ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Διαδικτυακή εφαρμογή επικοινωνίας κειμένου & ήχου»

Των φοιτητών:
Αλέξιαδη-Σελαμανίδη Τηλέμαχου
Αρ. Μητρώου: 164630
Ταναηλίδη Χρήστου
Αρ. Μητρώου: 164752

Επιβλέπων:
Αντώνιος Σιδηρόπουλος
Αναπληρωτής Καθηγητής

Ημερομηνία: 30/01/2024

Τίτλος: Διαδικτυακή εφαρμογή επικοινωνίας κειμένου & ήχου.
Κωδικός: 22107
Ονοματεπώνυμο φοιτητή: Αλεξιάδης-Σελαμανίδης Τηλέμαχος
Ονοματεπώνυμο φοιτητή: Ταναηλίδης Χρήστος
Ονοματεπώνυμο εισηγητή: Σιδηρόπουλος Αντώνιος
Ημερομηνία ανάληψης: 30/01/2022
Ημερομηνία περάτωσης: 30/01/2024

Βεβαιώνουμε ότι είμαστε οι συγγραφείς αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχαμε για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχουμε καταγράψει τις όποιες πηγές από τις οποίες κάναμε χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνουμε ότι αυτή η εργασία προετοιμάστηκε από εμάς προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Στον αγαπημένο συνάδελφο και φίλο Κομνά για τα γραφιστικά,
στον συνάδελφο και φίλο Χρήστο για την έμπρακτη βοήθεια,
στους φίλους και συναδέλφους,
στις οικογένειες μας,
και στους χρήστες της εφαρμογής που βοήθησαν για την δοκιμή της
Mithiko_teras, Syler, sar, Freemix, xristospav,
Mariapap, alator21, HarambeIsMissing, sisia, Purrcatory,
nasia, sotiris, AMOGUS, thopekas, Sassette, KatM,
TheoT, kurioskwlotoumpas, KomnasTheGreat,
mrgatoss, Natalia_dou, dtsia.»

Πρόλογος

Ο λόγος που επιλέξαμε την υλοποίηση της συγκεκριμένης εφαρμογής έχει να κάνει κυρίως με το πλήθος τον τεχνολογιών που θα μας επέτρεπε να χρησιμοποιήσουμε. Οι σημερινές εφαρμογές επικοινωνίας μας παρέχουν πλέον πολλούς τρόπους για να επικοινωνούμε. Η υλοποίηση λοιπόν μια τέτοιας εφαρμογής θα μας επέτρεπε να ασχοληθούμε με θέματα αυθεντικοποίησης του χρήστη και τεχνολογίες όπως το JWT, την αποστολή μηνυμάτων σε πραγματικό χρόνο μεταξύ δύο ή πολλών χρηστών ταυτόχρονα με την χρήση του πρωτοκόλλου WebSocket καθώς επίσης και την επικοινωνία δύο χρηστών μέσω φωνής χρησιμοποιώντας τεχνολογίες όπως το WebRTC σε συνδυασμό με το WebSocket. Αυτοί ήταν οι κύριοι λόγοι που επιλέξαμε μια εφαρμογή σαν αυτή, καθώς μας βοήθησαν να κατανοήσουμε περισσότερο πτυχές αυτών των τεχνικών & τεχνολογιών που μας ενδιέφεραν.

Περίληψη

Η εργασία αφορά την υλοποίηση διαδικτυακής εφαρμογής, όπου οι χρήστες της μπορούν να επικοινωνούν με άλλους χρήστες, σε πραγματικό χρόνο, μέσω μηνυμάτων και ήχου. Αρχικά οι χρήστες μπορούν να εγγραφούν στην εφαρμογή και μετά την επιβεβαίωση της ηλεκτρονικής αλληλογραφίας τους, να συνδεθούν σε αυτή. Έπειτα, μπορούν να στέλνουν αιτήματα φιλίας σε άλλους χρήστες ή να δημιουργούν ομάδες όπου μπορούν να συμμετέχουν διάφοροι χρήστες μέσω των προσκλήσεων που έστειλε ο δημιουργός της. Για να μπορέσει να πραγματοποιηθεί η επικοινωνία, οι χρήστες θα πρέπει να έχουν ανταλλάξει αιτήματα φιλίας και να έχουν γίνει αποδεκτά ή να συμμετέχουν σε κάποιο κοινό χώρο ομάδας. Για την περάτωση του, αναπτύχθηκαν δύο κομμάτια, αυτό του back-end και αυτό του front-end. Το back-end υλοποιήθηκε χρησιμοποιώντας το framework Node.js και πρόκειται για ένα GraphQL API, ενώ το front-end υλοποιήθηκε με την βοήθεια του framework Vue.js. Χρησιμοποιήθηκαν διάφορες τεχνολογίες, όπως το Mikro-ORM για την διαχείριση της MySQL βάσης δεδομένων, το Socket.io για την επικοινωνία σε πραγματικό χρόνο με το UI και το WebRTC όπου είναι υπεύθυνο για την πραγματοποίηση της επικοινωνίας ήχου.

Text & Audio Web Application

Alexiadis-Selamanidis Tilemachos

& Tanaildis Christos

Abstract

The thesis is about implementing a web application, where users can communicate with other users in real time, through messages and audio. At first, users can register in the application and after confirming their emails, they can log in. Then, they can send friend requests to other users or create groups where different users can participate through invitations sent by the creator. In order for communication to take place, users must have exchanged and accepted their friend requests or be members in a shared group. The application was developed in two parts, the back-end and the front-end. The back-end was implemented using the Node.js framework, and it is a GraphQL API and the front-end was implemented with the help of the Vue.js framework. Various technologies were used, such as the Mikro-ORM to manage the MySQL database, Socket.io for real-time communication with the UI and WebRTC which is responsible for making the audio communication.

Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε τους φίλους, την οικογένεια και τους συναδέλφους μας, που βοήθησαν με διάφορους τρόπους στην ολοκλήρωση αυτής της εργασίας. Επίσης θα θέλαμε να ευχαριστήσουμε και τον Κ. Σιδηρόπουλο, που μας επέτρεψε να αναλάβουμε την εργασία αυτή και μας βοήθησε να την ολοκληρώσουμε.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Αποκομμάτων Κώδικα

Συντομογραφίες

UI	User Interface (Διεπαφή Χρήστη)

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Το όνομα της εφαρμογής είναι sousou και αποτελεί ουσιαστικά ένα web application, που προσφέρει τη δυνατότητα στους χρήστες της να επικοινωνούν υπό την μορφή κειμένου ή ήχου. Για την είσοδο στην εφαρμογή, απαιτείται εγγραφή, κατά την οποία οι χρήστες δημιουργούν τον δικό τους λογαριασμό. Μετά την εγγραφή, και την είσοδο τους στην εφαρμογή, οι χρήστες έχουν την δυνατότητα να αναζητήσουν και να προσθέσουν άλλους χρήστες στην λίστα φίλων τους δημιουργώντας αιτήματα φιλίας. Επίσης μπορούν να δημιουργήσουν ομάδες, να προσκαλέσουν σε αυτές νέα μέλη και να τροποποιήσουν κάποιες ρυθμίσεις που την αφορούν. Επιπλέον, μέσα στις ομάδες, υπάρχει η δυνατότητα δημιουργίας καναλιών επικοινωνίας, που είναι το βασικό μέσο για την ανταλλαγή μηνυμάτων ή την έναρξη κάποιας ηχητικής κλήσης μεταξύ τους. Με την επιλογή ενός καναλιού από ένα μέλος, αποκτάται η πρόσβαση σε αυτό και η επικοινωνία μπορεί να αρχίσει άμεσα. Η επικοινωνία γίνεται ταυτόχρονα και σε πραγματικό χρόνο, από όλα τα μέλη που βρίσκονται εκείνη τη στιγμή στο κανάλι.

1.2 Περιγραφή Αρχιτεκτονικής

Η εφαρμογή αποτελείται από δύο σκέλη, το front-end και το back-end. Και τα δύο σκέλη χρησιμοποιούν την γλώσσα Typescript που κατά βάση είναι Javascript με περισσότερες δυνατότητες και ευκολίες. Το back-end είναι γραμμένο σε NodeJS σε συνδυασμό με την GraphQL και χρησιμοποιεί βιβλιοθήκες για την διευκόλυνση διαφόρων λειτουργιών της εφαρμογής, πχ η διαχείριση της MySQL βάσης γίνεται μέσω της βιβλιοθήκης Mikro-ORM, η αποστολή των email γίνεται με την χρήση της βιβλιοθήκης nodemailer κ.α. Επίσης, είναι υπεύθυνο για την εισαγωγή, επεξεργασία και ανάκτηση όλων των δεδομένων της εφαρμογής μέσω της GraphQL καθώς και για την αποστολή των τοποθεσιών των χρηστών στο διαδίκτυο (WebRTC candidates) μέσω WebSockets για την χρήση του voice-chat. Επομένως, όλη η λογική της εφαρμογής υλοποιείται στο back-end με σκοπό το front-end να επεξεργάζεται, όσο λιγότερο γίνεται, τα δεδομένα.

Το front-end κομμάτι της εργασίας ασχολείται με το UI του χρήστη και την επικοινωνία με το back-end είτε μέσω των GraphQL αιτημάτων, είτε μέσω του WebSocket με μηνύματα πραγματικού χρόνου (real time). Επίσης ασχολείται και με την P2P (peer to peer) επικοινωνία που απαιτείται για την έναρξη συνομιλίας μεταξύ χρηστών μέσω φωνής, η οποία είναι απαραίτητη για την τεχνολογία του WebRTC. Η υλοποίηση γίνεται με το framework της Vue.js σε συνδυασμό με την TypeScript και συνεργατικά με άλλες βιβλιοθήκες που ασχολούνται με διάφορα τμήματα που αφορούν το UI (πχ tailwind για μορφοποίηση μέσω CSS κλάσεων, Quasar για components φορμών, παραθύρων κλπ.). Το UI είναι reactive (αναδραστικό) και ενημερώνει συνεχώς τον χρήστη για τις αλλαγές που συμβαίνουν στην εφαρμογή.

Κατά την διάρκεια της υλοποίησης της εφαρμογής, δόθηκε ιδιαίτερη σημασία στην εύκολη επεκτασιμότητα και συντήρηση της, καθώς γνωρίζοντας -από άλλα κοινωνικά δίκτυα- υπάρχει η δυνατότητα να προστεθούν πολλές νέες λειτουργίες στο μέλλον.

1.3 Περιγραφή Κεφαλαίων

Στα παρακάτω κεφάλαια παρουσιάζονται αναλυτικά οι πληροφορίες για τις τεχνολογίες οι οποίες χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Η σχεδίαση της βάσης δεδομένων καθώς και όλες

τις απαραίτητες πληροφορίες για την κατανόηση του σχήματος της βάσης. Οι τρόποι και οι μεθοδολογίες που χρησιμοποιήθηκαν για την υλοποίηση του back-end αλλά και του front-end ξεχωριστά. Η ανάλυση το τρόπου που έγινε το deploy της εφαρμογής. Και τέλος αναφερόμαστε στην τελική μορφή της εφαρμογής καθώς και την χρήση της, με πλήρη επεξήγηση του τρόπου λειτουργίας.

Κεφάλαιο 2ο: Τεχνολογίες

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα ασχοληθούμε με τις τεχνολογίες που χρησιμοποιήσαμε για την υλοποίηση της εφαρμογής στο front-end και στο back-end. Από την κοινή γλώσσα προγραμματισμού την TypeScript, στην NodeJS και στον τρόπο που μας επιτρέπει να συντάσσουμε ένα JavaScript service στο server-side. Την ενναλακτική από την REST μεθοδολογία επικοινωνίας GraphQL που αλλάζει τον τρόπο επικοινωνίας του front-end με το back-end. Την τεχνολογία WebRTC που χρησιμοποιήθηκε για την επικοινωνία μέσω μορφής ήχου σε πραγματικό χρόνο από client προς client. Το πρωτόκολλο WebSockets και την βιβλιοθήκη socket.io που μας επιτρέπει την αποστολή δεδομένων σε πραγματικό χρόνο από το back-end στο front-end και το αντίστροφο. Η υλοποίηση του front-end που βασίστηκε πλήρως στο Framework της Vue.js καθώς και μία πιο ενισχυμένη εξέταση του πως δουλεύει και του τι προσφέρει. Τα δύο βοηθητικά εργαλεία για την υλοποίηση του front-end, την βιβλιοθήκη component Quasar και την βιβλιοθήκη μορφοποίησης CSS κλάσεων WindiCSS. Οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση των βάσεων δεδομένων όπως η MySQL και η τεχνολογία Mikro-ORM η οποία ασχολείται με την συσχέτιση των δεδομένων από την βάση σε αντικείμενα του back-end. Καθώς επίσης το Git και το VCS τα οποία είναι υπεύθυνα για την διατήρηση του πηγαίου κώδικα σε μορφή εκδόσεων. Την τεχνολογία JWT και την εξασφάλιση της εγκυρότητας των δεδομένων που προσφέρει. Και τέλος το PM2 process το οποίο είναι υπεύθυνο για την έκδοση της εφαρμογής.

2.2 JavaScript & TypeScript

Η γλώσσα που επιλέχθηκε για την ανάπτυξη της εφαρμογής, στο back-end αλλά και στο front-end, είναι η TypeScript. Δημιουργήθηκε το 2012 από την Microsoft και αποτελεί ένα υπερσύνολο της γλώσσας προγραμματισμού JavaScript[1], [2]. Η αρχική υλοποίηση της JavaScript έγινε το 1995 από την Netscape με σκοπό να χρησιμοποιηθεί για την δημιουργία online (web) εφαρμογών που συνδέουν μεταξύ τους αντικείμενα και πόρους στους πελάτες (clients) αλλά και στους εξυπηρετητές (servers). Η JavaScript, είναι γνωστή κατά βάση για την δυναμική και διαδραστική της συμπεριφορά στις web εφαρμογές. Ως Web εφαρμογές ορίζονται οι εφαρμογές που αξιοποιούνται μέσω κάποιου περιηγητή ιστού σε ένα δίκτυο, χρησιμοποιώντας γλώσσες που είναι συμβατές με αυτόν. Ως εφαρμογή εξυπηρετητή (server-side) ορίζεται μια εφαρμογή η οποία εκτελείται σε έναν εξυπηρετητή και ασχολείται με διάφορες λειτουργίες, όπως για παράδειγμα τη διαχείριση των δεδομένων και την επικοινωνία με βάσεις δεδομένων. Σε αντίθεση με τις εφαρμογές του πελάτη στις οποίες ο χρήστης έχει άμεση επαφή, καθώς τρέχει στην συσκευή του και επικεντρώνεται στο UI. Το σύνολο αυτών των δύο "πλευρών" αποτελεί μία ολοκληρωμένη web εφαρμογή. Αν και κατά βάση γλώσσα ανάπτυξης εφαρμογών πελάτη, η JavaScript τα τελευταία χρόνια η έχει υιοθετηθεί και στην πλευρά του εξυπηρετητή. Οι εφαρμογές εξυπηρετητή (γνωστές και ως back-end) έχουν την δυνατότητα να υλοποιηθούν με JavaScript χρησιμοποιώντας το περιβάλλον Node.js, το οποίο επιτρέπει την χρήση της JavaScript και στον εξυπηρετητή. Περισσότερα για την Node.js αναφέρονται στο κεφάλαιο 2.3 με λεπτομέρειες καθώς αποτελεί μέρος και της δικής μας υλοποίησης.

Η συγκεκριμένη γλώσσα μας προσφέρει τα οφέλη μιας αντικειμενοστρεφούς (object-oriented) γλώσσας προγραμματισμού. Καθώς επίσης και όλες τις γνωστές λειτουργίες κάθε γλώσσας όπως οι δομές ροών (if, if-else, κ.ο.κ.), οι επαναλήψεις (for, while, κ.ο.κ.), η δυνατότητα σύνταξης μεθόδων και η αποθήκευση μεταβλητών (χωρίς όμως την δήλωση τύπων). Επιπρόσθετα, από τις πολύ σημαντικές

δυνατότητες της συγκεκριμένης γλώσσας είναι η διαχείριση των αρχείων με μορφή "Μοντέλου Αντικειμένου Εγγράφου" (DOM - Document Object Model). Η μορφή αυτών των αρχείων έχει μια ιεραρχική δομή δέντρου. Μερικές από τις τεχνολογίες αυτής της μορφής αποτελούν οι HTML και XML. Η JavaScript έχει την δυνατότητα να διαχειρίζεται, να χρησιμοποιεί καθώς και να μορφοποιεί ετικέτες (tags) που βρίσκονται στην ιεραρχική αυτή δομή. Αυτό είναι ένα από τα κύρια προτερήματα της γλώσσας σε ό,τι αφορά την ανάπτυξη Web εφαρμογών.

Επίσης σημαντική είναι και η δυνατότητα διαχείρισης των γεγονότων (events) σχετικά με την HTML. Δηλαδή, ο τρόπος με τον οποίο στις Web διεπαφές ο πελάτης (χρήστης) μπορεί να δημιουργήσει κάποια εναύσματα (triggers), τα οποία η JavaScript μπορεί να "ακούσει" και να εκτελέσει μία ροή εντολών αντίστοιχα. Παραδείγματα εναυσμάτων είναι το πάτημα ενός κουμπιού, ή είσοδος του κέρσορα σε κάποιες ετικέτες, η επιλογή στοιχείων από το UI κ.α. Με αυτόν τον τρόπο η JavaScript έχει καταφέρει να εδραιωθεί στον χώρο του front-end, καθώς προσφέρει ένα πολύ ισχυρό εργαλείο για την δυναμική λειτουργία του κώδικα αλλά και την μορφοποίηση του UI, μέσω εναυσμάτων του χρήστη, σε συνεργασία πάντα με άλλες τεχνολογίες όπως είναι η HTML και τα CSS.

Μία ακόμη πολύ ισχυρή λειτουργία της γλώσσας αποτελεί και ο ασύγχρονος προγραμματισμός, ο οποίος πλέον αποτελεί ένα πολύ σημαντικό μέρος της ανάπτυξης Web εφαρμογών. Η δυνατότητα αυτή επιτρέπει στον προγραμματιστή να διαχειρίζεται την ασύγχρονή επικοινωνία με μηχανισμούς όπως ανακλήσεις (callbacks) και υποσχέσεις (promises). Με τις ασύγχρονες μεθόδους μία εφαρμογή μπορεί να συνταχθεί με τέτοιο τρόπο, ώστε οι διαδικασίες που εκτελούνται σε αυτή να μπορούν να περιμένουν διάφορες κλήσεις σε APIs που χρειάζονται κάποιο χρόνο να ολοκληρωθούν, και έπειτα να προχωρήσει η εκτέλεση της υπόλοιπης ροής του κώδικα, με μία σειριακή ροή.

Αν και πολύ ισχυρή από μόνη της η JavaScript παραμένει ελαφρώς πίσω σε κάποια κομμάτια της ανάπτυξης. Όπως αναφέρουμε και παραπάνω, η γλώσσα που επιλέξαμε για την δική μας υλοποίηση είναι η TypeScript. Κύριος λόγος της επιλογής αυτής, αποτελεί η προσθήκη του static typing (στατική τυποποίηση)[2]. Η συγκεκριμένη δυνατότητα εισάγει στον κόσμο της JavaScript την δυνατότητα να ορίσουμε τύπους δεδομένων σε μεταβλητές, παραμέτρους συναρτήσεων καθώς και στις επιστρεφόμενες τιμές τους. Αυτό επιφέρει πολλά οφέλη για την ανάπτυξη του κώδικα καθώς και στην μετέπειτα συντήρηση του. Η πρόωρη ανίχνευση σφαλμάτων, η ευαναγνωσιμότητα και η ενισχυμένη ποιότητα του κώδικα σε συνδυασμό με την -πλέον- ισχυρή κοινότητα που την υποστηρίζει, είναι κάποια από τα πολύ σημαντικά της προνόμια που την κάνουν μία τόσο καλή επιλογή. Όλες οι προσθήκες που αναφέρουμε κατά την εκτέλεση του κώδικα αφαιρούνται και ουσιαστικά μετατρέπονται σε μία «καθαρή» JavaScript υλοποίηση. Δηλαδή δεν παραμένουν κομμάτια που αφορούν την στατική τυποποίηση στον τελικό κώδικα του εκτελέσιμου (run-time), αφού περάσουν από τον μεταφραστή (compiler)[1].

