο παραλήπτης. Έτσι, όπως βλέπουμε το σχήμα 5.31, θα έχουμε τα πεδία sender και recipient τα οποία θα είναι references στον User.Τα types που θα έχουν θα είναι objectIds και βάζουμε required: true έτσι ώστε αν κάποιο από αυτά τα πεδία δεν δωθεί, τότε το mongoose πετάει αυτόματα error. Επίσης, δημιουργούμε τη σχέση με τα μηνύματα για να υποδείξουμε σε ποιο conversation ανήκει το κάθε message.

```
const conversationSchema = new mongoose.Schema({
    sender : {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User'
    },
    recipient: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: 'User'
    }
},{toJSON: {virtuals: true}})

conversationSchema.virtual('messages', {
    ref: 'Message',
    localField: '_id',
    foreignField: 'conversation'
})



const Conversation = new mongoose.model('Conversation',conversationSchema)
```

Σχήμα 5.31 Conversation Schema

5.3.9 Σχήμα like

Προκειμένου να έχουμε τη δυνατότητα να αναγνωρίσουμε ποιος χρήστης έχει κάνει like, το σχήμα του Like είναι απαραίτητο. Περιλαμβάνει τα πεδία song και owner τα οποία αποτελούν το κομμάτι στο οποίο έγινε το like και ο χρήστης που το έκανε (Σχήμα 5.32).

```
const mongoose = require('mongoose')

const likeSchema = new mongoose.Schema({
    song: {
        type: mongoose.Schema.Types.ObjectId,
        required: true
    },
    owner: {
        type: mongoose.Schema.Types.ObjectId,
        required: true
    }
})


const Like = new mongoose.model('Like', likeSchema)

module.exports = Like
```

Σχήμα 5.32 Like Schema

5.3.10 Αυθεντικοποίηση

Προκειμένου να έχουμε αυθεντικοποίηση στο API, θα χρησιμοποιήσουμε το node module «jsonwebtoken». Είναι ένας από τους πιο γνωστούς τρόπους για να εφαρμοστεί η αυθεντικοποίηση. Η διαδικασία είναι η εξής: παράγουμε ένα jwt και το στέλνουμε όταν ο χρήστης συνδεθεί με επιτυχία. Ο χρήστης για τα επόμενά του request προσκολεί το jwt στον header “Authorization”.

Το συγκεκριμένο jwt είναι μοναδικό και ο χρήστης στη μεριά του client έχει πρόσβαση σε αυτό μέχρι να γίνει logout ή γίνει κάποιο time-out. Στη συνέχεια, με τη βοήθεια του middleware που θα βάλουμε (σχήμα 5.33), αποκρυπτογραφούμε το συγκεκριμένο jwt με τη βοήθεια του secret-key που έχουμε ορίσει. Εάν η κρυπτογράφηση γίνει με επιτυχία τότε ο χρήστης θα πάρει το response που αναμένει. Σε περίπτωση που η κρυπτογράφηση είναι ανεπιτυχής τότε στέλνουμε ένα error [36].

```
const User = require('../models/user')
const jwt = require('jsonwebtoken')

const auth = async (req, res, next) => {
    try {
        const token = await req.header('Authorization').replace('Bearer ', '')
        const decoded = jwt.verify(token, process.env.JWT_SECRET)
        const user = await User.findOne({ _id: decoded._id, 'tokens.token': token })
        if (!user) {
            throw new Error()
        }
        req.token = token
        req.user = user
        next()
    } catch (e) {
        res.status(401).send({ error: 'Not authenticated' })
    }

}

module.exports = auth
```

Σχήμα 5.33 Authentication Middleware

5.3.11 Διαδρομές

Για τις διαδρομές, δημιουργούμε τα αρχεία comment, conversation, like, message, profile, song, user στο φάκελο routers. Σε κάθε αρχείο, φτιάχνουμε τον router μέσω του Express, εισάγουμε τα απαραίτητα μοντέλα και στη συνέχεια δηλώνουμε όλα τα routes με τη συγκεκριμένη μορφή όπως φαίνεται στο σχήμα 5.34.

Σε κάθε route δηλώνουμε το path του route και μια ασύγχρονη μέθοδο (callback) την οποία μας δίνει το Express. Σε αυτή τη μέθοδο, εισέρχονται παραμετρικά το request και το response. Μπορούμε να πάρουμε τους headers, params του request ή και να στείλουμε κάποιο response. Επίσης, πριν από τη

ασύγχρονη μέθοδο, μπορούμε να προσκολλήσουμε το authentication middleware όπου είναι απαραίτητο [37].

Χρησιμοποιούμε τις try catch για να πιάσουμε τα errors και να εμφανίσουμε κατάλληλο μήνυμα. Τέλος, κάνουμε export τον router και τον χρησιμοποιούμε στο index.js για να το δηλώσουμε (app.use()).

```
const express = require('express')
const router = express.Router()
const Like = require('../models/like')
const Song = require('../models/song')


router.post('/api/song/:id/like', async (req, res) => {
    const foundLike = await Like.findOne({ owner: req.user._id, song: req.params.id })
    if (foundLike) {
        return res.status(400).send({ error: 'You have already liked this song.' })
    }
    const like = new Like({
        song: req.params.id,
        owner: req.user._id
    })
    try {
        await like.save()
        const song = await Song.findById(req.params.id)
        song.likes++
        await song.save()
        res.send()
    } catch (e) {
        res.status(400).send(e)
    }

})
```

Σχήμα 5.34 Υπόδειγμα χρήσης routes μέσω Express

5.3.12 Διαδρομή εγγραφής

Όταν ο χρήστης κάνει εγγραφή, τότε αποθηκεύουμε τα στοιχεία του αρχικοποιώντας το μοντέλο User και προσθέτοντας παραμετρικά το body του request. Έπειτα, δημιουργούμε ένα προφίλ για το χρήστη που έκανε εγγραφή (Σχήμα 5.35). Στη συνέχεια, ελέγχουμε αν υπάρχουν εγγραφές είτε στο username είτε στο email του χρήστη με τη μέθοδο checkDuplicates (Σχήμα 5.36) την οποία έχουμε δημιουργήσει στο μοντέλο User. Αν το username ή το email υπάρχει ήδη στη βάση, κάνουμε throw Error και αφήνουμε την Express να αναλάβει το error με τον δικό της Error Handler.

Στη μέθοδο pre on save της Mongoose, προσθέτουμε τη λογική του hash για το password με τη βοήθεια της βιβλιοθήκης Bcrypt (Σχήμα 5.37) [38].

Παράγουμε ένα jwt με τη βοήθεια της μεθόδου generateAuthToken (Σχήμα 5.38), σώζουμε το user και profile που μόλις δημιουργήσαμε στη βάση και στέλνουμε σαν response τον user και το jwt.

Για να αποκρύψουμε τον κωδικό και τα jwts χρησιμοποιούμε τη μέθοδο toJSON η οποία ειναι ένα middleware το οποίο τρέχει πριν μετατραπεί από object σε json το body του response και σταλθεί στον client (Σχήμα 5.39).

```
router.post('/api/register', async (req, res) => {
    const user = new User(req.body)
    const profile = new Profile({
        owner : user._id
        })

    try {
        await user.checkDuplicates(user.username,user.email)
        const token = await user.generateAuthToken()
        await user.save()
        await profile.save()
        res.status(201).send({user,token})
    } catch (e){
        res.status(400).send(e)
    }

})
```

Σχήμα 5.35 Register Route

```
userSchema.methods.checkDuplicates = async (username,email) => {
    const existsUsername = await User.countDocuments({username})
    const existsEmail = await User.countDocuments({email})

    if(existsUsername>0){
        const error = {
            path : 'username',
            message: 'This username already exists.Please choose another one'
        }
        throw (error)
    }
    if(existsEmail>0){
        const error = {
            path : 'email',
            message: 'This email already exists.Please choose another one'
        }
        throw (error)
    }
}
```

Σχήμα 5.36 Μέθοδος checkDuplicates

```
userSchema.pre('save', async function(next) {
    const user = this
    if(user.isModified('password')){
        user.password = await bcrypt.hash(user.password,8)
    }
    next()

})
```