2.3 NodeJS

Η Node.js είναι μια πλατφόρμα ανοιχτού κώδικα που επιτρέπει την εκτέλεση Javascript χωρίς την παρουσία κάποιου περιηγητή. Έτσι, επιτρέπει την δημιουργία Javascript server-side εφαρμογών εκμεταλλεύοντας όλες τις χρήσιμες λειτουργίες της. Ο Ryan Dahl, δημιουργός της Node.js, κυκλοφόρησε την πρώτη της έκδοση το 2009, με σκοπό να προσφέρει μία ελαφριά και αποτελεσματική πλατφόρμα, που να δίνει την δυνατότητα να αναπτυχθούν επεκτάσιμες διαδικτυακές εφαρμογές. Το 2010 με 2012 έγινε αρκετά δημοφιλής απόκτησε μεγάλη δημοφιλία καθώς πολλοί προγραμματιστές ξεκίνησαν να την χρησιμοποιούν για την ανάπτυξη των back-end εφαρμογών τους, εκμεταλλεύοντας τα χαρακτηριστικά της όπως εύκολη επεκτασιμότητα και ταυτόχρονη διαχείριση πολλαπλών

συνδέσεων. Η χρήση της είναι ωφέλιμη σε πολλά επίπεδα. Σημαντικό όφελος, είναι η δυνατότητα ανάπτυξης διαδικτυακών εφαρμογών με την χρήση μίας μόνο γλώσσας προγραμματισμού. Έτσι προωθεί την μεθοδολογία του full-stack development, καθώς οι προγραμματιστές της εφαρμογής δεν χρειάζεται να γνωρίζουν κάποια άλλη γλώσσα προγραμματισμού, και μπορούν να εμπλέκονται και στο front-end κομμάτι της εφαρμογής, χωρίς να σπαταλάται χρόνος για προσαρμογή από τη μία γλώσσα προγραμματισμού στην άλλη[3].

2.3.1 NPM

Υπάρχουν πολλές λειτουργίες και μέθοδοι που είναι συνεχώς κοινές από εφαρμογή σε εφαρμογή. Πολλές από αυτές τις επαναλαμβανόμενες λειτουργίες, μπορούν να εμπεριέχονται σε πακέτα/βιβλιοθήκες (packages/libraries) και να δίνονται στους προγραμματιστές έτοιμες για χρήση. Δημιουργούνται και συντηρούνται κυρίως από την κοινότητα της εκάστοτε γλώσσας προγραμματισμού και συνήθως είναι ανοιχτού κώδικα, επιτρέποντας στους προγραμματιστές να επωφεληθούν γλυτώνοντας χρόνο από την υλοποίηση των ίδιων λειτουργιών. Τα πακέτα διαχειρίζονται από κάποιο εργαλείο διαχείρισης πακέτων (package manager), όπου είναι υπεύθυνα για την απλοποίηση των διαδικασιών εγκατάστασης, αναβάθμισης και επιλογή έκδοσης των πακέτων μιας εφαρμογής.

Το npm είναι ένας package manager της Javascript, και είναι αναμφίβολα το πιο δημοφιλές. Για την δημοφιλία του ευθύνεται η εύκολη διαχείριση των πακέτων, καθώς προσφέρει εύκολες και κατανοητές σε σύνταξη εντολές για την εγκατάσταση, την αναβάθμιση και την αφαίρεση πακέτων από το project. Τα αρχεία των πακέτων αποθηκεύονται στον φάκελο node_modules, ο οποίος συνήθως δεν συμπεριλαμβάνεται από το σύστημα διαχείρισης εκδόσεων (VCS - Version Control System) διότι καταλαμβάνει μεγάλη χωρητικότητα στον δίσκο και θα καθυστερούσε η κλωνοποίηση (clone) του project. Χρησιμοποιεί το αρχείο package.json για την αποθήκευση των μεταδεδομένων (metadata) των πακέτων, όπως το όνομα τους και την έκδοση τους, ώστε με την εντολή `npm install` να εγκαταστήσει όλα τα πακέτα του project στον φάκελο node_modules. Επίσης, δίνει την δυνατότητα εγκατάστασης πακέτων στο σύστημα του προγραμματιστή, ανεξάρτητα από το project, όπου συνήθως αποτελούν εργαλεία γραμμής εντολών. Πολύ σημαντικό, είναι πως δίνει την δυνατότητα στον καθένα να προσφέρει τις δικές του υλοποιήσεις σε ανοιχτό κώδικα, στους υπόλοιπους προγραμματιστές, και να βοηθήσει στην γρήγορη ανάπτυξη και εξέλιξη της κοινότητας[4].

2.4 GraphQL

Η GraphQL είναι μια γλώσσα ερωτημάτων ανοιχτού κώδικα για την δημιουργία APIs και αποτελεί μία τελείως διαφορετική μεθοδολογία από αυτή του REST. Το ταξίδι της ξεκίνησε το 2012 από την τότε Facebook (Meta) για την χρήση της ως εσωτερικό εργαλείο της εταιρείας. Το 2015 κυκλοφόρησε ως τεχνολογία ανοιχτού κώδικα επιτρέποντας έτσι τη χρήση της από την προγραμματιστική κοινότητα[5].

Ξεκίνησε να γίνεται δημοφιλής καθώς εξελισσόταν, με κύριο χαρακτηριστικό της επιτυχίας της, την αποτελεσματική ανάκτηση δεδομένων από τους clients. Ουσιαστικά τους δίνει την δυνατότητα, να ανακτούν όποια δεδομένα τους είναι χρήσιμα, την συγκεκριμένη στιγμή, λύνοντας έτσι δύο σημαντικά προβλήματα. Το πρώτο ονομάζεται over-fetching όπου πρόκειται για την ανάκτηση μεγάλου όγκου δεδομένων που δεν είναι χρήσιμα την προκειμένη στιγμή. Το δεύτερο ονομάζεται under-fetching και αφορά τις περιπτώσεις όπου ο μηχανικός του API δεν έχει προβλέψει τα δεδομένα που θα χρειαστεί ο client και δεν τα έχει συμπεριλάβει στο εκάστοτε request[5]. Η GraphQL επιλύει τα συγκεκριμένα προβλήματα χρησιμοποιώντας μόνο ένα end-point όπου παράγει τα κατάλληλα ερωτήματα σύμφωνα με τις ανάγκες του χρήστη την στιγμή που τα ζητάει και βάσει ενός αυστηρού τυποποιημένου σχήματος δεδομένων, όπου δηλώνεται στο API. Έτσι ο προγραμματιστής του API, μπορεί να ακολουθεί

περίπλοκες δομές δεδομένων αρκεί να δηλώνει συνεχώς το σχήμα των δεδομένων και ο προγραμματιστής του client είναι υπεύθυνος για να το ακολουθεί και να το τηρεί.

Άλλο σημαντικό χαρακτηριστικό της, είναι η συμβατότητα με τις παλαιότερες εκδόσεις (backwards compatibility) . Προκύπτει από τα παραπάνω, καθώς όταν προστίθενται νέα χαρακτηριστικά (features) ή νέα πεδία στο σχήμα των δεδομένων, δεν επηρεάζει την λειτουργία της ήδη υπάρχουσας δομής των client[6]. Οι clients συνεχίζουν να καλούν τα δεδομένα που χρειάζονται χωρίς να επηρεάζονται από τις αλλαγές του server. Έτσι δεν είναι απαραίτητο να αναβαθμιστούν ταυτόχρονα και οι δύο πλευρές, εκτός αν η πλευρά του client χρειαστεί δεδομένα και λειτουργίες από το καινούριο σχήμα του API.

2.4.1 Queries

Τα GraphQL Queries χρησιμοποιούνται για να δώσουν την δυνατότητα στον χρήστη να ανακτήσει δεδομένα από το API. Ουσιαστικά είναι μέθοδοι στο API όπου ανακτούν δεδομένα από την βάση και στην συνέχεια επιστρέφουν στον client τα δεδομένα που έχει δηλώσει πως χρειάζεται. Δέχονται μεταβλητές ερωτημάτων οι οποίες χρησιμοποιούνται από τις μεθόδους για να παραμετροποιήσουν τα δεδομένα. Πχ παράμετροι για την ταξινόμηση των δεδομένων με συγκεκριμένο πεδίο, για την σελιδοποίηση (pagination) των δεδομένων, για φίλτρα που να ικανοποιούν συγκεκριμένα πεδία στην βάση κ.α. Επίσης, δίνει την δυνατότητα στον client να χρησιμοποιεί τμήματα ερωτημάτων (fragments), όπου είναι επαναχρησιμοποιήσιμες δηλώσεις δεδομένων, ώστε ο προγραμματιστής να μην επαναλαμβάνει τα πεδία που χρειάζεται σε όμοια queries. Επιτρέπει την χρήση ντιρεκτιβών (directives), όπου είναι συνθήκες που δηλώνει ο client, για το αν θα ανακτήσει το πεδίο ή όχι. Φυσικά, σε περίπτωση λανθασμένων δεδομένων αναζήτησης/φίλτρων, τα Queries μπορούν να επιστρέψουν περιγραφικά σφάλματα (errors) για να ενημερώσουν τον client για την αιτία του σφάλματος.

2.4.2 Mutations

Τα GraphQL Mutations χρησιμοποιούνται για την επεξεργασία των δεδομένων στον server. Ενώ τα Queries είναι υπεύθυνα μόνο για την ανάκτηση δεδομένων, τα Mutations εισάγουν, επεξεργάζονται και διαγράφουν δεδομένα. Όπως και τα Queries, είναι δηλωμένα στο GraphQL σχήμα. Το κάθε Mutation έχει δηλωμένα στο σχήμα τύπους εισαγωγής (input types) και επιστροφής (return types) δεδομένων. Επίσης, τα Mutations περιέχουν κανόνες επικύρωσης (validation rules), όπου σε περίπτωση λανθασμένου input, δεν θα επιτρέψουν την εκτέλεση τους. Οι input τύποι αποτελούνται από πολλά πεδία εισαγωγής. Οι return τύποι συνήθως είναι οι τύποι των δεδομένων που επεξεργάζονται τα Mutations. Σκοπός τους είναι η επιστροφή των νέων δεδομένων, ώστε ο client να ανανεώσει τα επικυρωμένα δεδομένα από τον server. Όπως και στα Queries, ο client μπορεί να διαλέξει τα πεδία που επιθυμεί να επιστρέψει το εκάστοτε Mutation. Επίσης, όμοια με τα queries, αν κάτι δεν λειτουργήσει όπως θα έπρεπε και υπάρχει σφάλμα, θα επιστραφεί error στον client για να τον ενημερώσει για τον λόγο του σφάλματος πχ ‘Not Found Error’, ή ‘Validation Error’ κ.α.

2.4.3 Subscriptions

Τα subscriptions (συνδρομές) παρέχουν ένα μηχανισμό πραγματικού χρόνου επικοινωνίας μεταξύ του GraphQL server και του client. Σε αντίθεση με τα Queries και τα Mutations, όπου βασίζονται στην λογική "ερώτημα-απάντηση". Επιτρέπουν στους clients να δέχονται ανανεωμένα δεδομένα από τον server όταν συμβούν συγκεκριμένα events. Στα event του subscription ο client μπορεί να εγγραφτεί όποτε και αν επιθυμεί να λαμβάνει δεδομένα, όταν αυτά υπάρξουν. Μπορούν επίσης να εγγραφούν σε όσα subscriptions επιθυμούν. Τα subscritpions έχουν υλοποιηθεί αξιοποιώντας το WebSocket πρωτόκολλο όπου επιτρέπει την αμφίδρομη επικοινωνία. Θα αναφερθούμε σε αυτό αναλυτικότερα

παρακάτω, στο κεφάλαιο 2.6. Τα subscriptions μπορούν να δεχθούν μεταβλητές, ώστε οι clients να έχουν πρόσβαση στα δεδομένα που τους αφορούν και τους επιτρέπεται. Πχ στην περίπτωσή μας, κάθε χρήστης μπορεί να λαμβάνει μηνύματα από συγκεκριμένα άτομα (φίλοι) και σε συγκεκριμένα κανάλια (text channels). Τέλος, οι clients μπορούνε να απεγγραφούν (unsubscribe) οποιαδήποτε στιγμή δεν θέλουν να λαμβάνουν ανανεώσεις στα δεδομένα τους. Αυτό βοηθάει στο να μην σπαταλούνται πόροι και στην μεριά του client και του server, καθώς δεν στέλνονται πλεονάζοντα δεδομένα.

2.4.4 Resolvers

Οι Resolvers είναι οι μέθοδοι που είναι υπεύθυνοι για να αντιστοιχίζουν τα δεδομένα στο κατάλληλο πεδίο ενός GraphQL ερωτήματος. Είναι κρίσιμης σημασίας στην GraphQL καθώς ορίζουν πως θα ανακτηθούν τα δεδομένα για κάθε πεδίο που ζητάει ο client. Κάθε πεδίο, μπορεί να έχει το δικό του Resolver και να εκτελείται μόνο όταν καλείται το συγκεκριμένο πεδίο. Στο επίπεδο ρίζας (root level) της GraphQL, υπάρχουν root resolvers όπου εμπεριέχουν τύπους ανώτερων επιπέδων. Δηλαδή λειτουργούν ως ομαδοποιήσεις από Queries, Mutations και Subscriptions, και συνήθως δημιουργούνται resolvers ρίζας για κάθε οντότητα στην βάση. Επίσης, έχουν πρόσβαση στο 'context' αντικείμενο όπου είναι κοινό για όλους τους resolvers. Αυτό το αντικείμενο χρησιμοποιείται για να αποθηκεύονται πληροφορίες που είναι χρήσιμες σε όλους τους resolvers, όπως για παράδειγμα authentication tokens, η σύνδεση στην βάση δεδομένων ή το αντικείμενο του χρήστη που εκτελεί το εκάστοτε request. Μπορούν να δεχτούν arguments τα οποία χρησιμοποιούνται για να προσαρμόσουν τα δεδομένα που θα ανακτηθούν. Κατά την ανάπτυξη λογισμικού ή το testing, μπορούν να προσφέρουν ψεύτικα δεδομένα χωρίς να ανακτούν από την βάση. Αυτό είναι χρήσιμο στην ανάπτυξη χαρακτηριστικών στο frontent, όπου δεν έχουν υλοποιηθεί ακόμη στο back-end.

2.4.5 Apollo GraphQL

Το Apollo GraphQL είναι ένα σύνολο εργαλείων, υπηρεσιών και βιβλιοθηκών για την ανάπτυξη και χρήση ενός GraphQL API. Προσφέρει ένα οικοσύστημα που απλοποιεί την ανάπτυξη και την διαχείριση GraphQL εφαρμογών. Μερικά από τα εργαλεία που προσφέρει είναι:

- **Apollo Server:** Ο Apollo Server υποστηρίζει την ανάπτυξη GraphQL APIs σε πολλές γλώσσες προγραμματισμού, περιλαμβάνοντας και την Node.js. Επιτρέπει στους προγραμματιστές να δηλώνουν το GraphQL schema και να υλοποιούν Resolvers για την διαχείριση και την ανάκτηση δεδομένων.
- **Apollo Client:** Ο Apollo client είναι μία JavaScript βιβλιοθήκη για την διαχείριση και χρήση ενός GraphQL server από τον client. Σκοπός του είναι να διευκολύνει την ανάκτηση, την επεξεργασία και την προσωρινή αποθήκευση δεδομένων από έναν GraphQL server. Μπορεί να χρησιμοποιηθεί από πολλά γνωστά front-end frameworks, όπως React, Angular και φυσικά Vue.js.
- **Apollo Devtools:** Το Apollo παρέχει επεκτάσεις και εργαλεία προγραμματιστή που μπορούν να χρησιμοποιηθούν στους πιο δημοφιλείς browsers. Αυτά τα εργαλεία, βοηθάνε στον εντοπισμό σφαλμάτων, στον έλεγχο και στην ανάλυση ενός GraphQL query.

2.4.6 Apollo Playground

Το Apollo Playground είναι ένα διαδραστικό web-based περιβάλλον, όπου επιτρέπει στους προγραμματιστές να εξερευνούν και να αλληλοεπιδρούν με ένα GraphQL server. Συχνά χρησιμοποιείται κατά την διαδικασία ανάπτυξης και testing για να πειραματιστούν με τα GraphQL queries, mutations και subscriptions. Προσφέρει έναν διαδραστικό συντάκτη κειμένου, όπου μπορεί κανείς να γράψει και να εκτελέσει GraphQL queries. Για την διευκόλυνση του προγραμματιστή, διαθέτει χαρακτηριστικά όπως, υπογράμμιση σύνταξης και σφαλμάτων καθώς και αυτόματη

συμπλήρωση. Επιτρέπει τους προγραμματιστές να περιηγηθούν σε ολόκληρο το GraphQL schema, δίνοντας τους πρόσβαση στους τύπους, τα πεδία και στο documentation. Αυτό βοηθάει στην κατανόηση ολόκληρης της δομής και των λειτουργιών ενός GraphQL API. Φυσικά, υποστηρίζει την χρήση μεταβλητών ερωτημάτων, ώστε να είναι δυνατή η παραμετροποίηση των queries και mutations, δίνοντας έτσι έναν εύκολο τρόπο δοκιμής των λειτουργιών του API. Τα αποτελέσματα των παραπάνω λειτουργιών, εμφανίζονται απευθείας μετά την εκτέλεση τους και έτσι οι προγραμματιστές έχουν γρήγορο έλεγχο και κατανοούν το response της κάθε λειτουργίας.

2.5 WebRTC

Η τεχνολογία του WebRTC, ή αλλιώς Web Real-Time Communication (Επικοινωνία Πραγματικού Χρόνου μέσω Δικτύου), είναι μία επαναστατική τεχνολογία του Web σε ό,τι αφορά τις επικοινωνίες. Δημιουργήθηκε και εκδόθηκε το 2011, από την Google[7] η οποία πρόσφατα είχε εξαγοράσει την Global IP Solutions (GIPS), μία εταιρία γνώστη για την τεχνολογία επεξεργασίας του ήχου και της φωνής. Η συγκεκριμένη τεχνολογία άλλαξε ριζικά τον τρόπο με τον οποίο λειτουργεί πλέον η επικοινωνία πραγματικού χρόνου μέσω του διαδικτύου.

Το WebRTC είναι μία τεχνολογία ανοιχτού κώδικα η οποία επιτρέπει στους περιηγητές ιστού και στις εφαρμογές τηλεφώνων να επικοινωνούν άμεσα χωρίς την ανάγκη ενός ενδιάμεσου εξυπηρετητή (με την μόνη εξαίρεση για την αποστολή σημάτων και λίγες εξαιρετικά σπάνιες περιπτώσεις). Επιτρέπει την μεταφορά δεδομένων μορφής ήχου, βίντεο αλλά και δεδομένων μεταξύ τους[7]. Η τεχνολογία προσφέρει πολύ ισχυρές δυνατότητες καθώς επιτρέπει την επικοινωνία μέσω των ίδιων των περιηγητών ιστού, χωρίς να υπάρχει κάποια ανάγκη για επιπρόσθετα λογισμικά τύπου plugin. Η ευκολία της προσβασιμότητας είναι πολύ σημαντικός παράγοντας για την ευρεία υιοθέτηση της.

Ο κύριος σκοπός του WebRTC είναι να οργανώσει και να διαχειριστεί την επικοινωνία πραγματικού χρόνου. Αυτό συμπεριλαμβάνει την κλήση μέσω φωνής και βίντεο καθώς και την μεταφορά αρχείων μέσω ενός ομότιμου δικτύου (peer-to-peer network) απευθείας μέσω του περιηγητή ιστού[7]. Η τεχνολογία υποστηρίζεται από τους μεγαλύτερους και επικρατέστερους περιηγητές ιστού όπως ο Chrome, ο Firefox, ο Safari και ο Edge κάνοντας το ευρέως προσβάσιμο. Το WebRTC έχει πλέον γίνει μία καίρια τεχνολογία για πολλές πλατφόρμες επικοινωνίας μέσω δικτύου, καθώς επιτρέπει την γρήγορη, αποτελεσματική και ασφαλή επικοινωνία.

Χρησιμοποιεί ένα σύνολο από κάποια γνωστά πρωτόκολλα και APIs για να επιτύχει την λειτουργία του. Συμπεριλαμβάνει πρωτόκολλα δικτύου, ήχου και βίντεο καθώς και APIs για την καταγραφή και μετάδοση ήχου, εικόνας και δεδομένων. Ένα από τα πολύ σημαντικά σημεία του WebRTC είναι η ικανότητα του να προσαρμόζεται σε διαφορετικές καταστάσεις του δικτύου, το οποίο εξασφαλίζει την σταθερή σύνδεση ακόμη και σε μη ιδανικά περιβάλλοντα. Συμπεριλαμβάνει επίσης πολλά μέτρα ασφαλείας, όπως για παράδειγμα την τέλος προς τέλος κρυπτογράφηση (end-to-end encryption), για να διαβεβαιώσει πως η επικοινωνία είναι ασφαλής και ιδιωτική.