Σχήμα 5.37 Password hashing

```
userSchema.methods.generateAuthToken = async function() {

    const user = this
    const token = jwt.sign({_id : user._id.toString()},process.env.JWT_SECRET)
    user.tokens = user.tokens.concat({ token })

    return token

}
```

Σχήμα 5.38 Μέθοδος generateAuthJwt

```
userSchema.methods.toJSON = function(){
    const userObject = this.toObject()

    delete userObject.password
    delete userObject.tokens

    return userObject

}
```

Σχήμα 5.39 Μέθοδος για την απόκρυψη κωδικό και jwts

5.3.13 Διαδρομή σύνδεσης

Για τη σύνδεση του χρήστη, ελέγχουμε αν το username και ο κωδικός που έδωσε ο χρήστης αντιστοιχούν σε κάποιο χρήστη με τη μέθοδο checkCredentials (σχήμα 5.41) περνώντας παραμετρικά το username και τον κωδικό. Εάν βρεθεί κάποιος χρήστης, κάνουμε generate jwt και στέλνουμε σαν response τον user και το jwt που μολις δημιουργήσαμε (Σχήμα 5.40).

```
try {
    const user = await User.checkCredentials(req.body.username,req.body.password)
    const token = await user.generateAuthToken()
    await user.save()
    res.send({user,token})
} catch (e){
    res.status(400).send({error: e.message})
}
```

Σχήμα 5.40 Login Route

```
// Login authentication
userSchema.statics.checkCredentials = async (username,password) => {
     const user = await User.findOne({ username })

    if(!user){
        throw new Error('Credentials do not match')
    }
    const isMatched = await bcrypt.compare(password,user.password)
    if(!isMatched) {
        throw new Error('Credentials do not match')
    }
    return user

}
```

Σχήμα 5.41 Μέθοδος checkCredentials

5.3.14 Διαδρομή αποσύνδεσης

Για την αποσύνδεση του χρήστη, αφαιρούμε το jwt από τον πίνακα jwts που υπάρχει στο req, αποθηκεύουμε τις αλλαγές και στέλνουμε ένα κενό response με status 200 (σχήμα 5.42).

```
router.post('/api/logout', auth, async (req, res) => {
    try {
        req.user.tokens = req.user.tokens.filter((token) => {
            return token.token !== req.token
        })
        await req.user.save()
        res.send()
    } catch (error){
        res.status(500).send(error.message)
    }
})
```

Σχήμα 5.42 Logout Route

5.3.15 Διαδρομή ανεβάσματος κομματιού

Για να ανεβεί το κομμάτι στο σέρβερ και να το στείλουμε πίσω στο frontend όταν χρειαστεί, θα χρησιμοποιήσουμε το module Multer. Στην περίπτωση μας, το μετατρέπουμε σε μορφή **Base64** και το αποθηκεύουμε στη βάση. Με τις κατάλληλες ρυθμίσεις του middleware (Σχήμα 5.44) προστατεύουμε τον σερβερ μας από ανεπιθύμητα ή πολύ μεγάλα αρχεία.

Στη συνέχεια, τοποθετούμε το middleware μετά από το authentication middleware στο route μας. Αποθηκεύουμε τις πληροφορίες του κομματιού, τον owner και τον buffer του αρχείου (**base64**) και στέλνουμε status 200 (Σχήμα 5.43).

```
router.post('/api/song', auth, uploadSong.single('upload'), async (req, res) => {
    var body = JSON.parse(req.body.song)
    body = {
        ...body,
        owner: req.user._id,
        file: req.file.buffer
    }
    const song = new Song(body)
    await song.save()
    res.send()
}, (error, req, res, next) => {
    res.status(400).send({ error: error.message })
})
```

Σχήμα 5.43 Route για το ανέβασμα κοματιού με τη χρήση της βιβλιοθήκης Multer

```
const uploadSong = multer({
    limits: {
        fileSize: 7000000
    },
    fileFilter(req, file, cb) {
        if (!file.originalname.match(/\.(mp3|wav)$/)) {
            cb(new Error('File must be mp3 or wav format'))
        }
        cb(undefined, true)
    }
})
```

Σχήμα 5.44 Ρυθμίσεις για τη χρήση του Multer

5.3.16 Διαδρομή κομματιού

Για να βρούμε το κομμάτι, χρησιμοποιούμε τη μέθοδο μοντέλου findById περνώντας παραμετρικά το id από το GET request. Προσθέτουμε τον header «Content-type: audio/mpeg» και στέλνουμε το αρχείο με res.send (Σχήμα 5.45).

```
router.get('/api/song/:id/songFile', async (req, res) => {
    try {
        const song = await Song.findById(req.params.id)
        if (!song) {
            return res.status(404).send()
        }
        res.set('Content-type', 'audio/mpeg')
        res.send(song.file)
    } catch (e) {
        res.status(400).send(e)
    }
})
```

Σχημα 5.45 Get Song Route

5.3.17 Διαδρομή διαγραφής κομματιού

Για να διαγράψουμε ένα κομμάτι, χρησιμοποιούμε τη μέθοδο του μοντέλου findOneAndDelete περνώντας παραμετρικά το id του τραγουδιού από τα params και το id του owner. Έπειτα, για να διαγράψουμε όλα τα σχόλια που υπήρχαν σε αυτό το κομμάτι χρησιμοποιούμε τη μέθοδο deleteMany περνώντας παραμετρικά το id του κομματιού. Το αποτέλεσμα φαίνεται στο σχήμα 5.46.

```
router.delete('/api/song/:id', auth, async (req, res) => {
    try {
        const song = await Song.findOneAndDelete({
            _id: req.params.id,
            owner: req.user._id
        })
        if (!song) {
            return res.status(404).send()
        }
        await Comment.deleteMany({
            song: req.params.id
        })
        res.send(song)

    } catch (e) {
        res.status(400).send({ error: e.message })
    }
})
```

Σχήμα 5.46 Delete Song Route

5.3.18 Διαδρομή like κομματιού

Σύμφωνα με το σχήμα 5.47, για να κάνουμε like σε ένα κομμάτι, πρέπει πρώτα να ελέγξουμε αν έχουμε κάνει ήδη like στο συγκεκριμένο κομμάτι. Αν έχουμε ήδη κάνει like, πετάμε και το σχετικό error. Στη συνέχεια, δημιουργούμε ένα καινούργιο Like με τα properties song το οποίο είναι το id του κομματιού (το παίρνουμε μέσω params) και owner, το id του συνδεδεμένου χρήστη. Στη συνέχεια, σώζουμε το like στη βάση και έπειτα βρίσκουμε το τραγούδι στο οποίο έγινε το like με findById. Αυξάνουμε τον integer likes κατά ένα και σώζουμε το τραγούδι. Τέλος, στέλνουμε κενή απάντηση με 200 status.

Για το dislike song, η διαδικασία είναι παρόμοια με τη μόνη διαφορά ότι διαγράφουμε το like και μειώνουμε κατά ένα τα likes του τραγουδιού.

```
router.post('/api/song/:id/like', async (req, res) => {
    const foundLike = await Like.findOne({ owner: req.user._id, song: req.params.id })
    if (foundLike) {
        return res.status(400).send({ error: 'You have already liked this song.' })
    }
    const like = new Like({
        song: req.params.id,
        owner: req.user._id
    })
    try {
        await like.save()
        const song = await Song.findById(req.params.id)
        song.likes++
        await song.save()
        res.send()
    } catch (e) {
        res.status(400).send(e)
    }

})
```