Για την έναρξη της επικοινωνίας με το WebRTC είναι απαραίτητο να δημιουργηθεί μια χειραψία (handshake) μεταξύ των συμμετεχόντων για να γίνει η ανταλλαγή πληροφοριών που απαιτούνται για την κλήση (πχ IP διευθύνσεις και port numbers). Για αυτόν το λόγο υπάρχει η ανάγκη της λειτουργίας σηματοδότησης (signaling). Ουσιαστικά χρειάζεται μια επιπρόσθετη λειτουργία (ανεξάρτητα του WebRTC) η οποία μπορεί να χρησιμοποιεί οποιοδήποτε σύνολο από μεθόδους και πρωτόκολλα για να επιτευχθεί η σηματοδότηση της έναρξης επικοινωνίας. Μερικά παραδείγματα τέτοιων τεχνολογιών είναι το WebSocket (το οποίο χρησιμοποιείται και στην εφαρμογή μας), το XMPP ή ακόμη και τα mail. Μόλις ολοκληρωθεί η διαδικασία της σηματοδότησης μεταξύ των δύο πλευρών, τότε το WebRTC

χρησιμοποιεί το πρωτόκολλο της ICE (Interactive Connectivity Establishment) για να παρακάμψει τα προβλήματα ή περιορισμούς όπως το NAT (Network Address Translation) και τα τείχη προστασίας (firewalls). Το πρωτόκολλο αυτό είναι υπεύθυνο για την εδραίωση της peer-to-peer επικοινωνίας, διαλέγοντας με δυναμικό τρόπο την καλύτερη διαθέσιμη διαδρομή μέσα στο δίκτυο μεταξύ των συσκευών. Για να το επιτύχει χρησιμοποιεί μεθόδους όπως η STUN (Session Travesal Utilities) για να βρίσκει δημόσιες διευθύνσεις IP και η μέθοδος TURN (Travelsal Using Relays) ως δευτερεύουσα μέθοδο για να διαχειριστεί την επικοινωνία προς ένα εξυπηρετητή όταν μία σύνδεση peer-to-peer δεν είναι εφικτή[8].

Για την καταγραφή ήχου και βίντεο το WebRTC χρησιμοποιεί το API της βιβλιοθήκης MediaStream, χρησιμοποιώντας τα μικρόφωνα και τις κάμερες που είναι συνδεδεμένες ή ενσωματωμένες στην κάθε συσκευή. Το RTCDataChannel API χρησιμοποιείται για την μεταφορά αυτών των δεδομένων[8]. Οι ροές των συσκευών κωδικοποιούνται, συμπιέζονται και μεταφέρονται σε έναν ομότιμο (peer) καθώς το WebRTC υποστηρίζει πληθώρα κωδικοποιήσεων για βίντεο και ήχο.

2.6 WebSockets

Το WebSocket είναι ένα πρωτόκολλο επικοινωνίας μεταξύ υπολογιστικών συστημάτων όπου βοηθάει στην ταυτόχρονη αλληλεπίδραση μεταξύ τους μέσω του πρωτοκόλλου TCP (Transmission Control Protocol)[9]. Δημιουργεί πλήρης αμφίδρομες συνδέσεις μεταξύ μιας διαδικτυακής εφαρμογής και ενός διακομιστή, επιτρέποντας την επικοινωνία σε πραγματικό χρόνο (real time) βασισμένη σε συμβάντα (Events)[9]. Έτσι, ενεργοποιείται η δυνατότητα ανταλλαγής δεδομένων μεταξύ τους, οποιαδήποτε στιγμή, χωρίς την βοήθεια κάποιου HTTP request από τον client. Συνεπώς, μειώνει την αδράνεια που υπάρχει στο HTTP πρωτόκολλο, και έτσι καθίσταται ως ιδανικό σε εφαρμογές που έχει μεγάλη σημασία η ταχύτητα επικοινωνίας, όπως πχ, μία chatting εφαρμογή.

2.6.1 Socket.io

Το Socket.io είναι μία βιβλιοθήκη της Javascript όπου ενεργοποιεί την αμφίδρομη επικοινωνία μεταξύ client και server. Στην βάση του χρησιμοποιεί το WebSocket πρωτόκολλο και υλοποιεί κάποια επιπλέον χρήσιμα χαρακτηριστικά. Ένα από αυτά είναι η εναλλαγή (Fallback) που δίνει την δυνατότητα να χρησιμοποιηθεί διαφορετική μέθοδος ανταλλαγής δεδομένων (πχ long polling) στην περίπτωση που το WebSocket δεν είναι διαθέσιμο για χρήση, εξασφαλίζοντας την συμβατότητα σε διαφορετικά περιβάλλοντα δικτύου. Συνεπώς, αυτομάτως γίνεται συμβατό σε πολλά περισσότερα προγράμματα περιήγησης απ' ότι το WebSocket[10]. Άλλο σημαντικό χαρακτηριστικό του, είναι οι αυτόματοι μέθοδοι επανασύνδεσης σε περίπτωση που η σύνδεση διακοπεί[11]. Αυτό και άλλες αυτοματοποιημένες λύσεις, καθιστούν εύκολη την υλοποίηση εφαρμογών που το χρησιμοποιούν γλυτώνοντας αρκετό χρόνο.

2.6.2 Ανάγκη για Real Time data transfer στην εφαρμογή

Η φύση της εφαρμογής μας είναι τέτοια που επιβάλλει την ταυτόχρονη αμφίδρομη επικοινωνία μεταξύ clients και server. Ο χρήστης πρέπει να ενημερώνεται συνεχώς για τις αλλαγές που κάνουν οι υπόλοιποι χρήστες στο προφίλ τους, να λαμβάνει αιτήματα φιλίας και προσκλήσεις σε ομάδες (groups). Η σημαντικότερη, βέβαια, υλοποίηση του, είναι αυτή της ανταλλαγής μηνυμάτων με άλλους χρήστες, καθώς και η σηματοδότηση για την έναρξη μίας φωνητικής κλήσης μέσω του WebRTC. Σημαντικό ρόλο έχει και στην ανταλλαγή των τοποθεσιών στο δίκτυο (candidates) των χρηστών που θέλουν να εμπλακούν σε φωνητική κλήση, καθώς είναι απαραίτητα για την εκκίνηση της WebRTC σύνδεσης.

2.7 Vue.js

Ένας πιο μοντέρνος και αποτελεσματικός τρόπος για την ανάπτυξη του κώδικα στο κομμάτι του front-end αποτελούν στις μέρες μας τα front-end frameworks. Εκτενή πακέτα και βιβλιοθήκες που προφέρουν πληθώρα εργαλείων και δυνατοτήτων, με στόχο να διευκολύνουν την ανάπτυξη όσο και την μετέπειτα συντήρηση αλλά και επέκταση του κώδικα. Σε ένα τέτοιο framework είναι συνήθως προαπαιτούμενο να υπάρχουν βασικές γνώσεις στις γλώσσες και τεχνολογίες όπως HTML, JavaScript και CSS και αυτό κυρίως γιατί βασίζονται σε αυτές. Μερικά γνωστά frameworks είναι η React, η Angular και η Vue. Στην συγκεκριμένη εφαρμογή επιλέξαμε να υλοποιήσουμε το UI με το framework ανοιχτού κώδικα της Vue, και συγκεκριμένα με την 3η έκδοση (η οποία είναι και η νεότερη κατά την συγγραφή του κειμένου). Οι παραπάνω τεχνολογίες έχουν εξίσου ισχυρές δυνατότητες αλλά και αδυναμίες. Η Vue όντας η νεότερη εκ των τριών και με μια ισχυρή κοινότητα να την υποστηρίζει πλέον, αποτελεί μια πολύ καλή λύση για την εφαρμογή την οποία καλούμαστε να υλοποιήσουμε. Η υλοποίηση της ξεκίνησε το 2013 από τον Evan You ως ένα παράλληλο προσωπικό project του, ενώ ταυτόχρονα εργαζόταν για την Google. Ο σκοπός του ήταν να δημιουργήσει ένα καλύτερο framework από την Angular, το γνωστό framework της εταιρίας όπου εργαζόταν[12]. Η πρώτη έκδοση βγήκε τον Φεβρουάριο του 2014 με λίγες αλλά σημαντικές δυνατότητες. Ήταν ένα «ελαφρύ» framework το οποίο υποστήριζε την ανάδραση (reactivity) και ασχολούνταν με το UI στον μεγαλύτερο του βαθμό. Τον Σεπτέμβριο του 2016 κυκλοφόρησε και η $2^{η}$ έκδοση της Vue[12] με πολύ σημαντικές αλλαγές και μια μεγάλη γκάμα λειτουργιών που θα της επέτρεπε να χρησιμοποιηθεί και σε εταιρικό επίπεδο πλέον. Μετά την πολύ επιτυχημένη έκδοση 2 και την αρκετά μεγάλη σε αριθμό κοινότητα να την ακολουθεί, ακολούθησε και η έκδοση 3 τον Σεπτέμβριο του 2020[12], η οποία αφοσιώθηκε στην διόρθωση των λαθών, στην συνολική σωστή λειτουργία του framework και στην ευκολότερη συγγραφή του κώδικα. Σήμερα ανταγωνίζεται τα πιο γνωστά frameworks και χρησιμοποιείται σε ένα μεγάλο βαθμό στην ανάπτυξη των Web εφαρμογών.

2.7.1 Δομή της Vue

Η βασική μορφή σύνταξης κώδικα σε Vue αποτελείται πάντα από τρία μέρη: το template, το script και το style[13]. Ένα αρχείο Vue λοιπόν χωρίζεται σε 3 μέρη, το πρώτο είναι το template και το συγκεκριμένο τμήμα αφορά την σύνταξη του κώδικα που δημιουργεί το UI. Πιο συγκεκριμένα, χρησιμοποιεί την γνωστή DOM μορφή της HTML και ουσιαστικά είναι το κομμάτι το οποίο «εκτυπώνεται» (γίνεται το render) στον περιηγητή ιστού. Το τμήμα του script αποτελεί τον λειτουργικό κώδικα που τρέχει στις εφαρμογές αυτές και μπορεί να είναι γραμμένος είτε σε JavaScript, είτε σε TypeScript. Πλέον η Vue υποστηρίζει σε μεγάλο βαθμό την σύνταξη του script σε TypeScript. Σε αυτό το τμήμα του αρχείου εκτελούνται όλες οι λειτουργίες που απαιτεί η κάθε εφαρμογή και έχει μια ιδιαίτερη δομή ως προς την σύνταξη της και μπορεί να διαφέρει ελαφρώς ανάλογα και με την έκδοση της Vue. Στο τελικό κομμάτι style βρίσκονται οι μορφοποιήσεις των DOM στοιχείων και γενικότερα της εφαρμογής. Εδώ μπορεί κανείς να χρησιμοποιήσει CSS ή και άλλες τεχνολογίες που την επεκτείνουν, όπως SCSS και SASS. Έτσι λοιπόν σε ένα αρχείο μορφής ".vue" μπορεί κανείς να βρει μαζεμένη όλη την πληροφορία που χρειάζεται για ένα τμήμα κώδικα. Στις δύο πρώτες περιπτώσεις (template και script) η Vue είναι άρρηκτα συνδεδεμένη ως προς τον τρόπο που αναπτύσσεται ο κώδικας, σε αντίθεση με το style που ουσιαστικά αποτελεί απλά ένα τμήμα μορφοποίησης με CSS.

2.7.2 Directives

Ένας από τους λόγος που επιλέχθηκε η συγκεκριμένη τεχνολογία αποτελούν οι ενσωματωμένες λειτουργίες των ντιρεκτιβών (Vue Directives). Οι ντιρεκτίβες της Vue επιτρέπουν την εύκολη

διαχείριση των στοιχείων του DOM, με την χρήση απλών λέξεων όπως “v-if”, “v-for”, “v-bind” και “v-on”[13]. Η ντιρεκτίβα “v-model” είναι επίσης μια πολύ ισχυρή λέξη κλειδί για τα στοιχεία της HTML που δέχονται κάποια είσοδο δεδομένων από τον χρήστη. Σε μία φόρμα με διάφορα στοιχεία εισόδου (πχ: input, ή text-area) μπορεί κάποιος να αναθέσει ως γνώρισμα την ντιρεκτίβα v-model και να συνδέσει απευθείας το κάθε στοιχείο εισόδου με κάποια μεταβλητή του script που θα διατηρεί τα στοιχεία που εισάγει ο χρήστης, χωρίς όμως να χρειαστεί να γραφτεί κάποιος παραπάνω κώδικας για να ενημερωθεί η μεταβλητή αυτή. Έτσι, έχουμε την δυνατότητα για εύκολη και επαναλαμβανόμενη αναδόμηση, μορφοποίηση και απόδοση γνωρισμάτων αλλά και γεγονότων (event) σε HTML στοιχεία αλλά και δυνατότητες εισόδου δεδομένων από τον χρήστη χωρίς την άμεση παρέμβαση της JavaScript[13]. Οι ντιρεκτίβες αποτελούν ένα από τα σημαντικότερα κομμάτια της ανάπτυξης εφαρμογών με Vue, καθώς μπορούν να διευκολύνουν κατά ένα μεγάλο βαθμό την σύνταξη του κώδικα αφαιρώντας πολλές γραμμές που απαιτεί η JavaScript, κάνοντας τον μικρότερο και πιο ευανάγνωστο. Ως προς την χρήση τους, τοποθετούνται σαν γνωρίσματα στα στοιχεία της HTML ή στα components της Vue και λειτουργούν αυτόνομα και αναδραστικά.

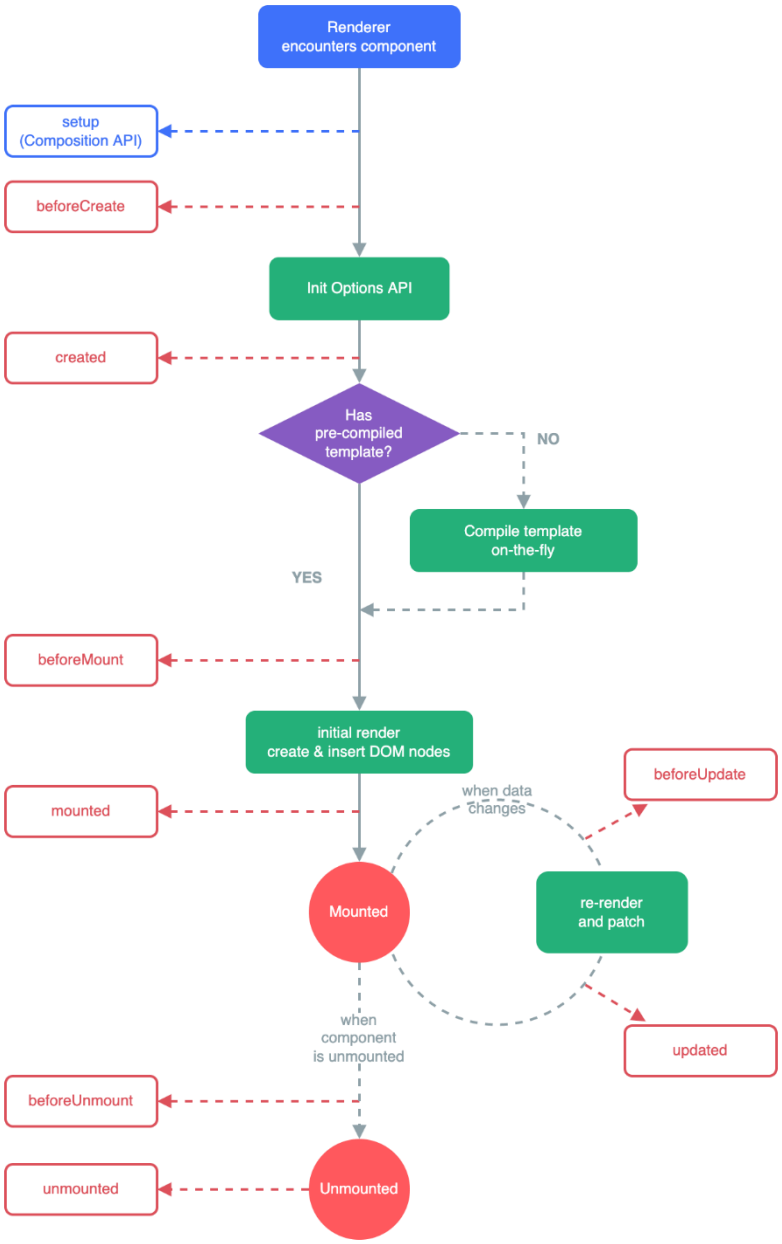
2.7.3 Components

Η δυνατότητα δημιουργίας επαναχρησιμοποιήσιμου κώδικα (component) είναι επίσης ένα βασικό χαρακτηριστικό του framework. Με τα components μπορεί κανείς να δημιουργήσει δυναμικά κομμάτια κώδικα μορφής Vue, τα οποία μπορούν να είναι ανεξάρτητα μεταξύ τους και το καθένα μπορεί να μορφοποιείται δυναμικά με την χρήση των props. Τα props αποτελούν μια εξελιγμένη μορφή των γνωρισμάτων (attributes) της HTML και μας επιτρέπουν να περνάμε δεδομένα στο κάθε component δυναμικά και με ανάδραση[13]. Το κάθε γνώρισμα μπορεί να διαφέρει σε κάθε περίπτωση και με αυτά το component μπορεί να μορφοποιηθεί διαφορετικά, να εκτελέσει διαφορετικές λειτουργίες καθώς επίσης και να εμφανίσει ή να «εξαφανίσει» πληροφορίες ανάλογα με τις πληροφορίες που λαμβάνει από αυτά. Η δυνατότητα αυτή είναι πολύ ισχυρή για το framework και αποτελεί ένα σημαντικό κομμάτι της ανάπτυξης και της επέκτασης της εφαρμογής χωρίς να δημιουργούνται επιπλοκές με τον υπάρχων κώδικα.

2.7.4 Lifecycle Hooks

Όπως αναφέρθηκε και παραπάνω η μορφή του script έχει μια συγκεκριμένη δομή στην Vue. Συνήθως χωρίζεται σε τμήματα όπως η δήλωση των component, η δήλωση των props κ.α. Το βασικού κομμάτι του script όμως βρίσκεται στο τμήμα του setup. Στο setup εκτελούνται όλες οι λειτουργίες κάθε component. Ένα setup έχει πρόσβαση στα σημεία του κύκλου ζωής της Vue στα οποία μπορούν να τρέξουν συναρτήσεις ή ολόκληρα block κώδικα. Μερικά από αυτά τα συχνά χρησιμοποιούμενα σημεία ζωής του κύκλου είναι: `beforeMounted`, `onMounted`, `onUpdated`, `onUnmounted`, `onBeforeMount`, `onActivated`, `onDeactivated` [13]κ.α. Όπως φαίνεται και στην Εικόνα 1 κάθε σημείο του κύκλου ζωής τρέχει σειριακά το ένα μετά το άλλο. Αφού ολοκληρωθούν οι διαδικασίες που τρέχουν στο παρασκήνιο της Vue σε κάθε ένα κόμματι του κύκλου, ακολουθούν τα τμήματα κώδικα που έχουν συνταχθεί από τον χρήστη. Η συγκεκριμένη λειτουργία θυμίζει αρκετά στον κύκλο ζωής του DOM της HTML, με την μόνη διαφορά ότι το κάθε component ακολουθεί τον δικό του κύκλο ξεχωριστά. Ένα αρχείο component λοιπόν, μπορεί να έχει ένα οποιοδήποτε σύνολο από αυτά τα σημεία του κύκλου ζωής, με διαφορετικές λειτουργίες να τρέχουν στο κάθε ένα, δημιουργώντας ένα προσωπικά διαμορφωμένο επαναχρησιμοποιήσιμο τμήμα κώδικα. Ένα σύνηθες παράδειγμα χρήσης είναι αυτό της `onMounted`, η οποία «πυροδοτείται» αφότου έχει ολοκληρωθεί η εκτύπωση του component στο DOM σχήμα της HTML, και συνηθώς μέσα σε αυτό συντάσσεται κάποιος κώδικας όπου αρχικοποιεί

μεταβλητές ή γίνεται κάποιο request σε ένα ή πολλά APIs για να ληφθούν πληροφορίες που απαιτεί το συγκεκριμένο component.