Σχήμα 5.47 Like Song Route

5.3.19 Διαδρομή ενημέρωσης προφίλ

Το update του προφίλ περιλαμβάνει 3 properties: την ηλικία, την τοποθεσία και τα όργανα. Ελέγχουμε αν το body περιλαμβάνει τα 3 συγκεκριμένα properties. Εαν δεν τα περιλαμβάνει τότε πετάμε error με status 400. Εαν τα παριλαμβάνει, τότε βρίσκουμε το προφίλ με τη βοήθεια του req.user.id, αντικαθιστούμε τα properties, σώζουμε το προφίλ στη βάση και στέλνουμε πίσω το προφίλ σαν response έχοντας κάνει populate το owner property (Σχήμα 5.48).

```
router.patch('/api/profile', auth, async (req, res) => {
    const updates = Object.keys(req.body)
    const allowedUpdates = ['age','city','instruments']
    const isValidOperation = updates.every((update) => allowedUpdates.includes(update))

    if(!isValidOperation){
        return res.status(400).send({error: 'Invalid Updates!'})
    }
    try {
        const profile = await Profile.findOne({owner: req.user._id}).select('-avatar')
        updates.forEach((update) => {
            profile[update] = req.body[update]
        })

       await profile.save()
       await profile.populate('owner').execPopulate()
       res.send(profile)
    } catch (error){
        res.status(400).send({error:error.message})
    }
})
```

Σχήμα 5.48 Update Profile Route

5.3.20 Διαδρομή αποστολής μηνύματος

Για να στείλουμε ένα μήνυμα, θα πρέπει πρώτα να δούμε αν υπάρχει κάποιο conversation στη βάση με sender είτε τον χρήστη που έστειλε το μήνυμα είτε τον recipient του μηνύματος και το αντίστοιχο για τον recipient.

Αν υπάρχει η συζήτηση στη βάση, τότε δεν χρειάζεται να δημιουργήσουμε καινούργια συζήτηση. Απλά δημιουργούμε ένα καινούργιο Message το οποίο θα περιλαμβάνει το body του request (message,recipient), τον sender ο οποίος θα είναι το id του user που είναι συνδεδεμένος και το conversation που είναι το id του conversation που βρήκαμε (Σχήμα 5.49).

Εάν δεν υπάρχει στη βάση (Σχήμα 5.50), δημιουργούμε το conversation, δημιουργούμε το Message με τη μόνη διαφορά ότι τώρα στο property conversation, θα βάλουμε το id του conversation που μόλις φτιάξαμε. Σώζουμε το message και το conversation και το στέλνουμε σαν response κάνοντας populate το πεδίο sender.

```
router.post('/api/message', auth, async (req,res) => {

    try {
        const foundConversation = await Conversation.findOne({
            $or : [
             {sender : req.user._id, recipient: req.body.recipient},
             {sender : req.body.recipient, recipient: req.user._id }
            ]
        })

      if(foundConversation){
        const message = await new Message({
            ...req.body,
            sender: req.user._id,
            conversation : foundConversation._id
        })

        await message.save()
        await Message
        .populate(message,{path: 'sender'},function(err,doc) {
          return res.send(doc)
      })
    }
```

Σχήμα 5.49 Send Message Route - Περίπτωση 1η

```
      const conversation = await new Conversation({
          sender: req.user._id,
          recipient: req.body.recipient
      })

      const message = await new Message({
        ...req.body,
        sender: req.user._id,
        conversation : conversation._id
  })

    await message.save()
    await conversation.save()

    await Message
    .populate(message,{path: 'sender'},function(err,doc) {
        res.send(doc)
    })
```

Σχήμα 5.50 Send Message Route - Περίπτωση 2

5.3.21 Διαδρομή αποστολής αρχείου μέσω μηνύματος

Για να στείλουμε το κομμάτι ως μήνυμα συνδυάζουμε την τακτική του μηνύματος με την τακτική της αποστολής κομματιού που είδαμε παραπάνω. Τοποθετούμε το middleware του multer στη διαδρομή και στη συνέχεια δημιουργούμε το καινούργιο μήνυμα με τα πεδία text, sender, recipient, conversation, file (Σχήμα 5.51). Στη συνέχεια, στέλνουμε το μήνυμα σαν απάντηση.

```
    const message = await new Message({
      ...req.body,
      sender: req.user._id,
      file: req.file.buffer,
      conversation : conversation._id
})

  await message.save()
  await conversation.save()

  await Message
  .populate(message,{path: 'sender',populate: {path: 'profile',select: 'name'}},function(err,doc) {
     return res.send(doc)
  })

  res.send(message)
```

Σχήμα 5.51 Αποστολή μηνύματος με κομμάτι

5.3.22 Σελιδοποίηση

Όπως βλέπουμε στο σχήμα 5.52, για τη σελιδοποίηση χρησιμοποιούμε τα queries που μας στέλνει ο client για να ορίσουμε πεδία όπως το skip και το limit. Χρησιμοποιούμε το populate για να γεμίσουμε πεδία με σχέσεις τα οποία χρειάζονται στο front-end. Έτσι, με τις επιλογές που μας δίνει το mongoose όπως το skip,limit και sort, ανακατεσκευάζουμε την απάντηση έτσι ώστε να περιέχει σελιδοποίηση.

```
router.get('/api/song/:id/comment', async (req,res) => {
    try {
        let totalPages = 0
        const comments = await Comment.find({song : req.params.id})
        totalPages = Math.ceil(comments.length / 9)


        await Comment.find({song : req.params.id})
        .limit(parseInt(req.query.limit))
        .skip(parseInt(req.query.skip))
        .sort({'createdAt': -1})
        .populate({path: 'owner',populate: {path:'profile',select: '_id'}})
        .exec((err,docs) => {
            res.send({
                comments : docs,
                totalPages
            })
        })


    } catch (e) {
        res.status(400).send({error: e.message})
    }
```

Σχήμα 5.52 Παράδειγμα χρήσης Pagination

5.4 Επίλογος

Σε αυτό το κεφάλαιο υλοποιήσαμε όλη την εφαρμογή. Ξεκινήσαμε με το front-end και ολοκληρώσαμε λειτουργίες όπως σελιδοποίηση, προστατευμένες διαδρομές κλπ. Εκμεταλλευτήκαμε πλήρως τα στοιχεία του Vuex και το store του κάνοντας κλήσεις API και αλλάζωντας το state. Δημιουργήσαμε τα απαιραίτητα components που χρειάζεται η εφαρμογή όπως Loader, Song, Player. Όσο αναφορά το back-end, δημιουργήσαμε το μοντέλα – σχήματα, σχέσεις μεταξύ τους, μεθόδους για την επεξεργασία των δεδομένων πριν ή μετα το request, τις απαραίτητες διαδρομές για την εφαρμογή μας.

Κεφάλαιο 6: Συμπεράσματα και βελτιώσεις

Η πτυχιακή αυτή ήταν πάρα πολύ ενδιαφέρουσα καθώς είχε να κάνει κυρίως με μουσική. Χρησιμοποιήθηκαν πολλές τεχνολογίες με αποτέλεσμα να μου δωθεί η ευκαιρία να βελτιωθώ στον προγραμματισμό και να εξοικειωθώ με εργαλεία τα οποία δεν είχα ασχοληθεί στο παρελθόν. Επίσης, έμαθα πολλά πρόγματα για το πως λειτουργεί γενικότερα το web, οι βάσεις δεδομένων και πως γίνεται η αλληλεπίδραση μεταξύ front-end και back-end.

Όσο αναφορά το αποτέλεσμα, κάθε χαρακτηριστικό που σχεδιάστηκε για να γίνει στην εφαρμογή, ολοκληρώθηκε. Παρ'όλ'αυτά, θα μπορούσαν να γίνουν μεγάλες βελτιώσεις.

Μια σημαντική βελτίωση που θα μπορούσε να εφαρμοστεί είναι η αλλαγή όλου το κώδικα στα πλαίσια της TypeScript. Η TypeScript είναι μια ειδική compilable γλώσσα η οποία αποτελεί τη Javascript μαζί με πολλά χαρακτηριστικά όπως types (number, string, array), interfaces (όπως στη Java) κλπ. Αυτή η αλλαγή θα έκανε τον κώδικα πιο ευανάγνωστο και πιο εύκολο να εφαρμοστούν αλλαγές πάνω σε αυτόν.