Εικόνα 1: Vue Component Lifecycle[13]

2.7.5 Options & Composition API

Όπως αναφέρθηκε και παραπάνω ανάλογα την έκδοση της Vue μπορεί να διαφέρει ελαφρώς και το πως συντάσσεται ο κώδικας. Στην συγκεκριμένη περίπτωση υπάρχει μια αρκετά σημαντική διαφορά στις εκδόσεις της Vue σε ότι αφορά τον κύκλο ζωής. Παραδείγματα κώδικα από την έκδοση 2 με την χρήση του Options API (επάνω μέρος) και της έκδοσης 3 με Composition API (κάτω μέρος) δίνονται παρακάτω στο Απόκομμα Κώδικα 1, τα οποία υλοποιούν τις ίδιες ακριβώς λειτουργίες απλώς με διαφορετική σύνταξη. Στην δική μας εφαρμογή έχοντας επιλέξει την 3η έκδοση της vue, αποφασίστηκε να υλοποιηθεί με την νεότερη μορφή σύνταξης, δηλαδή αυτή του Composition API η οποία εισήχθη με την έκδοση αυτή. Ωστόσο, αξίζει να σημειωθεί πως υπάρχει και η δυνατότητα σύνταξης και σε Options

API. Είναι σημαντικό κατά την εκκίνηση της ανάπτυξης ενός προγράμματος να εξετάζονται οι τελευταίες και σταθερές εκδόσεις των τεχνολογιών που πρόκειται να χρησιμοποιηθούν, με μία τάση να επιλέγεται η νεότερη για να μην υπάρχουν αργότερα θέματα λιγοστής υποστήριξης. Στην συγκεκριμένη περίπτωση είναι προφανές πως επωφελούμαστε και συντακτικά πέραν της υποστήριξης. Αυτό γιατί ο κώδικας στο Composition API απλοποιείται καθώς λιγοστεύουν οι γραμμές του, κάνοντας το πιο ευανάγνωστό.

```
<script>
export default {
  data() {
    return {
      name: '',
      age: 0,
      aboveAge:false
    }
  },
  methods: {
     verifyUser() {
     if(this.age < 18){
     this.aboveAge = false
    } else {
    this.aboveAge = true
      }
    },
  },
  mounted() {
    console.log('Application mounted');
  },
}
</script>
```

```
<script setup>
import {ref, reactive, onMounted } from 'vue'

const profile = reactive({name:'', age:''})
const aboveAge = ref(false)

const verifyUser = () => age.value < 18 ? aboveAge = false : aboveAge =
true

onMounted (() => console.log('Application mounted'))
</script>
```

Απόκομμα Κώδικα 1: Vue 2 με Options API και Vue 3 με Composition API[14]

2.7.6 Reactivity

Η λειτουργία κλειδί για να χρησιμοποιήσει κανείς την Vue σε αντίθεση με οποιαδήποτε άλλη τεχνολογία του βελησνεκούς της αποτελεί το reactivity, δηλαδή η αναδραστικότητα του που επιφέρει το framework. Η αναδραστικότητα αφορά το τρόπο με τον οποίο το UI μπορεί να ανανεώνει πληροφορίες ανάλογα με την ανάδραση του χρήστη ή ακόμα και αυτοματοποιημένα από τον κώδικα χρησιμοποιώντας τα σημεία του κύκλου ζωής[13], [14]. Η αναδραστικότητα μπορεί να αφορά όλη την εφαρμογή, κάποια σελίδα ή ακόμα και το κάθε component που δημιουργείται ξεχωριστά. Κάθε στοιχείο που δημιουργείται στην Vue λοιπόν μπορεί να υποστηρίξει την ανάδραση δίνοντας έτσι μια πολύ ισχυρή λειτουργία στον προγραμματιστή, ο οποίος σε μία απλή HTML με JavaScript εφαρμογή θα αναγκαζόταν να γράψει πολλές γραμμές κώδικα σε συνδυασμό με άλλες βιβλιοθήκες όπως η JQuery.

Για παράδειγμα, σε ένα component το οποίο εμφανίζει μια λίστα με χρήστες υπάρχει η δυνατότητα «προσθήκη νέου χρήστη». Με το reactivity της Vue η συγκεκριμένη λειτουργία είναι πολύ εύκολη να υλοποιηθεί με την χρήση των αναδραστικών μεταβλητών (ref values)[13]. Στο παράδειγμα μας οι χρήστες που εμφανίζονται στην λίστα του UI διατηρούνται σε έναν πίνακα της JavaScript με ενεργοποιημένη την αναδραστικότητα της Vue. Με την κλήση κάποιου γεγονότος από το UI (πχ: πάτημα ενός κουμπιού) προστίθεται ένας νέος χρήστης στον πίνακα αυτό. Αφού ολοκληρωθεί η προσθήκη, οι χρήστες θα ανανεωθούν αυτόματα στην λίστα του UI -στα σημεία που χρειάζεται μόνο- χωρίς να χρειάζεται κάποια προγραμματιστική παρέμβαση, λόγω της αναδραστικής μεταβλητής. Πιο συγκεκριμένα μια μεταβλητή αυτής της μορφής μπορεί να περάσει από το script κομμάτι στο template απευθείας και να γίνεται άμεσα η εκτύπωση της τιμής του αλλά και η αυτόματη ανανέωση του. Ένα απλό παράδειγμα μιας αναδραστικής λίστας που εκτυπώνεται από το script στο template δίνεται στο Απόκομμα Κώδικα 2. Στην συγκεκριμένη περίπτωση μπορεί κανείς να δει πως και το κάθε `<li>` στοιχείο έχει ένα γνώρισμα ref. Αυτό γιατί, πέραν της αναδραστικότητας βάσει μεταβλητών που διατηρούν τιμές, επιτρέπεται και η αναδραστικότητα με στοιχεία HTML. Μπορεί δηλαδή ένα στοιχείο της HTML να διατηρηθεί σε μία μεταβλητή της JavaScript απευθείας, και να ανανεώνεται κατευθείαν και να ενημερώνει και τις δύο πλευρές: script και template, αμφίδρομα. Με την συγκεκριμένη λειτουργία φεύγουν τελείως οι ανάγκες της JavaScript για την χρήση γεγονότων όπως `getElementById` ή `getElementByName`, καθώς κάθε στοιχείο του UI που χρειάζεται να διατηρηθεί σε μεταβλητή χρησιμοποιεί απλά το γνώρισμα ref και επικοινωνεί απευθείας με τον κώδικα.

```
<script setup>
import { ref, onMounted } from 'vue'

const list = ref([
 'list item 1',
 'list item 2',
 'list item 3',
 'list item 4',
])

const itemRefs = ref([])

onMounted(() => console.log(itemRefs.value))
</script>

<template>
 <ul>
  <li v-for="item in list" ref="itemRefs">
   {{ item }}
  </li>
 </ul>
</template>
```

Απόκομμα Κώδικα 2: Vue ref values

Ωστόσο η Vue προσφέρει μία ακόμη πολύ ισχυρή λειτουργία σε ότι αφορά το reactivity, τα computed values. Η λειτουργία αυτή επιτρέπει την δημιουργία μεταβλητών με παραγόμενες τιμές που εξαρτώνται από άλλες ref μεταβλητές. Όταν μία από αυτές τις εξαρτώμενες μεταβλητές αλλάξει τιμή, τότε η συνάρτηση της computed ξανά τρέχει και μπορεί να δώσει ένα νέο αποτέλεσμα κάθε φόρα[13], [14]. Η ιδιότητα αυτή βοηθάει ακόμη περισσότερο την αυτοματοποιημένη ανάδραση της εφαρμογής και βελτιώνει σε μεγάλο βαθμό τα προβλήματα του επαναλαμβανόμενου κώδικα. Οι μεταβλητές αυτού του τύπου δεν μπορούν να μεταβληθούν με ανάθεση τιμής, καθώς είναι readonly και η τιμή τους ορίζεται μόνο από την επιστρεφόμενη τιμή της δηλωμένης τους συνάρτησης. Παράδειγμα μίας computed

μεταβλητής δίνεται στο Απόκομμα Κώδικα 3. Στο συγκεκριμένο παράδειγμα η μεταβλητή `count` ορίζεται ως μία ref μεταβλητή που διατηρεί έναν αριθμό και η computed μεταβλητή `squared` δίνει πάντα την δύναμη της `count` τιμής. Μόλις για οποιονδήποτε λόγο η `count` αλλάξει τιμή, η `squared` θα τρέξει πάλι την συνάρτηση της και θα αποθηκεύσει το νέο αποτέλεσμα της σε αυτήν.

```
import { ref, computed } from 'vue';

const count = ref(1);

const squared = computed(() => count.value * count.value);

console.log(squared.value); // Output: 1

count.value = 2;
console.log(squared.value); // Output: 4
```

Απόκομμα Κώδικα 3: Computed values[13]

2.7.7 Styling

Η μορφοποίηση ενός UI είναι επίσης σημαντική να είναι εύκολη και ευέλικτη σε ότι αφορά την συγγραφή του κώδικα. Η Vue προσφέρει αρκετούς τρόπους για την ανάθεση CSS μορφοποιήσεων στο UI. Ο κλασικός τρόπος των inline styles και classes είναι ο πρώτος. Εδώ μπορεί κανείς να δώσει απευθείας σε ένα HTML στοιχείο, CSS κανόνες από τα γνωρίσματα style και class. Αυτό βέβαια είναι δυνατό να γίνει και με την απλή HTML. Στην Vue την μεγάλη διαφορά κάνει η χρήση των prop γνωρισμάτων v-style και v-class ή αλλιώς :style και :class για συντομία[13]. Σε αυτά μπορεί κανείς να αναθέσει είτε απευθείας μορφοποιήσεις ή ονόματα κλάσεων με την διαφορά πως μπορούν να μεταβάλλονται δυναμικά και μπορούν να έχουν αλφαριθμητικές τιμές, τιμές αντικειμένων είτε πινάκων. Συγκεκριμένα ένα inline style prop μπορεί να έχει αλφαριθμητικές τιμές (Απόκομμα Κώδικα 4) ή αντικείμενα τύπου CSS (Απόκομμα Κώδικα 5). Στη περίπτωση των κλάσεων δίνονται τιμές ονομάτων από κλάσεις με αλφαριθμητική μορφή ή αλλιώς με πίνακα που περιέχει ένα σύνολο από αλφαριθμητικά στοιχεία (Απόκομμα Κώδικα 6: Inline class με πίνακα). Σε όλα αυτά παίζει πολύ μεγάλο ρόλο και η αναδραστικότητα καθώς ενημερώνονται αυτόματα οι μορφοποιήσεις χωρίς μεγάλο προγραμματιστικό κόστος.

Μία άλλη δυνατότητα που προσφέρει η Vue, η οποία αναφέρθηκε προηγουμένως στο κεφάλαιο 2.7.1, είναι η χρήση του style τμήματος ενός Vue αρχείου. Σε αυτό μπορεί κανείς να δώσει CSS μορφοποιήσεις με τον γνωστό και κλασικό τρόπο, απλά με την επιπρόσθετη δυνατότητα προσθήκης του ορίσματος scoped. Στη περίπτωση που το style έχει ενεργοποιημένο το scoped όρισμα, τότε όλες οι αλλαγές θεωρούνται τοπικές και επηρεάζουν μονάχα ό,τι υπάρχει στο αρχείο του component. Έτσι λύνονται προβλήματα από κλάσεις της εφαρμογής που μπορεί να έχουν το ίδιο όνομα αλλά να χρήζουν διαφορετικές μορφοποιήσεις. Επιπρόσθετα, ένα αρχείο Vue μπορεί να έχει παραπάνω από ένα style τμήματα και το καθένα μπορεί να είναι σε διαφορετική γλώσσα (πχ: SCSS ή SASS) ή να έχει το όρισμα scoped.

```
<template>
 <div
  :style="'background-color: ' + backgroundColor"
 >
  This is a simple example with inline styles.
 </div>
</template>
```

```
<script setup>
const backgroundColor = ref('red')

</script>
```

Απόκομμα Κώδικα 4: Inline style με αλφαριθμητικές τιμές

```
<template>
 <div
  :style="styleObj"
 >
  This is a simple example with inline styles.
 </div>
</template>

<script setup>
const styleObj = ref({
 backgroundColor: 'red'
})

</script>
```

Απόκομμα Κώδικα 5: Inline style με αντικείμενο

```
<template>
 <div
  :class="[
   isOnline ? 'is-online' : 'is-offline',
   isSelected ? 'is-selected' : 'is-not-selected'
  ]"
 >
  This is a simple example with inline styles.
 </div>
</template>

<script setup>

</script>

<style>
.is-online {
 background-color: green;
}

.is-offline {
 background-color: red;
}

.is-selected {
 font-weight: 800;
}

.is-not-selected {
 font-weight: 400;
}
</style>
```

Απόκομμα Κώδικα 6: Inline class με πίνακα

2.7.8 Watchers

Οι watchers είναι μηχανισμοί που επιτρέπουν την παρακολούθηση αλλαγών σε μία ή πολλές ref μεταβλητές (είτε αυτή είναι ref μεταβλητή είτε computed) και την εκτέλεση συναρτήσεων σε μπλοκ κώδικα όταν υπάρξει κάποια αλλαγή στην παρακολουθούμενη μεταβλητή[13]. Ουσιαστικά εκτελείται όπως μία computed αναδραστική μεταβλητή, ωστόσο δεν περιορίζεται σε κάποιο αποτέλεσμα και μπορεί να μεταβάλει διάφορες μεταβλητές εσωτερικά. Ένα σύνηθες σενάριο που οι watchers μπορούν να φανούν χρήσιμοι είναι σε εφαρμογές με εγγραφές που ομαδοποιούν τα αποτελέσματα τους βάσει φίλτρων. Σε περίπτωση που κάποιο φίλτρο αλλάξει τιμή, τότε αν αυτή η μεταβλητή παρακολουθείται από έναν watcher, μπορεί να εκτελεσθεί ένα request στον server που ζητά τα νέα δεδομένα με τα επιλεγμένα φίλτρα του χρήστη. Το ίδιο ισχύει και στις εφαρμογές με σελιδοποίηση. Με κάθε αλλαγή στην μεταβλητή της σελιδοποίησης, το front-end μπορεί να ζητά νέα δεδομένα για τις επόμενες σελίδες.

2.7.9 Router

Ο router (δρομολογητής) είναι μέρος του οικοσυστήματος της Vue, και δημιουργήθηκε για εφαρμογές τύπου SPA (Single Page Applications, Εφαρμογές Μονής Σελίδας)[14]. Είναι ένα ισχυρό και ευέλικτο εργαλείο για την διαχείριση της δρομολόγησης στης εφαρμογές που έχουν υλοποιηθεί με την Vue. Είναι ενσωματωμένο ως επιπρόσθετο plugin και δουλεύει απρόσκοπτα με τον πυρήνα της Vue για την εναλλαγή των σελίδων-component εντός της εφαρμογής.

Η λειτουργία του είναι απλή, ο δρομολογητής ουσιαστικά αντιστοιχίζει διάφορα paths της σελίδας με κάποιο component. Κάθε διαδρομή (route) του δρομολογητή είναι σχετιζόμενο με ένα component, όταν ο σύνδεσμος URL του περιηγητή ιστού αλλάξει, ο δρομολογητής αυτομάτως ενημερώνει το τι εμφανίζεται στον χρήστη, ανάλογα με το τι έχει εισάγει στο path. Όλα τα παραπάνω ορίζονται σε ένα αρχείο που ρυθμίζει τον δρομολογητή και εκεί ορίζονται όλα τα paths και το σε ποιο component αντιστοιχίζονται.

Ένα από τα σημεία κλειδιά του δρομολογητή είναι η δυναμική δρομολόγηση. Αυτό σημαίνει πως υπάρχει η δυνατότητα προσθήκης παραμέτρων στην κάθε διαδρομή, τα οποία είναι τα δυναμικά τμήματα ενός URL. Για παράδειγμα σε μία σελίδα με προϊόντα για αγορά, η εφαρμογή μπορεί να διατηρεί κάποιες σελίδες προβολής του κάθε προϊόντος με πληροφορίες για αυτό. Κάθε προϊόν μπορεί να προβάλλεται χρησιμοποιώντας την ίδια σε σελίδα-component, απλά με κάποιο διαφορετικό αναγνωριστικό στο τέλος του URL, να εμφανίζονται οι πληροφορίες του καθενός ανάλογα με την τιμή που του δόθηκε. Ο δρομολογητής έχει την δυνατότητα να μετατρέπει αυτές τις δυναμικές παραμέτρους της διεύθυνσης σε props του component.

Άλλο ένα σημαντικό κομμάτι του δρομολογητή είναι τα guards (έλεγχοι προστασίας) των διαδρομών[13], [14]. Τα guards επιτρέπουν την εκτέλεση διάφορων ελέγχων προτού ο δρομολογητής μπει σε ένα path, είτε αφότου βγει. Παραδείγματα τέτοιων guard είναι ο έλεγχος εισόδου (αν ο χρήστης έχει κάνει login), ο έλεγχος ρόλου ή επιπέδου πρόσβασης ή αν κάποιο στοιχείο φορτίζει και δεν πρέπει να επιτραπεί η πρόσβαση προτού αυτό ολοκληρωθεί. Είναι πολύ σημαντικό αυτό χαρακτηριστικό καθώς επιτρέπει τον εύκολο έλεγχο πρόσβασης πριν ξεκινήσει ο κύκλος ζωής ενός component.

Υπάρχει επίσης η δυνατότητα των εμφωλευμένων διαδρομών, το οποίο επιτρέπει την δημιουργία πολύπλοκων UI με εμφωλευμένες σελίδες προβολής. Αυτό είναι χρήσιμο καθώς μεγάλες εφαρμογές μπορεί να αλλάζουν εντελώς ή τμηματικά το πως και τι εμφανίζονται ανάλογα το επιλεγμένο path

Ο δρομολογητής είναι πλήρως ενσωματωμένος στο αναδραστικό σύστημα της Vue. Έχει συγκεκριμένα μία μεταβλητή πλήρους εμβέλειας (global) την `$route`, την οποία μπορεί να χρησιμοποιήσουν όλα τα

component αναδραστικά. Πράγμα το οποίο σημαίνει, ότι όλες οι πληροφορίες της μεταβλητής ενημερώνονται αυτόματα παντού ανάλογα με το που βρίσκεται το path του URL. Κάποιες από τις πληροφορίες που διατηρεί η μεταβλητή είναι η τρέχουσα διαδρομή, το όνομα της διαδρομής και οι παράμετροι (props) αν αυτοί υπάρχουν. Επιπρόσθετα η μεταβλητή αυτή επιτρέπει την διαχείριση της πλοήγησης, δηλαδή είναι δυνατό να αλλάξει η τρέχουσα επιλεγμένη διαδρομή προγραμματιστικά αλλάζοντας και την προβαλλόμενη σελίδα χωρίς να απαιτείται κάποια είσοδος ή ανάδραση από τον χρήστη.

2.7.10 Pinia (State Management Store)

Η Pinia είναι η πλέον επίσημη βιβλιοθήκη για την προσωρινή αποθήκευση δεδομένων, η όποια παρουσίασε μια πιο ευέλικτη και απλούστερη λύση από την προηγούμενη εναλλακτική του VueX που υπήρχε για τις παλαιότερες εκδώσεις της Vue. Με την έκδοση 3 της Vue, η Pinia έγινε διάσημη λόγω της εύκολης χρήσης, της βελτιωμένης υποστήριξης της TypeScript καθώς και της συμβατότητας με την νέα δομή της Vue 3 με το Composition API[15].

Η βιβλιοθήκη επιτρέπει την δημιουργία stores (αποθηκών) τα οποία είναι “κοντέινερ” για κοινή χρήση από όλα τα components της εφαρμογής. Αυτά τα store είναι σχεδιασμένα με τέτοιο τρόπο ώστε να συνυπάρχουν με την αναδραστικότητα της Vue. Έτσι όλες οι πληροφορίες που αλλάζουν σε ένα κεντρικό σημείο (του store) με την αναδραστικότητα της Vue ανανεώνονται αυτόματα σε όλα τα components που τα χρησιμοποιούν. Όλες οι δυνατότητες του store ισχύουν όσο ο χρήστης βρίσκεται στην εφαρμογή και αν αυτή κλείσει ή γίνει επαναφόρτιση της σελίδας (refresh) όλα τα δεδομένα χάνονται.

Υπάρχουν κάποια πλεονέκτημα του Pinia store που το καθιστούν καλύτερο από αυτό του VueX. Σημαντικότερο όλων η απλότητα και ευκολία της δημιουργίας τους. Κατά την δημιουργία ενός Pinia store απλά καλείται μία συνάρτηση `defineStore` στην οποία δίνεται ή βασική δομή των δεδομένων (state), οι λειτουργίες (actions) και οι συναρτήσεις επιστροφής (getters)[15]. Οι λειτουργίες είναι ουσιαστικά οι μέθοδοι που αλλάζουν την δομή του state, ενώ οι συναρτήσεις επιστροφής είναι ουσιαστικά computed μεταβλητές που εξάγουν αποτελέσματα βάσει του state.