Ένα πρόβλημα που υπάρχει στην εφαρμογή είναι στα μηνύματα. Ο χρήστης, για να δει αν έχει έρθει καινούργιο μήνυμα πρέπει να πατήσει ανανέωση. Για αυτόν τον λόγο, θεωρώ άκρως απαραίτητη τη χρήση Sockets. Το socket, ουσιαστικά, είναι ένα connection το οποίο παραμένει ανοιχτό ανάμεσα σε client και server. Μόλις γίνεται μια αλλαγή στον client, ο server «ακούγοντας» την αλλαγή αυτή, την αναμεταδίδει στους άλλους clients που είναι συνδεδεμένοι σε αυτόν χωρίς να γίνει κάποιο request. Έτσι, όταν ένας χρήστης στείλει μήνυμα σε κάποιον άλλον, δεν θα χρειαστεί το κουμπί της ανανέωσης.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[1] Berners-Lee Tim, Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. San Francisco: Harper, 2000.

[2] Jeffrey Zeldman, Designing With Web Standards, New Riders, 2009.

[3] Meyer Eric Cascading Style Sheets: The Definitive Guide, Third Edition, O'Reilly Media, 2006.

[4] Meyer Eric Cascading Style Sheets 2.0 Programmer's Reference, McGraw-Hill Osborne Media, 2001.

[8] Flanagan David, JavaScript: The Definitive Guide, O'Reilly Media, 2006.

[9] Herman David, Effective JavaScript, Addison-Wesley, 2013.

[10] Nelson Brett, Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch, Apress, 2018.

[11] Macrae Callum, Vue.js: Up and Running: Building Accessible and Performant Web Apps, O'Reilly Media, 2018.

[16] Hughes Croucher, Up and Running with Node.js, O'Reilly Media, 2012.

[17] Ornbo George, Sams Teach Yourself Node.js in 24 Hours, SAMS Publishing, 2012.

[18] Teixeira Pedro, Professional Node.js, John Wiley & Sons, 2012.

[19] Gackenheimer Cory, Node.js Recipes: A Problem-Solution Approach, Apress, 2013.

[20] Pirtle, Mitch, MongoDB for Web Development (1st ed.), Addison-Wesley Professional, 2011.

[21] Chodorow, Kristina; Dirolf, Michael, MongoDB: The Definitive Guide (1st ed.), O'Reilly Media, 2010.

[22] Banker Kyle, MongoDB in Action (1st ed.), Manning, 2011.

[23] Wiese Lena, Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases, DeGruyter/Oldenbourg, 2015.

[24] Canavan John, Fundamentals of Networking Security. Norwood, MA: Artech House, 2001.

[25] Richardson Leonard, RESTful Web APIs, O'Reilly Media, 2013.

[28] Ojamaa, Andres; Duuna, Karl, Assessing the Security of Node.js Platform, International Conference for Internet Technology and Secured Transactions, IEEE, 2012.

[30] Jones Michael, JSON Web Token (JWT), IETF, 2015.

Internet Site

[5] Sass Team 2006-2020 https://sass-lang.com/guide

[6] Sass Mixins https://sass-lang.com/documentation/at-rules/mixin

[7] Sass Variables https://sass-lang.com/documentation/variables

[12] Vue JS Components https://vuejs.org/v2/guide/components.html

[13] Vue JS Directives https://v3.vuejs.org/api/directives.html

[14] Vuex https://vuex.vuejs.org/guide/

[15] Vue Router https://router.vuejs.org/guide/#javascript

[26] Vue JS Life Cycle Hooks https://vuejs.org/v2/guide/instance.html

[27] Base64 MDN https://developer.mozilla.org/en-US/docs/Glossary/Base64

[29] Fetch API MDN https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[31] HTMLAudioElement https://developer.mozilla.org/en-US/docs/Web/API/HTMLAudioElement

[32] FormData API MDN https://developer.mozilla.org/en-US/docs/Web/API/FormData

[33] Axios https://github.com/axios/axios#axios-api

[34] Mongoose Schema https://mongoosejs.com/docs/guide.html

[35] Mongoose Virtuals https://mongoosejs.com/docs/guide.html#virtuals

[36] Express Middleware https://expressjs.com/en/guide/using-middleware.html

[37] Express Routing https://expressjs.com/en/guide/routing.html

[38] Bcrypt https://github.com/kelektiv/node.bcrypt.js/

[39] Multer https://github.com/expressjs/multer