2.8 Quasar Framework & WindiCSS

2.8.1 Quasar

Για την υλοποίηση του front-end χρησιμοποιήθηκε η UI βιβλιοθήκη Quasar. Η συγκεκριμένη βιβλιοθήκη είναι ένα πολύ βοηθητικό εργαλείο για την εκκίνηση ενός Vue project, και αυτό γιατί προσφέρει ένα πακέτο από τεχνολογίες, λειτουργίες και έτοιμα components τα οποία ακολουθούν την μορφή του Material Design. Βασίζεται άρρηκτα στην Vue πράγμα που σημαίνει πως είναι υλοποιημένη για την συγκεκριμένη τεχνολογία και καλύπτει πολλές από της ανάγκες της αυτόματα χωρίς κάποια προσθήκη από τον προγραμματιστή. Επιτρέπει την λειτουργία πολλαπλής πλατφόρμας, δηλαδή μία υλοποίηση της εφαρμογής μπορεί πολύ εύκολα να μετατραπεί από Web εφαρμογή, σε εφαρμογή Android, iOS ή και desktop[16]. Προσφέρει λειτουργίες για μετατροπή του UI βάσει του μεγέθους της οθόνης (responsive), καθώς και λειτουργίες για dark και light theming[16]. Προσφέρει έτοιμα components υλοποιημένα σε Vue όπου το καθένα συμπεριλαμβάνει διάφορες λειτουργίες πολύ χρήσιμες για τον προγραμματιστή. Πχ: τα input components της βιβλιοθήκης έρχονται με έτοιμες λειτουργίες input validation (ελέγχους εισόδου), με λειτουργίες debounce που επιτρέπουν την ακύρωση εισόδου μετά από ένα χρονικό διάστημα, με διάφορα έτοιμα styles για το κάθε ένα κ.α. Το σύνολο των λειτουργιών αυτών βοηθάει στους χρόνους ανάπτυξης, δίνοντας την ευκολία στην κάθε εφαρμογή να

αναπτύξει περισσότερο τα πιο εξειδικευμένα θέματα της, παρά τις συχνότερα χρησιμοποιούμενες λειτουργίες που συνήθως υλοποιούνται κατ' επανάληψη σε κάθε εφαρμογή. Επίσης προσφέρει ένα μικρό σύνολο από CSS κλάσεις για μορφοποίηση, τα οποία είναι πολύ βασικά πράγματα όπως το flex-box, κάποιες χρωματικές παλέτες κλπ.

2.8.2 WindiCSS (Windi)

Ενώ η Quasar προσφέρει κάποια μικρή βοήθεια σε ότι αφορά την μορφοποίηση των component σε θέματα CSS, χρειάζεται και μία μικρή ενίσχυση. Την συγκεκριμένη ενίσχυση προσφέρει η Windi βιβλιοθήκη (ή αλλιώς WindiCSS), η οποία παρέχει ένα πολύ μεγάλο σύνολο από κλάσεις οι οποίες δίνουν μία απευθείας μορφοποίηση στις ετικέτες ή τα components που τα χρησιμοποιούν. Σκοπός της είναι να διαμορφώνει τα δεδομένα και τα στοιχεία ενός UI, κατά βάσει με την χρήση CSS κλάσεων, εξαλείφοντας την σύνταξη CSS style στον μεγαλύτερο βαθμό και κάνοντας τον κώδικα μικρότερο αλλά και πιο ευανάγνωστο. Η Windi πηγάζει την έμπνευση της από μια πολύ μεγάλη βιβλιοθήκη ίδιας λογικής, την Tailwind[17]. Η διαφορά τους έχει να κάνει κυρίως με του χρόνους εκτέλεσης των CSS καθώς και λειτουργίες που επιτρέπουν στον προγραμματιστή να επηρεάσει και να δοκιμάσει κλάσεις κατά το run time του κώδικα[17]. Η μορφή μιας κλάσης της Windi (ή της Tailwind) μοιάζει ως εξής

```
bg-red-500
```

Απόκομμα Κώδικα 7: Χρήση κλάσεων στην WindiCSS

Όπως φαίνεται και παραπάνω η συγκεκριμένη κλάση αποτελείται από 3 μέρη: την λέξη "bg" που ορίζει το CSS γνώρισμα "background και το υπόλοιπο τμήμα αφορά την απόδοση τιμής στο γνώρισμα αυτό, συγκεκριμένα εδώ χρησιμοποιείται μία από τις ήδη υπάρχουσες μεταβλητές της βιβλιοθήκης το "red-500", το οποίο είναι η 5^{η} απόχρωση κόκκινου που διαθέτει (100 έως 900). Ωστόσο μπορεί κανείς να δώσει και δικές του αποθηκευμένες τιμές στην Windi και να τις χρησιμοποιεί με τον ίδιο ή και με πιο απλό τρόπο. Πχ:

```
bg-myRed
```

Απόκομμα Κώδικα 8: Χρήση μεταβλητών στην WindiCSS

Αν στην μεταβλητή "myRed" της Windi έχει οριστεί χρώμα, τότε το χρώμα φόντου του στοιχείου που το χρησιμοποιεί θα πάρει την τιμή του αποθηκευμένου χρώματος. Επιπλέον υπάρχει και η δυνατότητα δυναμικής απόδοσης τιμών χωρίς την χρήση μεταβλητών. Αυτό γίνεται με την χρήση αγκυλών πχ:

```
bg-[rgb(100, 20, 15)] ή bg-[#aa13bc]
```

Απόκομμα Κώδικα 9: Χρήση δυναμικής τιμής σε κλάσεις της WindiCSS

Με αυτούς τους τρόπους είναι πολύ εύκολο κανείς να αποφύγει πλήρως την σύνταξη style σε μεγάλο βαθμό και να κάνει πολύ πιο γρήγορα τις αλλαγές του καθώς κάποιες κλάσεις με μία μόνο λέξη ορίζουν ένα σύνολο μορφοποιήσεων CSS πχ:

truncate	overflow: hidden; text-overflow: ellipsis; white-space: nowrap;

Απόκομμα Κώδικα 10: Σύγκριση Windi με CSS

Όπως φαίνεται και στο Απόκομμα Κώδικα 10, στα αριστερά βρίσκεται η κλάση της βιβλιοθήκης που χρησιμοποιείται για την αποκοπή λέξεων σε περίπτωση που χρειάζεται αναδίπλωση γραμμάτων κάποιο στοιχείο και στα δεξιά βρίσκεται η υλοποίηση γίνεται με CSS.

Επίσης προσφέρει και ένα σύνολο από ψευδοκλάσεις τις οποίες μπορεί κανείς να χρησιμοποιήσει συνδυαστικά στα components. Πχ αν θέλουμε ένα στοιχείο να αλλάζει χρώμα όταν ο κέρσορας βρίσκεται από πάνω του αλλά και όταν κάνει κάποιος click σε αυτό τότε θα μπορούσε να χρησιμοποιήσει το παρακάτω σύνολο των κλάσεων:

```
bg-blue-500 hover:bg-myHoverBlue focus:bg-[#12af22]
```

Απόκομμα Κώδικα 11: Χρήση ψεδοκλάσεων στην WindiCSS

Η 1η κλάση δίνει το χρώμα blue-500 στο στοιχείο, η 2η δίνει το χρώμα που έχει αποθηκευτεί στην μεταβλητή myHoverBlue όσο ο κέρσορας βρίσκεται πάνω του, ενώ η 3η αποδίδει τον δεκαεξαδικό χρωματισμό “#12af22” όσο πραγματοποιείται το κλικ του κέρσορα σε αυτό.

Σε περίπτωση που μία κλάση θέλουμε να της δοθεί η λέξη κλειδί “!important” που δίνει προτεραιότητα σε περίπτωση που έχουμε επικαλυπτόμενες κλάσεις τότε απλά προσθέτουμε ένα θαυμαστικό μπροστά από την κλάση δηλαδή:

```
bg-red-500 !bg-blue-800
```

Απόκομμα Κώδικα 12: Χρήση "!important" στην WindiCSS

Στην παραπάνω περίπτωση (αν και λάθος συντακτικά) η κλάση bg-blue-500 θα είναι αυτή που θα αποδώσει το χρώμα της στο στοιχείο που την χρησιμοποιεί.

2.9 Βάσεις Δεδομένων

Σε αυτό το υπο-κεφάλαιο θα αναλύσουμε κάποια χαρακτηριστικά και τύπους βάσεων δεδομένων, τι βάση επιλέχτηκε για την εφαρμογή και πως την διαχειριζόμαστε βάσει των αναγκών μας.

2.9.1 MySQL

Η υλοποίηση της MySQL ξεκίνησε από το Monty Widenius το 1979 και αρχικός της σκοπός, ήταν η δημιουργία ενός εργαλείου αναφορών γραμμένο σε BASIC για την εταιρεία που ονομαζόταν TcX. Το 1995, ο ίδιος και δύο συνεργάτες Allan Larsson και Axmark, ίδρυσαν την εταιρεία MySQL AB στην Σουηδία, κυκλοφορόντας την πρώτη έκδοση της MySQL ως σχεσιακή βάση δεδομένων και λογισμικό ανοιχτού κώδικα[18]. Αρχικά, είχαν σκοπό να υλοποιηθεί με το σύστημα δεδομένων mSQL αλλά στην πορεία επέλεξαν να αναπτύξουν ένα δικό τους σύστημα χαμηλού επιπέδου (ISAM), ώστε να επιλύσουν προβλήματα επίδοσης που προκύπταν με την χρήση του mSQL[19]. Η εύκολη χρήση της σε συνδυασμό με τον ανοιχτό κώδικα και την ανθεκτικότητα της, την κατέστησαν ως μία από η πιο διαδεδομένη βάση δεδομένων καθώς είναι γνωστή για την ταχύτητά, την επεκτασιμότητα και την αξιοπιστία της. Αποθηκεύει τα δεδομένα σε προκαθορισμένους πίνακες με συγκεκριμένη δομή το οποίο βοηθάει στην αποτελεσματική διαχείριση των δεδομένων. Η ακεραιότητα των δεδομένων, προκύπτει από διάφορους περιορισμούς, όπως για παράδειγμα τα primary και foreign keys καθώς και από τους τύπους των δεδομένων[20]. Επομένως, ο λόγος που επιλέχθηκε ως λύση για την εφαρμογή μας, είναι η εύκολη και δωρεάν εγκατάσταση, η ταχύτητα, καθώς η φύση της εφαρμογής απαιτεί ανάκτηση δεδομένων μέσα από μεγάλο όγκο δεδομένων, και τέλος, η αξιοπιστία της.

2.9.2 Διαχείρηση βάσης Mikro-ORM

Το ORMs (Object Relational Mapping) είναι μία τεχνική προγραμματισμού όπου αντιστοιχίζει τα δεδομένα μιας βάσης δεδομένων, σε μορφή αντικειμένων (Objects) στον κώδικα και ξεκίνησε από τα τέλη των 80s. Πριν την υιοθέτηση του, οι προγραμματιστές έπρεπε να γράφουν αυτοσχέδιο κώδικα SQL όπου προκύπταν προβλήματα όπως συντακτικά λάθη και δύσκολη συντηρησιμότητα. Ξεκίνησε να γίνεται δημοφιλές στις αρχές των 00s, με υλοποιήσεις όπως το Hibernate για την Java και το Entity Framework για την .NET όπου απλοποιούσαν την αλληλεπίδραση του κώδικα με την βάση δεδομένων. Υπάρχουν διάφορες εκδοχές του, όπως το Data-Mapper όπου ξεχωρίζει την βάση από το τα αντικείμενα bussiness logic, δίνοντας την δυνατότητα αλλαγής βάσης δεδομένων χρησιμοποιώντας ακριβώς τον ίδιο κώδικα, και το Active Record όπου η δομή της βάσης δεδομένων είναι αλληλένδετη με τα αντικείμενα όπου καθιστά δυσκολότερη την μετάβαση σε άλλη βάση[21]. Συχνά κρίνονται ως μη αποδοτική μέθοδος, καθώς κατηγορούνται για μη αποδοτική δομή SQL ερωτημάτων με αποτέλεσμα την μείωση της εκτέλεσης τους. Παρά την κακή τους φήμη επιδόσεων, συνεχίζουν να χρησιμοποιούνται ευρέως για την ανάπτυξη λογισμικού, καθώς κερδίζεται πολύς χρόνος κατά το ανάπτυξη[22].

Αρχικά, τα ORMs (Object Relational Mapper) έχουν την δυνατότητα να σχεδιάσουν την δομή της βάσης δεδομένων, αντιστοιχίζοντας τα πεδία ενός Object/Class, σε στήλες ενός πίνακα. Επομένως, όταν για παράδειγμα κάνεις εισαγωγή δεδομένων στην βάση, τοποθετεί στην κατάλληλη στήλη της βάσης, το αντίστοιχο πεδίο του Object, διευκολύνοντας έτσι στην διαχείριση των δεδομένων. Αντίστοιχα, κατά την ανάκτηση δεδομένα από την βάση, αντιστοιχίζει τις στήλες του πίνακα κατευθείαν στα πεδία του Object, και χωρίς περαιτέρω επεξεργασία, είναι έτοιμα για χρήση. Επίσης, μέσω έτοιμων κανόνων και συναρτήσεων, μπορούν να δημιουργήσουν οποιοδήποτε ερώτημα επιθυμείς για την βάση.

Το Mikro-ORM λοιπόν, είναι μια βιβλιοθήκη ORM της NodeJS, όπου χρησιμοποιεί την μέθοδο Active Record και κατατάσσεται στην κατηγορία Typescript ORM, καθώς υποστηρίζεται πλήρως από την Typescript. Ρόλος της στην εφαρμογή είναι η διαχείριση της βάσης δεδομένων (MySQL), από τον σχηματισμό της μέχρι και την ανάκτηση/επεξεργασία των δεδομένων. Το ισχυρό της σημείο, είναι η διαδικασία δημιουργίας ερωτημάτων στην βάση, καθώς με απλούς τους απλούς κανόνες που διαθέτει, λόγω της αντιστοιχίας Object-DB_table, δίνεται η δυνατότητα να δημιουργήσεις περίπλοκα ερωτήματα με μεγαλύτερη ασφάλεια, καθώς το ερώτημα παράγεται αυτόματα. Επομένως, βοηθάει στην ταχύτητα του development καθώς δεν χρειάζεται να ανησυχείς για τυχόν συντακτικά λάθη στα query strings. Επίσης, βοηθάει στην διατήρηση του κώδικά. Πχ, όταν θέλεις να κάνεις μετονομασία μία στήλη ενός πίνακα, το μόνο που χρειάζεται να κάνεις, είναι να μετονομάσεις το πεδίο του Object και στην συνέχεια να δημιουργήσεις, μέσω του CLI του Mikro-ORM, το αρχείο (Migration) που περιέχει το SQL ερώτημα που θα μετονομάσει την στήλη. Επίσης, λόγω της συσχέτισης με το Object, τα ήδη υπάρχον ερωτήματα θα εκτελεστούν κανονικά, χωρίς να κάνεις κάποια αλλαγή.

Παραδείγματα παραγωγής ερωτημάτων:

1. Η γραμμή κώδικα: const users = await em.find(User, { name: { $like: '%Tilemachos%' } }, { orderBy: { name: 'DESC' } }) θα παράξει το εξής ερώτημα: SELECT * FROM user WHERE name LIKE '%Tilemachos%' ORDER BY name DESC και θα αποθηκεύσει το αποτέλεσμα στην μεταβλητή users.
2. Η γραμμή κώδικα: const user = await em.findOneOrFail(User, { id: 1 }) θα παράξει το ερώτημα SELECT * FROM user WHERE id = 1 LIMIT 1 και θα αποθηκεύσει το αποτέλεσμα στην μεταβλητή user. Αν δεν βρεθεί κάποιο αποτέλεσμα, το Mikro-ORM θα ρίξει 404 error.

Φυσικά, υπάρχουν πολλές ακόμα επιλογές ώστε να παραχθεί ερώτημα για κάθε περίπτωση, πχ ομαδοποίηση (group by), επιλογή συγκεκριμένων πεδίων από τον πίνακα, εύκολη ανάκτηση των relations κ.α.

2.10 Version Control System (VCS)

Το Version Control είναι μια πρακτική για να παρακολουθείς και να διαχειρίζεσαι το πηγαίο κώδικα ενός λογισμικού και για άλλα ψηφιακά αρχεία. Είναι θεμελιώδης έννοια στον τομέα ανάπτυξης λογισμικού και πολύ σημαντικό για την ανάπτυξη του από πολλούς προγραμματιστές ταυτόχρονα. Αποθηκεύουν το ιστορικό σε εκδόσεις, δίνοντας την δυνατότητα ανάκτησης προηγούμενης έκδοσης όταν αυτή χρειάζεται[23]. Τα οφέλη του είναι αρκετά, όπως η εύκολη διατήρηση και η σταθερότητα του κώδικα ανεξαρτήτως αλλαγών, η δυνατότητα συνεργασίας μεταξύ πολλών μελών μιας ομάδας, διαχείριση των αλλαγών οδηγώντας στην καλύτερη διαχείριση του έργου και ευθυνών[24]. Υπάρχουν δύο είδη Version Control (πχ Subversion), τα Centralized (πχ Git) και τα Distributed. Τα Centralized συστήματα, έχουν αποθηκευμένο τον κώδικα σε μια "κεντρική" αποθήκη, και οι προγραμματιστές κάνουν αλλαγές κατευθείαν εκεί και αν το μοναδικό αντίγραφο καταστραφεί δεν μπορούν να γυρίσουν σε προηγούμενη έκδοση. Τα θετικά τους είναι η απλή τους λογική στην χρήση και η εύκολη διαχείριση τους από τους διαχειριστές. Τα Destributed συστήματα, έχουν μεγαλύτερη ασφάλεια, καθώς ο κάθε ένας προγραμματιστής, κάνει αντίγραφο του έργου στον υπολογιστή του και έτσι, αν καταστραφεί ο πηγαίος κώδικας, η ομάδα μπορεί να χρησιμοποιήσει το αντίγραφο ενός προγραμματιστή μειώνοντας έτσι το ρίσκο για **χάσιμο** ολόκληρου του ιστορικού. Επίσης, λόγω τον τοπικών αντιγράφων, οι προγραμματιστές μπορούν να δημιουργήσουν εκδόσεις χωρίς να επηρεάζουν απευθείας τον κεντρικό κώδικα, επιτρέποντάς τους να είναι ευέλικτοι χωρίς να δημιουργούν μεγάλο όγκο αλλαγών ανά έκδοση[25].

2.10.1 Git

Το Git ξεκίνησε το 2005 και έκτοτε είναι το πιο δημοφιλές VCS. Αυτό που το έκανε τόσο δημοφιλές, είναι ο εξαιρετικός χειρισμός των διακλαδώσεων (branching) και συγχώνευσης (merging). Το branching επιτρέπει στους προγραμματιστές να αναπτύσσουν παράλληλα πολλά features και να διορθώνουν σφάλματα, δημιουργώντας για το καθένα ένα καινούριο κλαδί (branch), χωρίς να επηρεάζει τον βασικό κώδικα. Επίσης, η εναλλαγή μεταξύ των branch είναι αρκετά εύκολη, βοηθώντας έτσι στην ευελιξία των προγραμματιστών, δίνοντάς τους την δυνατότητα να επικεντρώνονται σε ζητήματα που προκύπτουν και έχουν προτεραιότητα. Το merging είναι η διαδικασία που ενώνει τις αλλαγές από ένα branch σε ένα άλλο. Πρόκειται για μια αυτοματοποιημένη διαδικασία μειώνοντας έτσι το ρίσκο ανθρώπινου λάθους. Ωστόσο, κατά την διαδικασία της συγχώνευσης, μπορεί να προκύψουν συγκρούσεις (conflicts) όπου χρειάζεται ο ανθρώπινος παράγοντας για την επίλυσή τους πριν την ολοκληρωθεί η συγχώνευση[26].

2.11 Json Web Token (JWT)

Το JWT είναι βασισμένο στην JSON μορφή δεδομένων, όπου είναι μία ελαφριά μορφή ανταλλαγής δεδομένων και σε συνδυασμό με την απλότητα του και την εύκολη ανάγνωση του, αποτελεί την πιο δημοφιλής μορφή[27]. Το JWT αποτελείται από τρία κομμάτια, την επικεφαλίδα (header), το φορτίο δεδομένων (payload) και την υπογραφή του ιδιωτικού κλειδιού[27]. Χρησιμοποιείται ευρέως για είναι να καλύψει την ανάγκη για ασφαλή ταυτοποίηση του χρήστη, και αυτό το επιτυγχάνει δημιουργώντας ένα token υπογραφόμενο από το ιδιωτικό κλειδί του server, η οποία υπογραφή υποδηλώνει την γνησιότητα του payload[27]. Το payload μπορεί να διαβαστεί από τον οποιοδήποτε, χωρίς αποκωδικοποίηση με κάποιο κλειδί. Συνεπώς, δεν πρέπει να εμπεριέχονται ευαίσθητα δεδομένα όπως,

κωδικός πρόσβασης, ευαίσθητες προσωπικές πληροφορίες του χρήση κ.α. Επίσης, κατά την δημιουργία του, του δίνεται χρονοσήμανση λήξης και μόλις αυτή περάσει, το token θεωρείται αναξιόπιστο και πρέπει να ανανεωθεί. Δίνεται η δυνατότητα χρήσης του μετά το πέρας την χρονοσήμανση λήξης, δεν ενδείκνυται όμως, καθώς το token μπορεί να βρεθεί σε κακόβουλα χέρια. Επομένως, η καλύτερη μέθοδος είναι η τακτική λήξη και ανανέωση του.

Στην εφαρμογή χρησιμοποιήθηκε ως ασφαλής αλληλεπίδραση του UI με το back-end. Κατά την σύνδεση του χρήστη, παράγεται το JWT με σύντομη διάρκεια ισχύος και ένα refresh token το οποίο αποθηκεύεται στον client με αρκετά μεγαλύτερο χρόνο ισχύος. Το back-end ελέγχει αν εμπεριέχεται το JWT του χρήστη σε κάθε request, και στην συνέχεια επιτρέπει την εκτέλεση του, αφού ελέγξει ότι έχει υπογραφεί από το ίδιο, δεν έχει λήξει και ότι είναι νεότερο κλειδί που έχει ζητήσει ο χρήστης. Για την επίτευξη του τελευταίου ελέγχου, στην εγγραφή του χρήστη στην βάση, σώζεται η χρονοσήμανση λήξης του τελευταίου JWT που παράχθηκε από τον χρήστη. Για την ανανέωση του token χρησιμοποιείται ειδικό request όπου εμπεριέχει το refresh token και επιστρέφει το νέο JWT και το νέο refresh token.

2.12 PM2

Το PM2 είναι ένας ευρέως γνωστός διαχειριστής διαδικασίας (process manager) για Node.js εφαρμογές. Σχεδιάστηκε για να απλοποιήσει την διαδικασία έκδοσης μιας εφαρμογής και να διευκολύνει την εκκίνηση τους σε περιβάλλον παραγωγής (production environment). Διαθέτει λειτουργίες CLI που διαχειρίζεται τον κύκλο ζωή των Node.js applications, όπως την διαδικασία της εκκίνησης, της παύσης και επανεκκίνησης. Διασφαλίζει ότι η διαδικασία θα τρέχει στο παρασκήνιο και επανεκκινώντας την σε περίπτωση σφάλματος. Παρέχει εργαλεία που διευκολύνουν την παρακολούθηση της κατάστασης, αρχείων καταγραφής και των πόρων που χρειάζεται για την εκτέλεση της, όπως η χρήση του επεξεργαστή, της μνήμης κ.α. Επιτρέπει την αναβάθμιση της εφαρμογής, χωρίς να διακόπτεται λειτουργία του και χωρίς να γίνει αντιληπτό από τους χρήστες. Αυτό το επιτυγχάνει εκκινώντας καινούρια αναβαθμισμένη διαδικασία με τα καινούρια χαρακτηριστικά, ανακατευθύνει την κίνηση από την παλιά στην καινούρια μόλις η δεύτερη είναι έτοιμη, και τέλος, τερματίζει την παλιά. Κάθε διαδικασία μπορεί να χρησιμοποιεί της δικές του παραμέτρους περιβάλλοντος, χωρίς να επηρεάζεται από τις άλλες, διευκολύνοντας την διαχείριση διάφορων περιβαλλόντων, όπως δοκιμής, παραγωγής και ανάπτυξης. Συγκεντρώνει όλα τα logs της εφαρμογής σε ειδικά αρχεία, είτε εξόδου είτε σφαλμάτων, διευκολύνοντας την ανάλυση και την αντιμετώπιση τυχών σφαλμάτων. Τέλος, μερικές εντολές CLI που μας βοήθησαν κατά την ανάπτυξη της εφαρμογής:

- **pm2 start:** Αφορά την εκκίνηση της διαδικασίας
- **pm2 stop:** Αφορά την παύση της διαδικασίας
- **pm2 restart:** Αφορά την επαννεκίνηση της διαδικασίας μετά την αναβάθμιση του λογισμικού
- **pm2 logs:** Εκτυπώνει σε πραγματικό χρόνο το output της εφαρμογής βελτιώνοντας την κατανόηση της διαχείρισης των δεδομένων και την αντιμετώπιση σφαλμάτων
- **pm2 list:** Εκτυπώνει έναν πίνακα με τις διαδικασίες που έχουν ξεκινήσει τουλάχιστον μία φορά, δίνοντας πληροφορίες όπως, την κατάσταση της διαδικασίας, την χρήση μνήμης και το ποσοστό του επεξεργαστή
- **pm2 show {process_id}:** Προσφέρει χρήσιμες πληροφορίες για μια συγκεκριμένη διαδικασία όπως την τοποθεσία των αρχείων που αποθηκεύονται τα logs της εφαρμογής.

Κεφάλαιο 3ο: Σχεδίαση της Βάσης Δεδομένων

Η βάση δεδομένων αποτελεί σημαντικό παράγοντα στην εφαρμογή. Σκοπός της είναι να φιλοξενεί και να επεξεργάζεται όλα τα δεδομένα των χρηστών, όπως τις πληροφορίες του προφίλ του, τα μηνύματα που ανταλλάσσει με άλλους χρήστες, τα Group που συμμετέχει κ.α. Σχεδιάστηκε με προσοχή, ώστε να καθιστά εύκολη την κάθε ανάγκη για παραμετροποιήσεις της στο μέλλον, και δόθηκε ιδιαίτερη σημασία στον σχεδιασμό ως προς την αποδοτικότητα των ερωτημάτων που πρόκειται να εξυπηρετήσει. Σε αυτό το κεφάλαιο λοιπόν, θα δούμε όλους τους πίνακες της βάσης δεδομένων που χρησιμοποιήθηκε, καθώς και τα πεδία τους και που αυτά αποσκοπούν.

3.1 IDs των πινάκων

Πριν προχωρήσουμε στην περιγραφή των πινάκων, θα πρέπει να αναφερθεί πως σχεδόν όλοι, περιέχουν ένα αλφαριθμητικό πεδίο id, όπου πρόκειται για το πρωτεύων κλειδί (primary key) κάθε εγγραφής. Οπότε είναι το κλειδί που συσχετίζει την σχέση (relation) με τις υπόλοιπες οντότητες (entities) της βάσης. Το id είναι τύπου varchar(255) και αποθηκεύει ένα UUIDv4 το οποίο παράγεται από την βιβλιοθήκη uuid της Node.js. Το UUIDv4 ουσιαστικά παράγει ένα μοναδικό αλφαριθμητικό id, μεγέθους 36 χαρακτήρων και όγκου 128-bit. Τα 4 bits αφορούν την έκδοση του UUID και τα 2 αφορούν την παραλλαγή του υπολογισμού. Τα υπόλοιπα 122 bits υπολογίζονται τυχαία με τον συνδυασμό των εξής μεθόδων:

- **Χρονοσήμανση:** Καθώς το UUIDv4 βασίζεται κατά κύριο λόγο στην τυχαιότητα, ένα κομμάτι του 60 bits αποτελείται από μία χρονοσήμανση (timestamp). Αυτό υπολογίζεται κάθε 100 νανοδευτερόλεπτα με σημείο εκκίνησης το Γρηγοριανό ημερολόγιο από το 1582.
- **Ακολουθία Ρολογιού:** Περιέχει ένα κομμάτι 14 bits, όπου βοηθάει στην αποφυγή δημιουργίας διπλότυπων ids σε περίπτωση που δημιουργηθούν πολλά κλειδιά κατά την διάρκεια των 100 νανοδευτερολέπτων.
- **Αναγνωριστικό κόμβου:** Τα τελευταία 48 bits ονομάζεται πεδίο κόμβου (Node field) και αντιπροσωπεύει την διεύθυνση του υλικού (hardware), δηλαδή περιλαμβάνει την διεύθυνση MAC της συσκευής που παράγει το uuid. Αυτό αυξάνει την μοναδικότητα ακόμη και όταν παράγονται κλειδιά ακριβώς την ίδια στιγμή από δύο διαφορετικές συσκευές.

3.2 mikro_orm_migrations Table

Ο πίνακας mikro_orm_migrations περιέχει όλα τα εκτελεσμένα migration αρχεία που περιέχονται στο back-end. Κάθε γραμμή του περιγράφει και ένα migration αρχείο ώστε το Mikro-ORM να γνωρίζει από ποιο αρχείο και μετά θα συνεχίσει να εκτελεί τα migrations, όταν αυτό του ζητηθεί. Τα πεδία του είναι:

- **id:** Είναι τύπου INT και είναι το PRIMARY KEY και ποθηκεύει ένα αύξον αριθμό σε κάθε νέα γραμμή του πίνακα. Είναι ο μόνος πίνακας στην βάση που περιέχει id και δεν είναι τύπου UUIDv4.
- **name:** Είναι τύπου VARCHAR(255) και αποθηκεύει το όνομα του αρχείου που εκτελέστηκε για την τροποποίηση στην βάση.
- **executed_at:** Είναι τύπου DATETIME και αποθηκεύει την ημερομηνία και ώρα που εκτελέστηκε το αρχείο.

3.3 user Table

Ο πίνακας user αποσκοπεί στην αποθήκευση και επεξεργασία όλων των προσωπικών πληροφοριών του κάθε χρήστη. Τα πεδία που περιλαμβάνει είναι τα εξής:

- **displayName:** Είναι τύπου VARCHAR(255) και είναι το όνομα που ο χρήστης επιθυμεί να εμφανίζεται στου υπόλοιπους χρήστες.
- **username & code:** Το username είναι τύπου VARCHAR(255) και είναι το όνομα χρήστη του κάθε user, και ο κώδικας (code) VARCHAR(255) είναι ένας τετραψήφιος αριθμός. Ο συνδυασμός των πεδίων username και code είναι μοναδικός. Αυτό βοηθάει στην αναζήτηση μεταξύ χρηστών, χρησιμοποιώντας ένα από τα δύο πεδία ως βάση στην αναζήτηση τους, και το δεύτερο πεδίο ως συμπληρωματικό. Έτσι επιτρέπεται ένα username να χρησιμοποιηθεί 9999 φορές, δίνοντας την δυνατότητα στους χρήστες να χρησιμοποιούν το ίδιο username χωρίς να κατοχυρώνεται από έναν.
- **email:** Είναι τύπου VARCHAR(255) UNIQUE, δηλαδή το κάθε email μπορεί να δημιουργήσει μόνο έναν λογαριασμό στην εφαρμογή.
- **password:** Είναι τύπου VARCHAR(255) και αποτελεί τον κωδικό πρόσβασης του χρήστη στην εφαρμογή. Αποθηκεύεται κρυπτογραφημένος σε μορφή hash.
- **confirmEmailToken:** Είναι τύπου VARCHAR(255) και αποθηκεύει ένα JWT που παράγεται κατά την δημιουργία του λογαριασμού του χρήστη, και αποστέλλεται στο email του για να προχωρήσει στην επιβεβαίωση του.
- **resetPasswordToken:** Είναι τύπου VARCHAR(255) και είναι της ίδια λογικής με το confirmEmailToken, συμβάλλοντας όμως, στην επανέκδοση του κωδικού πρόσβασης του χρήστη.
- **emailConfirm:** Είναι τύπου TINYINT (boolean) και σκοπός του είναι να πάρει θετική τιμή μόλις επιβεβαιωθεί το email του χρήστη, ώστε να αποκτήσει πρόσβαση στην εφαρμογή.
- **isLoggedIn:** Είναι τύπου TINYINT και παίρνει τιμή 1 όσο ο χρήστης είναι συνδεδεμένος στην εφαρμογή και τιμή 0 όταν είναι ανενεργός.
- **lastLoggedInDate:** Είναι τύπου DATETIME και αποσκοπεί στην αποθήκευση της τελευταίας ημερομηνίας όπου ο χρήστης ήταν συνδεδεμένος στην εφαρμογή.
- **icon:** Είναι τύπου VARCHAR(255) και αποθηκεύει το url της φωτογραφίας προφίλ του χρήστη.
- **createdAt:** Είναι τύπου DATETIME και αποθηκεύει την ημερομηνίας και ώρα εγγραφής του χρήστη στην εφαρμογή.
- **preferences:** Είναι τύπου TEXT και αποθηκεύει διάφορες προσωπικες ρυθμίσεις που αφορούν την εφαρμογή, όπως την ένταση του output μιας κλήσης, την ένταση του μικροφώνου του χρήστη κ.α, σε ένα JSON object. Ο λόγος οι ρυθμίσεις αποθηκεύονται με αυτόν τον τρόπο και όχι ξεχωριστά σε κάποιον πίνακα ή και σε πεδία του πίνακα User, είναι πως το JSON είναι αρκετά abstract, επομένως προσφέρει ευκολία στην αφαίρεση και προσθήκη νέων ρυθμίσεων.
- **connectedVoiceChannel:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY όπου είναι το id του πίνακα VoiceChannel. Η σχέση τους είναι many to one και σκοπός του είναι η αποθήκευση του καναλιού κλήσης ενός group που είναι συνδεδεμένος ο χρήστης.

Αυτά είναι τα πεδία που βρίσκονται στον πίνακα User. Βέβαια, ο πίνακας έχει πολλές σχέσεις one to many και many to many, όπου θα αναλυθούν στην πορεία με την περιγραφή των επόμενων πινάκων.

3.4 friend_request Table

Ο πίνακας friend_request αποσκοπεί στην αποθήκευση όλων των αιτημάτων φιλίας μεταξύ των χρηστών. Σκοπός είναι να εμφανίζονται τα αναπάντητα αιτήματα στον χρήστη ώστε να μπορεί να τα αποδεχτεί ή τα απορρίψει. Αυτό επιτυγχάνεται με την χρήση των παρακάτω πεδίων:

- **message:** Είναι τύπου VARCHAR(255) και σκοπός του είναι να αποθηκεύσει ένα μήνυμα συνοδεύει το αίτημα φιλίας.
- **from_user_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY όπου είναι το id του πίνακα user. Αποθηκεύει το id του χρήστη που έστειλε το αίτημα φιλίας.
- **to_user_id:** Και αυτό είναι τύπου VARCHAR(255) και είναι FOREIGN KEY όπου είναι το id του πίνακα user. Αποθηκεύει το id του χρήστη που λαμβάνει το αίτημα φιλίας.

- **answer:** Είναι τύπου TINYINT NULL και δείχνει την απάντηση του χρήστη που έλαβε το αίτημα φιλίας. Όταν είναι null, το αίτημα φιλίας δεν είχε κάποια απάντηση. Όταν αποθηκεύει τιμή 0, ο χρήστης έχει απορρίψει το αίτημα, ενώ όταν είναι 1 το έχει δεχτεί.
- **canceled:** Είναι τύπου TINYINT NULL και δείχνει αν ο αποστολέας του αιτήματος, τελικά το ακύρωσε. Όταν η τιμή είναι 1 σημαίνει πως το αίτημα είναι ακυρωμένο και σταματάει να εφμανίζεται στον χρήστη.
- **createdAt:** Είναι τύπου DATETIME και αποθηκεύει την ημερομηνία και ώρα της δημιουργίας του αιτήματος φιλίας.
- **updatedAt:** Είναι τύπου DATETIME και αποθηκεύει την ημερομηνία και ώρα της απάντησης του παραλλήπτη, ή την ημερομηνία ακύρωσης από τον αποστολέα.

3.5 user_friend_ list Table

Ο πίνακας friend_list είναι υπεύθυνος για την αποθήκευση όλων των φίλων του κάθε χρήστη. Ουσιαστικά είναι ένας πίνακας pivot της many to many σχέσης από τον πίνακα user στον πίνακα user. Τα πεδία του είναι:

- **user_1_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY όπου είναι το id του πίνακα user. Περιέχει το έναν από τους δύο χρήστες που είναι φίλοι μεταξύ τους.
- **user_2_id:** Το ίδιο με το user_1_id αλλά αυτό περιέχει τον δεύτερο user.

3.6 personal_chat Table

Ο πίνακας personal_chat είναι ο χώρος που δύο φίλοι ανταλλάσσουν προσωπικά μηνύματα μεταξύ δύο ή και περισσότερων χρηστών. Τα πεδία του είναι:

- **is_group_personal_chat:** Είναι τύπου TINYINT και δηλώνει αν το personal_chat είναι ομαδικό όταν η τιμή είναι 1.
- **disabled:** Είναι τύπου TINYINT και δηλώνει αν ένα personal_chat έχει μπλοκαριστεί από κάποιον χρήστη όταν η τιμή είναι 1.
- **mute:** Είναι τύπου TINYINT και δηλώνει αν ένα personal_chat έχει μπει σε σίγαση από κάποιον χρήστη.

Καθώς μερικά από αυτά τα features δεν υλοποιήθηκαν στην εφαρμογή, γίνεται τώρα αντιληπτό, πως το πεδία disabled και mute δεν θα έπρεπε να ήταν σε αυτόν τον πίνακα αλλά στον αμέσως επόμενο, personal_chat_user_pivot. Ο λόγος είναι, πως αυτές οι λειτουργίες αυτήν την στιγμή επιρεάζουν το personal_chat συνολικά και όχι κατά την προτίμηση του κάθε χρήστη ξεχωριστά.

3.7 personal_chat_user_pivot Table

Το personal_chat_user_pivot περιγράφει την many to many σχέση μεταξύ των πινάκων user και personal_chat. Επομένως σκοπός του είναι να αποθηκεύει, ποιοι χρήστες έχουν πρόσβαση στα personal chats. Τα πεδία του είναι:

- **personal_chat_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα personal_chat.
- **user_pivot:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα user.
- **last_read_message:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα personal_message που θα δούμε στην πορεία. Ουσιαστικά είναι το τελευταίο μήνυμα που διάβασε ο κάθε χρήστης στο συγκεκριμένο personal chat.

Όπως ειπώθηκε και στην περιγραφή του πίνακα personal_chat, σε αυτόν τον πίνακα θα έπρεπε να φιλοξενούνται και τα πεδία disabled και mute, καθώς αυτός ο πίνακας περιέχει την σχέση του κάθε user με το κάθε personal chat.

3.8 personal_message Table

Ο πίνακας personal_message είναι υπεύθυνος για την αποθήκευση όλων των πληροφοριών ενός μηνύματος που στέλνεται από κάποιον χρήστη, σε ένα personal chat. Επίσης, έχει την ιδιότητα να περιγράφει αν ένα μήνυμα είναι τύπου κλήσης, όπου περιέχει χαρακτηριστικά που αφορούν τις κλήσης. Ουσιαστικά είναι ένας εύκολος τρόπος να τυπώνουμε στους χρήστες τις κλήσεις που πραγματοποιήθηκαν σε ένα personal chat. Τα πεδία του είναι:

- **createdAt:** Είναι τύπου DATETIME και περιγράφει την ημερομηνία και ώρα που έστειλε κάποιος χρήστης το μήνυμα. Ο ρόλος του είναι αρκετά σημαντικός, καθώς η ταξινόμηση των μηνυμάτων ενώς personal chat γίνεται με αυτό το πεδίο.
- **text:** Είναι τύπου TEXT και αποθηκεύει το κείμενο που περιέχει το μήνυμα.
- **deleted:** Είναι τύπου TINYINT και σκοπός του είναι να περιγράφει αν ένα μήνυμα έχει διαγραφεί από τον αποστολέα.
- **state:** Είναι τύπου ENUM(‘delete_for_me’, ‘delete_for_all’). Αποθηκεύει δηλαδή την κατάσταση ενός μηνύματος. Όταν περιέχει την τιμή ‘delete_for_me’, σημαίνει πως ο αποστολέας του μηνύματος θέλει να σβήσει το μήνυμα για τον εαυτό του, ώστε να μην φαίνεται στον λογαριασμό του. Όταν περιέχει την τιμή ‘delete_for_all, σημαίνει πως το μήνυμα διαγράφηκε για όλους του χρήστες του personal_chat.
- **file:** Είναι τύπου VARCHAR(255) NULL και σκοπός του είναι να αποθηκεύει το url ενός αρχείου που μπορεί να περιλαμβάνεται στο μήνυμα.
- **from_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα user. Σκοπός του είναι να αποθηκεύει τον αποστολέα του μηνύματος.
- **personal_chat_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα personal_chat. Αποθηκεύει τον δέκτη personal chat που έστειλε ο αποστολέας το μήνυμα.
- **is_call:** Είναι τύπου TINYINT με default τιμή 0. Οταν η τιμή είναι 1, τότε περιγράφει πως το μήνυμα πρόκειται για πειγραφή κλήσης.
- **call_data:** Είναι τύπου TEXT και αποθηκεύει JSON Object που περιέχει πληροφορίες σε περίπτωση που το μήνυμα περιγράφει μια κλήση, όπως το πόσο χρόνο διήρκησε μία κλήση ή πόση ώρα καλούσε ο χρήστης. Τα πεδία που αποθηκεύει είναι:
    - **call_answer:** Είναι τύπου TINYINT NULL και αποθηκεύει την απάντηση του παραλήπτη μιας κλήσης. Όταν η τιμή είναι NULL σημαίνει πως ο παραλήπτης δεν απάντησε στην κλήση. Όταν είναι 1 σημαίνει πως απάντησε θετικά, ενώ όταν είναι 0 σημαίνει πως απέρριψε την κλήση.
    - **call_start_timestamp:** Είναι τύπου DATETIME NULL και αποθηκεύει την ημερομηνία και ώρα αποδοχής της κλήσης. Αν η κλήση δεν έγινε αποδεκτή, η τιμή παραμένει NULL.
    - **call_end_timestamp:** Είναι τύπου DATETIME NULL και αποθηκεύει την ημερομηνία που η κλήση τερματίστηκε.
    - **call_end_calling_timestamp:** Είναι τύπου DATETIME NULL και αποθηκεύει την ημερομηνία και ώρα που ο χρήστης που καλεί σταμάτησε την κλήση ή ο παραλλήπτης την απέρριψε.

3.9 personal_message_deleted_from_user Table

Ο πίνακας personal_message_deleted_from_user έχει σκοπό να αποθηκεύει τα μηνύματα που ο χρήστης διέγραψε, αλλά δεν είναι δικά του. Δηλαδή τα διαγράφει μόνο για τον εαυτό του. Ουσιαστικά είναι ένας πίνακας pivot για να περιγράψει την many to many σχέση μεταξύ users και διαγεγραμμένα personal_messages. Τα πεδία του είναι:

- **personal_message_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα personal_message. Περιγράφει δηλαδή το διαγεγραμμένο μήνυμα.
- **user_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα user. Περιγράφει τον user που διέγραψε το μήνυμα για τον εαυτό του.

3.10 group Table

Ο πίνακας group αποθηκεύει τις ομαδικούς χώρους όπου φιλοξενούν διάφορα κανάλια κειμένου και ήχου και χρήστες που έχουν προσκληθεί να είναι μέλη τους. Τα πεδία του είναι:

- **name:** Είναι τύπου VARCHAR(255) και δηλώνει το όνομα του group που εμφανίζεται στα μέλη του.
- **icon:** Είναι τύπου VARCHAR(255) και αποθηκεύει το URL της εικόνας που χρησιμοποιείται για avatar στο group.
- **color:** Είναι τύπου VARCHAR(255) και αποθηκεύει το χρώμα του group όπου χρησιμοποιείται στο UI για να δώσει κάποιες ξεχωριστές λεπτομέρειες σε κάθε group.
- **owner_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα user και αποθηκεύει τον χρήστη που δημιούργησε το group.
- **created_at:** Είναι τύπου DATETIME και αποθηκεύει την ημερομηνία και ώρα της δημιουργίας του.
- **preferences:** Είναι τύπου TEXT και αποθηκεύει ένα JSON Object που περιέχει πεδία για διάφορες ρυθμίσεις του group, όπως να επιτρέπει τα μέλη του να στέλνουν προσκλήσεις σε άλλους users.

3.11 group_invite Table

Ο πίνακας group_invite είναι υπεύθυνος για την αποθήκευση των προσκλήσεων, που στέλνουν οι χρήστες σε φίλους τους για να συμμετέχουν σε κάποιο group. Τα πεδία του είναι ακριβώς τα ίδια με τον πίνακα friend_request, με ένα επιπλέον πεδίο:

- **group_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα group και περιγράφει το group που προσκαλείται να συμμετέχει ο παραλήπτης.

3.12 group_members Table

Ο πίνακας group_members είναι ο pivot πίνακας για να περιγράψει ποιοι χρήστες έχουν λάβει μέρος σε ποια groups. Επομένως δίνει πρόσβαση στους χρήστες να χρησιμοποιούν τα κανάλια ήχου κειμένου ενός group. Τα πεδία του είναι:

- **group_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα group περιγράφει το group που προσκαλείται να συμμετέχει ο παραλήπτης.
- **user_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα user και περιγράφει τον χρήστη που είναι μέλος στο συγκεκριμένο group.

3.13 group_invitation_permission_users Table

Ο πίνακας group_invitation_permission_users είναι ένας pivot πίνακας μεταξύ groups και users και σκοπός του είναι να αποθηκεύει τα μέλη του group τα οποία έχουν πρόσβαση στην αποστολή προσκλήσεων σε άλλους χρήστες. Τα πεδία του είναι:

- **group_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα group περιγράφει το group που ένα μέλος μπορεί να στείλει προσκλήσεις.
- **user_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα user και περιγράφει το μέλος του group που μπορεί να στείλει προσκλήσεις.

3.14 text_channel Table

Ο πίνακας text_channel περιγράφει ένα κανάλι κειμένου, όπου μπορεί να αξιοποιηθεί από τα μέλη ενός group για να στέλνουν μηνύματα που είναι προσβάσιμα από όλα τα μέλη του group. Τα πεδία του είναι:

- **name:** Είναι τύπου VARCHAR(255) και αποθηκεύει το όνομα ενός καναλιού κειμένου.

- **slow_mode:** Είναι τύπου INT NULL και περιγράφει αν το κανάλι κειμένου επιτρέπει στα μέλη να στέλνουν απανωτά μηνύματα. Όταν η τιμή του είναι NULL, τα μέλη δεν έχουν κάποιο περιορισμό, ενώ αν περιέχει τιμή, τα μέλη μπορούν να στείλουν μηνύματα ανά το διάστημα δευτερολέπτων που έχει οριστεί. Η τιμή μετριέται σε χιλιοστά δευτερολέπτου.
- **group_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα group και περιγράφει το group που ανήκει το text_channel.

3.15 text_channel_message Table

Ο πίνακας text_channel_message αποθηκεύει όλα τα μηνύματα που στέλνονται από τα μέλη ενός group, σε κάποιο text_channel. Τα πεδία του είναι ακριβώς τα ίδια με το personal_message χωρίς τα πεδία call_data και personal_chat_id. Παρόλα αυτά το personal_chat_id αντικαθίσταται από το πεδίο text_channel_id όπου είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα text_channel και περιγράφει το κανάλι κειμένου που έστειλε το μέλος το μήνυμα.

3.16 text_channel_user_pivot Table

Ο πίνακας text_channel_user_pivot είναι ο pivot πίνακας της many to many σχέσης μεταξύ users και text_channesls, και περιέχει τις ρυθμίσεις και άλλα δεδομένα που σχετίζονται με το μέλος του group και το text_channel. Τα πεδία του είναι:

- **text_channel_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα text_channel και περιγράφει το text_channel της σχέσης με τον user.
- **user_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα user και αποθηκεύει το μέλος που επιρεάζει τα υπόλοιπα πεδία.
- **last_read_message_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα text_channel_message και αποθηκεύει το τελευταίο μήνυμα που διάβασε το μέλος του group.
- **mute:** Είναι τύπου TINYINT DEFAULT 0 και η τιμή του παραμετροποιείται από τον χρήστη όταν δεν θέλει να λαμβάνει ειδοποιήσεις για τα μηνύματα που αφορούν το συγκεκριμένο text_channel. Όταν η τιμή είναι 0, λαμβάνει κανονικά ειδοποιήσεις, ενώ όταν είναι 1 σταματάει να ειδοποιείται.

Σε αντίθεση με την λειτουργία του mute του personal_chat, εδώ χρησιμοποιήθηκε σωστά και μπορεί ο κάθε χρήστης να το παραμετροποιήσει για τον εαυτό του.

3.17 text_channel_message_deleted_from_user Table

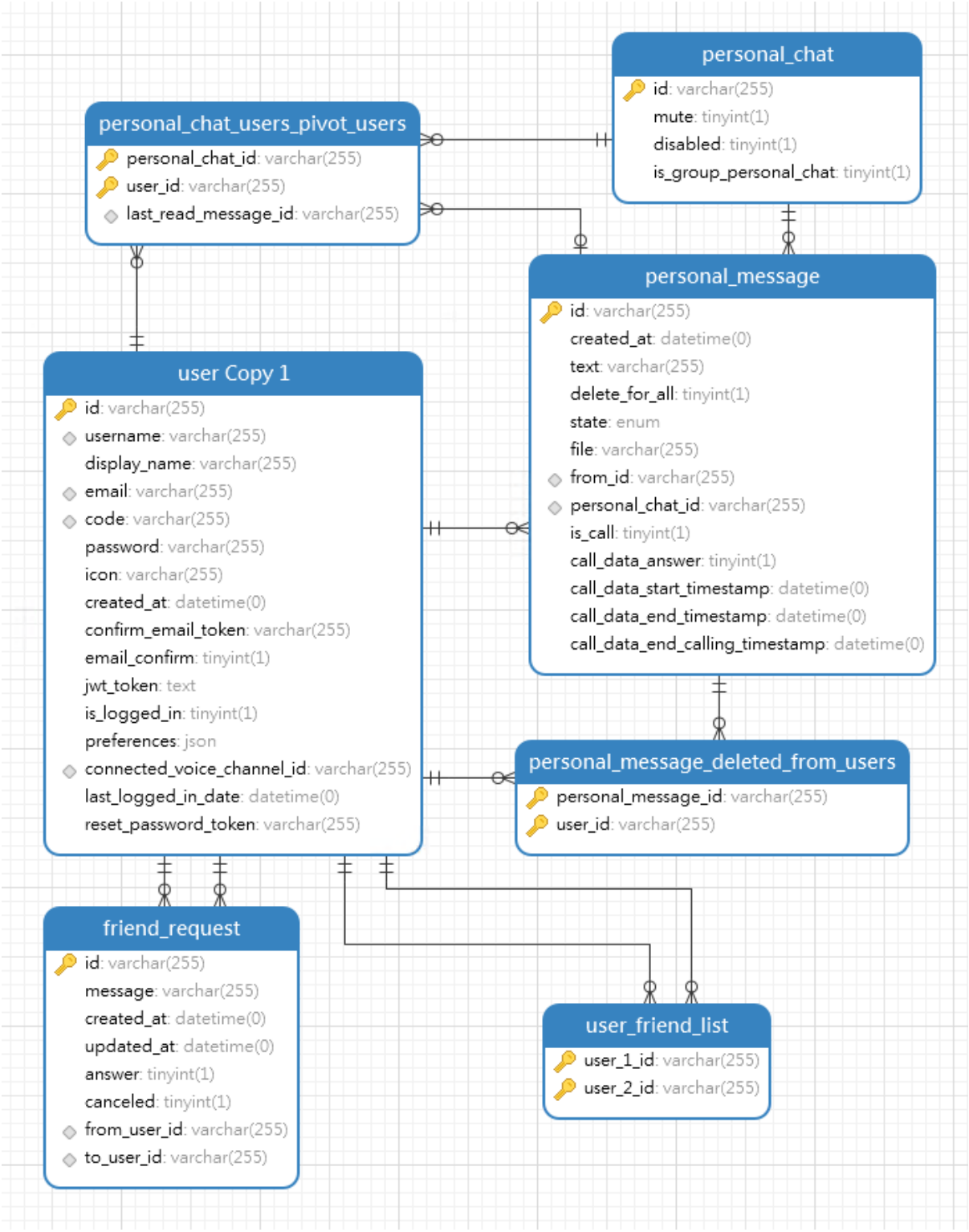
Ο πίνακας έχει την ίδια ακριβώς λειτουργία με τον πίνακα personal_message_deleted_from_user με την διαφορά πως το πεδίο personal_message_id ονομάζεται text_channel_id και είναι το FOREIGN KEY του text_channel_message που έχει διαγράψει το μέλος του group για τον εαυτό του.

3.18 voice_channel Table

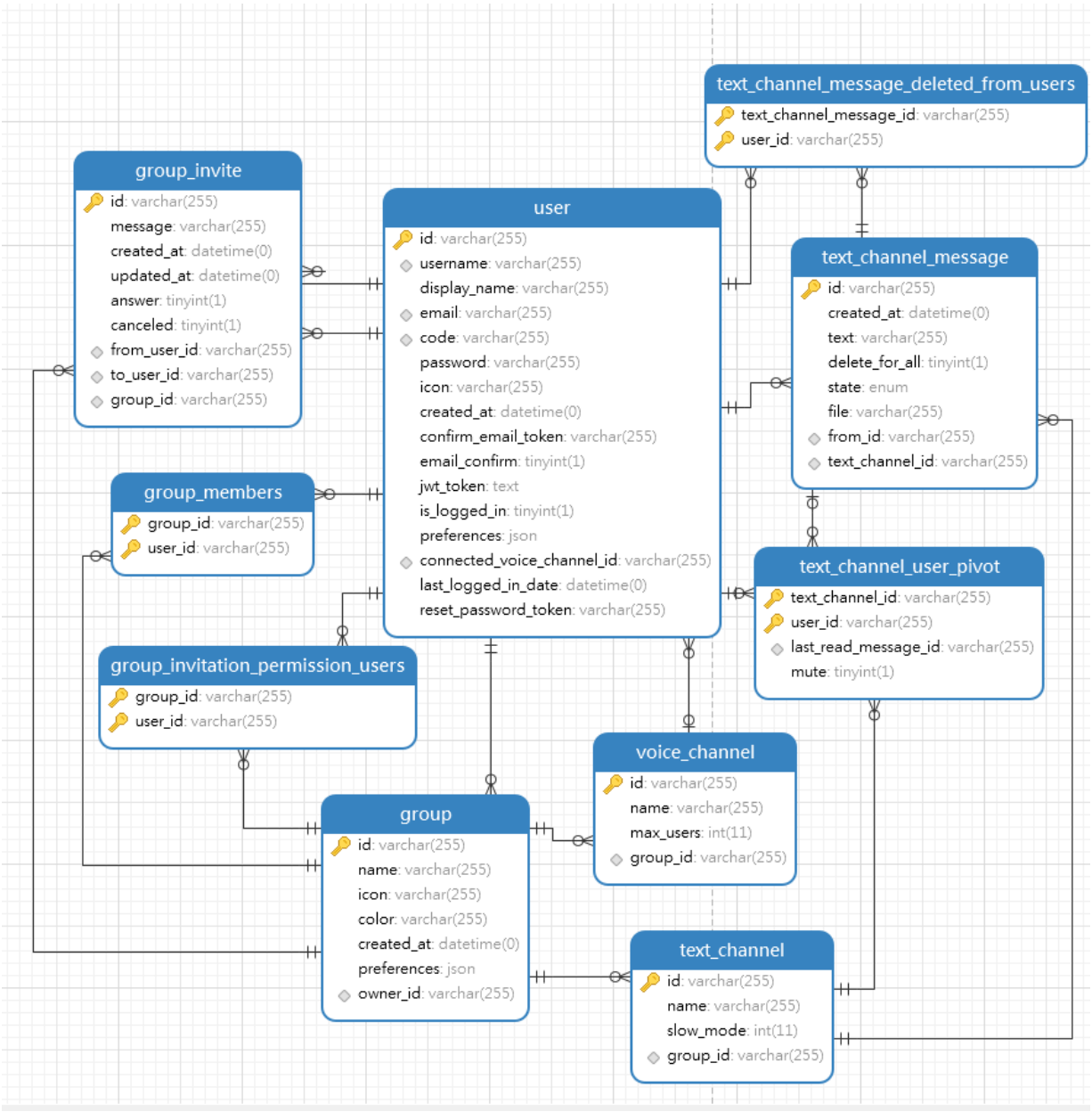
Ο πίνακας voice_channel περιγράφει ένα κανάλι ήχου όπου τα μέλη ενός group μπορούν να συνδεθούν και να μιλήσουν μεταξύ τους μέσω ήχου. Τα πεδία του είναι:

- **name:** Είναι τύπου VARCHAR(255) και αποθηκεύει το όνομα του καναλιού ήχου που βλέπουν τα μέλη του group.
- **max_users:** Είναι τύπου INT NULL και περιγράφει το πλήθος τον μελών που μπορούν να συνδεθούν ταυτόχρονα στο κανάλι ήχου. Όταν η τιμή του είναι NULL δεν υπάρχει κάποιος περιορισμός, ενώ όταν έχει τιμή, επιτρέπεται να συνδεθούν μόνο ο αριθμός χρηστών της τιμής.
- **group_id:** Είναι τύπου VARCHAR(255) και είναι FOREIGN KEY του πίνακα group και περιγράφει το group όπου ανήκει το κανάλι ήχου.

3.19 ER Διαγράμματα



Εικόνα 2: Συσχέτιση πίνακα User με πίνακες λειτουργιών φίλων



Εικόνα 3: Συσχέτιση πίνακα User με πίνακες λειτουργιών group

Κεφάλαιο 4ο: Υλοποίηση Back-End

Σε αυτό το κεφάλαιο θα δούμε την διαδικασία ανάπτυξης του back-end της εφαρμογής τι δυσκολίες εμφανίστηκαν κατά την διάρκεια της και πως αντιμετωπίστηκαν.

4.1 Επιλογή Είδος API

Αρχικά, έπρεπε να διαλέξουμε το είδος του API που πρέπει να χρησιμοποιήσουμε. Επειδή η δομή των δεδομένων είναι περίπλοκη και επειδή τα δεδομένα που μπορεί να χρειαστεί να ανακτήσει κάποιος χρήστης, με την πάροδο του χρόνου, μπορεί να ήταν μεγάλα σε όγκο, αποφασίστηκε να υλοποιηθεί ένα GraphQL API. Αυτό δίνει την δυνατότητα στον client να ανακτήσει ακριβώς τα δεδομένα που χρειάζεται την εκάστοτε στιγμή, μειώνοντας έτσι τον όγκο των δεδομένων.

4.2 Επιλογή Διαχείριση της Βάσης Δεδομένων

Για την διαχείριση της βάσης δεδομένων, αποφασίστηκε να χρησιμοποιηθεί ORM ώστε να μην χρειάζεται να υλοποιηθούν custom query strings για κάθε αλληλεπίδραση με την βάση. Καθώς υπάρχουν πολλές βιβλιοθήκες ORMs για την Node.js, αποφασίστηκε λόγω οικειότητας να χρησιμοποιηθεί το Mikro-ORM. Χρησιμοποιεί "διακοσμητές" (decorators) για την αναγνώριση των πινάκων και των πεδίων τους. Για την δημιουργία και επεξεργασία όλων των πινάκων της εφαρμογής, εκτελείται ένα script που ονομάζεται migration. Αυτό το script βλέπει όλα τα decorators του Mikro-ORM και να μεταφράζει ως ερωτήματα στην βάση για την δημιουργία ή επεξεργασία των πινάκων της. Στο επόμενο υποκεφάλαιο που αφορά τα μοντέλα της εφαρμογής, θα δείξουμε το πως δηλώνονται οι πίνακες και τα πεδία τους.

4.3 Δήλωση Μοντέλων/Κλάσεων και Πεδίων

Για την αναγνώριση μιας κλάσης από το Mikro-ORM και την GraphQL, πρέπει η κάθε κλάση και κάθε πεδίο της να περιγράφεται από τουλάχιστον έναν decorator από κάθε βιβλιοθήκη. Τα πεδία που δεν έχουν κάποιο decorator δεν αναγνωρίζονται ούτε από το Mikro-ORM ούτε από την GraphQL. Για να μπορέσουμε να δώσουμε ένα παράδειγμα για το πως χρησιμοποιήθηκαν και πως ορίζονται τα GraphQL και Mikro-ORM πεδία, θα αναλύσουμε την κλάση Users, ώστε να δούμε όλες σχέσεις που χρησιμοποιούνται στην εφαρμογή μεταξύ των μοντέλων.

4.4 Μοντέλο User

Καθώς ανοίγουμε το αρχείο User.ts, προσπερνώντας τα imports των διάφορων dependencies, φτάνουμε στην δήλωση της κλάσης Users:

```
@Entity()
@Unique({ properties: ['username', 'code'] })
@ObjectType()
export class User {
```

Απόκομμα Κώδικα 13: Αρχικοποίηση μοντέλου User

Και παρατηρούμε, ότι πάνω από την δήλωση της κλάσης, υπάρχουν οι εξής decorators:

- @Entity(): Αυτός ο decorator είναι της Mikro-ORM και σκοπός του είναι η αναγνώριση της κλάσης User ως έναν πίνακα της βάσης δεδομένων. Επομένως, αντιστοιχίζει την κλάση με τον πίνακα user της βάσης. Στο εσωτερικό των παρενθέσεων, μπορεί να δεχτεί ένα αντικείμενο "options", όπου μπορείς να επεξεργαστείς τις ρυθμίσεις της κλάσης, όπως για παράδειγμα, ο

decorator @Entity({ tableName: 'sousou_users' }) αντιστοιχεί την κλάση στον πίνακα sousou_users. Επομένως, όταν θα εκτελεσθεί το script για την δημιουργία του migration αρχείου, γνωρίζει πως πρέπει να δημιουργηθεί ο πίνακας με το default όνομα της κλάσης (user) ή αν έχει δηλωθεί διαφορετικό. Αν στην βάση δεν υπάρχει ο πίνακας user, το script θα δημιουργήσει το ερώτημα "create table `user` ..." για την βάση δεδομένων.

- @Unique({ properties: ['username', 'code'] }): Και αυτός ο decorator είναι του Mikro-ORM και χρησιμοποιείται όταν υπάρχει η ανάγκη μοναδικότητας συνδυασμού μεταξύ δύο ή παραπάνω πεδίων του μοντέλου. Στην προκειμένη περίπτωση, περιγράφει την μοναδικότητα του συνδυασμού των πεδίων username και code στον πίνακα του μοντέλου User. Επομένως, η εκτέλεση του migration script, θα παράξει το εξής ερώτημα: "alter table `user` add unique `user_username_code_unique`(`username`, `code`);".
- @ObjectType(): Αυτός ο decorator είναι της GraphQL και σκοπός του είναι να προσθέσει το μοντέλο User στο GraphQL schema ώστε ο client να μπορεί να ζητάει πεδία του, όταν εκτελεί queries που επιστρέφουν δεδομένα του μοντέλου User.

Στην συνέχεια θα δούμε κάποια από τα πεδία του μοντέλου User.

4.4.1 Πεδίο id

```
@PrimaryKey({ fieldName: 'user_id' })
@Field()
 id: string = v4()
```

Απόκομμα Κώδικα 14: Πεδίο id του μοντέλου User

Παρατηρούμε τους εξής decorators:

- @PrimaryKey(): Αυτός ο decorator είναι του Mikro-ORM και σκοπός του είναι να αντιστοιχίσει το πεδίο id της κλάσης στο πεδίο id του πίνακα του μοντέλου User και να το δηλώσει ως κύριο κλειδί του. Όπως και στην περίπτωση του πίνακα, έτσι και εδώ, μέσα στις παρεθνέσει μπορεί να δεχτεί ένα αντικείμενο options, ώστε να προσαρμόσεις το πεδίο στις ανάγκες της αφαρμογής. Πχ ο decorator @PrimaryKey({ fieldName: 'user_id' }) θα αντιστοιχίσει το πεδίο id της κλάσης στο πεδίο user_id του πίνακα user στην βάση δεδομένων. Επομένως, η εκτέλεση του migration script, διαμορφώνει το ερώτημα της δημιουργίας του πίνακα ως "create table `user` (`id` varchar(255) not null, ..., primary key (`id`))".
- @ Field(): Αυτός ο decorator είναι της GraphQL και δηλώνει το πεδίο id ως μέρος του μοντέλου User ώστε να προστεθεί στο GraphQL schema και να μπορεί να ζητηθεί από τον client στα queries.

4.4.2 Πεδίο username

```
@Property()
@Field()
 username: string
```

Απόκομμα Κώδικα 15: Πεδίο username του μοντέλου User

και συναντάμε ένα νέο decorator:

- @Property(). Αυτός ο decorator είναι του Mikro-ORM και σκοπός του είναι η αντιστοίχιση του πεδίο username του μοντέλου, στο πεδίο username του πίνακα. Όπως και στους προηγούμενους, χρησιμοποιείται το αντικείμενο options για την παραμετροποίηση του πεδίου στις ανάγκες της εφαρμογής. Επομένως, κατά την εκτέλεση του migration script, δημιουργεί ή επεξεργάζεται το πεδίο του πίνακα που του αντιστοιχεί.

4.4.3 Πεδίο email

```
@Property({ unique: true })
@Field()
 email: string
```

Απόκομμα Κώδικα 16: Πεδίο email του μοντέλου User

Παρατηρούμε πως στα options του @Property decorator, δίνεται η τιμή true στο πεδίο unique. Αυτό έχει ως αποτέλεσμα να καθορίζει το email κάθε χρήστη ως μοναδικό στην βάση. Κατά την εκτέλεση του migration script παράγει ερώτημα για να δημιουργήσει ή επεξεργαστεί το πεδίο email.

4.4.4 Πεδίο password

```
@Property({ hidden: true })
 password: string
```

Απόκομμα Κώδικα 17: Κρυφό πεδίο password του μοντέλου User

Βλέπουμε πως έχει μόνο έναν decorator, του Mikro-ORM και όχι της GraphQL.

- Αυτό συμβαίνει γιατί δεν θέλουμε το password να μην μπορεί να ανακτηθεί από τους χρήστες και να παραμένει κρυφό, ασχέτως που είναι κρυπτογραφημένο.
- @Property({ hidden: true }): Παρατηρούμε πως στο αντικείμενο options, δίνεται η τιμή true στο πεδίο hidden. Όταν με το Mikro-ORM εκτελείς κάποιο select ερώτημα, από default επιστρέφει όλα τα πεδία του πίνακα. Ενεργοποιώντας την επιλογή hidden, το πεδίο δεν ανακτάται, εκτός αν ζητηθεί συγκεκριμένα το πεδίο.

4.4.5 Πεδίο jwtEndTimestamp

```
@Property({ type: 'bigint', nullable: true })
@Field({ nullable: true })
 jwtEndTimestamp?: number
```

Απόκομμα Κώδικα 18: Πεδίο jwtEndTimestamp του μοντέλου User

Παρατηρούμε πως υπάρχουν κάποιες διαφορές από τις προηγούμενες δηλώσεις.

- @Property({ type: 'bigint', nullable: true }): Βλέπουμε πως παραμετροποιούνται δύο options, το type και το nullable:
    - type: Το type αφορά τον τύπο του πεδίου στην βάση δεδομένου. Στην προκειμένη περίπτωση χρειαζόμασταν μεγάλο integer καθώς αποθηκεύει το timestamp της λήξης ενός JWT.
    - nullable: Όταν ενεργοποιείται, επιτρέπει στο πεδίο να αποθηκεύσει την τιμή null. Όσα πεδία δεν έχουν αυτήν την επιλογή ενεργοποιημένη, απαιτείται να αποθηκεύσουν κάποια τιμή.
- @Field ({ nullable: true }): Η επιλογή nullable της GraphQL, έχει την ίδια ιδιότητα με το nullable του Mikro-ORM. Επομένως, όλα τα πεδία που δεν την περιλαμβάνουν, απαιτείται να περιέχουν τιμή, όταν ζητηθούν από τον client.

4.4.6 Πεδίο connectedVoiceChannel

Αφορά την σχέση μεταξύ του μοντέλου User και του μοντέλου VoiceChannel.

```
@ManyToOne(() => VoiceChannel, { nullable: true })
@Field(() => VoiceChannel, { nullable: true })
 connectedVoiceChannel?: VoiceChannel
```

Απόκομμα Κώδικα 19: Πολλά προς ένα σχέση του μοντέλου VoiceChannel και User

- @ManyToOne(() => VoiceChannel, { nullable: true }): Αυτός ο decorator είναι του Mikro-ORM και περιγράφει την M:1 σχέση μεταξύ του πίνακα του μοντέλου User και του πίνακα του μοντέλου VoiceChannel. Κατά την εκτέλεση του migration script, παράγεται τα ερωτήματα:
  - "create table `user`, (..., `connected_voice_channel_id` varchar(255) null);"
  - "alter table `user` add index `user_connected_voice_channel_id_index` (`connected_voice_channel_id`);" και
  - "alter table `user` add constraint `user_connected_voice_channel_id_foreign` foreign key (`connected_voice_channel_id`) references `voice_channel` (`id`) on update cascade on delete set null;"
- @Field(() => VoiceChannel, { nullable: true }): Δηλώνει στο GraphQL schema πως αυτό το πεδίο περιέχει ένα Object τύπου VoiceChannel.

Και στους δύο decorators, απαιτούν την ύπαρξη των decorators @Entity και @ObjectType στο μοντέλο που δείχνουν αντίστοιχα.

4.4.7 Πεδίο groupInvites

```
@OneToMany(() => GroupInvite, group => group.toUser)
@Field(() => [GroupInvite])
 groupInvites = new Collection<GroupInvite>(this)
```

Απόκομμα Κώδικα 20: Ένα προς πολλά σχέση του μοντέλου GroupInites με το μοντέλο User

Αφορά την σχέση μεταξύ του μοντέλου User και GroupInvite.

- @OneToMany(() => GroupInvite, group => group.fromUser): Είναι decorator του Mikro-ORM και περιγράφει την 1:M σχέση του μοντέλου User και GroupInvite. Στο μοντέλο GroupInvite, υπάρχει ακόμα μια περιγραφή M:1 του πεδίο toUser με το μοντέλο User. Επομένως, η πληροφορία αυτής της σχέση αποθηκεύετεται στον πίνακα του μοντέλου GroupInvite στο πεδίο toUser.
- Field(() => [GroupInvite]): Δηλώνει στο GraphQL schema, πως το πεδίο groupInvites αποτελείται από έναν πίνακα από αντικείμενα τύπου GroupInvite.

4.4.8 Πεδίο friendList

```
   @ManyToMany(() => User)
@Field(() => [User])
 friendList = new Collection<User>(this)
```

Απόκομμα Κώδικα 21: Πολλά προς πολλά σχέση του μοντέλου User με τον αυτό του

Αυτό το πεδίο αφορά την σχέση του μοντέλου User με τον εαυτό του. Ο @ManyToMany(() => User): είναι decorator του Mikro-ORM και περιγράφει την M:M σχέση του μοντέλου User με τον εαυτό του. Αυτό γίνεται γιατί η λίστα φίλων αφορά μόνο users. Κατά την εκτέλεση του migration script, παράγονται τα παρακάτω ερωτήματα:

- "create table `user_friend_list` (`user_1_id` varchar(255) not null, `user_2_id` varchar(255) not null, primary key (`user_1_id`, `user_2_id`));" όπου δημιουργεί τον πίνακα user_friend_list.
- "alter table `user_friend_list` add index `user_friend_list_user_1_id_index` (`user_1_id`);" όπου προσθέτει ως index το πεδίο ` user_1_id`
- "alter table `user_friend_list` add index `user_friend_list_user_2_id_index` (`user_2_id`);" όπου προσθέτει ως index το πεδίο ` user_2_id`
- "alter table `user_friend_list` add constraint `user_friend_list_user_1_id_foreign` foreign key (`user_1_id`) references `user` (`id`) on update cascade on delete cascade;" όπου θέτει ως ξένο κλειδί το `user_1_id`

- “alter table `user_friend_list` add constraint `user_friend_list_user_2_id_foreign` foreign key (`user_2_id`) references `user` (`id`) on update cascade on delete cascade;” όπου θέτει ως ξένο κλειδί το `user_2_id`

Προφανώς υπάρχουν πολλά ακόμα πεδία και πολλές σχέσεις μεταξύ διάφορων μοντέλων, αλλά δεν χρειάζεται να αναλυθούν, καθώς σχεδόν όλες οι περιπτώσεις έχουν ήδη καλυφθεί.

4.5 Resolvers

Σε αυτό το υποκεφάλαιο θα αναλύσουμε πως δηλώνονται οι Resolvers και πως αυτοί εκτελούν της κατάλληλες μεθόδους. Ο τρόπος που δημιουργούμε όλους τους Resolvers και τις διαδικασίες τους είναι ίδιος. Δηλώνουμε τον Resolver, δηλώνουμε τις μεθόδους που εμπεριέχει και τέλος τρέχουμε την διαδικασία που εμπεριέχεται στο αρχείο “`όνομα_μοντέλου`Actions.ts”. Για την βοήθεια της περιγραφής θα χρησιμοποιηθεί ο UserResolver.

Όπως βλέπουμε το αρχείο UserResolver.ts, συναντάμε την δήλωση της κλάσης του:

```
@Resolver()
export class UserResolver {
```

Απόκομμα Κώδικα 22: User Resolver

Ο decorator @Resolver() είναι υπεύθυνος για την δήλωση του του resolver στο GraphQL schema. Συνεχίζοντας, βλέπουμε τις λειτουργίες τους, που χωρίζονται σε Queries και Mutations. Θα αναλύσουμε κάποιες από τις λειτουργίες τους

4.5.1 getLoggedUser Query

Αυτό το Query είναι υπεύθυνο ώστε ο συνδεδεμένος χρήστης, να ανακτήσει πληροφορίες για τον λογαριασμό του, που είναι απαραίτητες για την εκκίνηση της αλληλεπίδρασης του με την εφαρμογή.

```
@Query(() => User)
async getLoggedUser (
 @Ctx('em') em: EntityManager,
  @Ctx('user') user: User
): Promise<User> {
 return await getLoggedUserAction(user, em)
}
```

Απόκομμα Κώδικα 23: Παράδειγμα Query του μοντέλου User

Βλέποντάς το, παρατηρούμε 2 καινούριους GraphQL decorators:

- @Query(() => User): Ο decorator @Query δηλώνει πως η συγκεκριμένη μέθοδος αποτελεί μόνο μέσω ανάκτησης δεδομένων και όχι εισαγωγής ή επεξεργασίας. Το εσωτερικό των παρεθέσεων δηλώνει το μοντέλο που επιστρέφεται εκτελόντας αυτήν την διαδικασία. Προφανώς το μοντέλο πρέπει οποσδήποτε να περιέχει τον decorator @ObjectType(), διαφορετικά η GraphQL δεν θα το αναγνωρίσει.
- @Ctx(): Ο decoratore @Ctx έχει τον ρόλο ανάκτησης δεδομένων που είναι αποθηκευμένο στο context του request. Αυτό μπορεί να είναι οποιαδήποτε πληροφορία μπορεί να είναι χρήσιμη για την εκτέλεση κάποια διαδικασίας. Στην προκειμένη διαδικασία για να ολοκληρωθεί, είναι απαραίτητη η ύπαρξη του διαχειρηστή της βάσης δεδομένων (em: EntityManager) και ο χρήστης (user: User) που αποθηκεύεται στο context με βάση το JWT που στέλνει ο client στο Authorization header.

4.5.2 deleteFriend Mutation

Αυτό το mutation είναι υπεύθυνο για την διαγραφή κάποιου χρήστη από την λίστα φίλων.

```
@Mutation(() => Boolean)
async deleteFriend (
 @Ctx('em') em: EntityManager,
  @Ctx('io') io: Server,
  @Ctx('ctx') ctx: AuthCustomContext,
  @Arg('id') id: string
): Promise<boolean> {
 return await deleteFriendAction(id, ctx.user, io, em)
}
```

Απόκομμα Κώδικα 24: Παράδειγμα Mutation του μοντέλου User

Παρατηρούμε τους εξής decorators:

- @Mutation(() => Boolean): Ο decorator @Mutation δηλώνει πως η συγκεκριμένη διαδικασία αφορά κυρίως την εισαγωγή, επεξεργασία ή διαγραφή κάποιων δεδομένων από την βάση δεδομένων. Ωστόσο, έχει την δυνατότητα να επιστρέφει δεδομένα στον client όπως και τα Queries. Στο εσωτερικό των παρενθέσεων δηλώνεται ο τύπος της απάντησης στο request.
- @Ctx(‘io’) io: Server: Το io είναι ο αποθηκευμένος WebSocket server. Ο λόγος για τον οποίο ανακτάτε στην συγκεκριμένη διαδικασία, είναι γιατί θέλουμε να στείλουμε πληροφορία στον χρήστη που διαγράφηκε, ώστε να ανανεωθεί η λίστα φίλων του σε πραγματικό χρόνο, χωρίς να εκτελέσει κάποιο περεταίρω request.
- @Arg(‘id’) id: string: Ο decorator @Arg αφορά τα δεδομένα εισαγωγής του χρήστη (input). Στην συγκεκριμένη περίπτωση, ο χρήστης στέλνει το id του χρήστη που θέλει να διαγράψει από την λίστα φίλων του.

Υπάρχουν ακόμα πολλοί Resolvers με πολλά Mutations και Queries, αλλά δεν χρειάζεται να αναφερθούμε σε όλα, καθώς καλύφθηκαν οι περιπτώσεις που χρειαζόμασταν για να κατανοήσουμε τους Resolvers.

4.6 main.ts file

Σε αυτό το υποκεφάλαιο, θα δούμε πως αρχικοποιούνται όλα τα services της εφαρμογής. Το αρχείο main.ts, είναι το αρχείο που εκτελείται πρώτο όταν ξεκινάει η εφαρμογή. Επομένως, περιέχει όλα τα σημαντικά χαρακτηριστικά όπως, η σύνδεση στην βάση, αρχικοποίηση του GraphQL server κλπ.

4.6.1 Αρχικοποίηση Σύνδεσης με την βάση

Το πρώτο πράγμα που συναντάμε στο main.ts αρχείο, είναι η γραμμή:

```
const connection = await MikroORM.init()
```

Απόκομμα Κώδικα 25: Αρχικοποίηση σύνδεσης του Mikro-ORM με MySQL

Αυτή είναι υπεύθυνη για την αρχικοποίηση της σύνδεσης του API με την βάση δεδομένων. Αυτό που κάνει η μέθοδος MikroORM.init() είναι να διαβάζει το αρχείο `mikro-orm.config.ts` ώστε να πάρει πληροφορίες για την βάση που ενδέχεται να συνδεθεί, όπως ο χρήστης της βάσης δεδομένων, ο κωδικός του, σε ποια ip φιλοξενείται κ.α. Έπειτα, επιχειρεί να συνδεθεί στην βάση με αυτά τα δεδομένα. Εφόσον η προσπάθεια είναι επιτυχής, επιστρέφει την σύνδεση στην μεταβλητή connection.

4.6.2 Αρχικοποίηση GraphQL Schema

Αμέσως επόμενη διαδικασία που συναντάμε, είναι η αρχικοποίηση του GraphQL schema:

```
const schema = await buildSchema({
  resolvers: [
   UserResolver,
   FriendRequestResolver,
   GroupResolver,
   GroupInviteResolver,
   AuthFreeResolver,
   TextChannelResolver,
   VoiceChannelResolver,
   PersonalMessagesResolver
  ],
  globalMiddlewares: [isLogged, ErrorInterceptor]
 })
```

Απόκομμα Κώδικα 26: Αρχικοποίηση GraphQL σχήματος

Όπως βλέπουμε, με την βοήθεια της μεθόδου buildSchema, που χρησιμοποιούμε από την βιβλιοθήκη 'type-graphql', δημιουργούμε το GraphQL schema. Εδώ πρέπει να προσθέσουμε όλους τους Resolvers που θέλουμε να είναι ανοιχτοί προς τους clients. Αν κάποιος Resolver δεν δηλωθεί στο συγκεκριμένο σημείο, δεν θα μπορεί να χρησιμοποιηθεί από τους clients. Επίσης, όπως βλέπουμε εδώ δηλώνονται και τα GraphQL Middlewares, όπου εκτελούνται κάθε φορά που κάποιος χρήστης εκτελεί ένα GraphQL request στο API. Το isLogged αφορά τον έλεγχο αν ο χρήστης είναι ενεργός ή όχι. Το ErrorInterceptor middleware, είναι υπεύθυνο για να μην επιτρέψει κάποιο τυχόν σφάλμα να σταματήσει την λειτουργία του API και να επιστρέφει στους χρήστες να κατάλληλα μηνύματα και κωδικούς σφαλμάτων. Είναι επίσης σημαντικό να αναφερθεί, πως τα http middlewares εκτελούνται πριν από τα GraphQL middlewares.

4.6.3 Αρχικοποίηση Koa και io Server

Για να λειτουργήσει ένα GraphQL API, πρέπει να χρησιμοποιηθεί ένας http server και να δεσμεύσει το route "${api_ip}/graphql" όπου και φιλοξενεί όλο το GraphQL API.

```
  const app = new Koa()
 const httpServer = createServer(app.callback())

 const io = new Server(httpServer, {
  pingInterval: 25000,
  pingTimeout: 60000,
  transports: ['websocket']
 })

 await initSocketEvents(io, connection.em.fork())
```

Απόκομμα Κώδικα 27: Αρχικοποίηση Koa και socket.io server

Όπως βλέπουμε αρχικοποιούμε έναν Koa server όπου θα φιλοξενήσει το GraphQL API. Στην πορεία, δημιουργούμε έναν Socket.io server ώστε να μπορέσει το API να διατηρήσει μια σύνδεση WebSockets με τους clients ώστε να τους στέλνει πληροφορίες σε πραγματικό χρόνο που τους αφορούν. Αμέσως μετά, τρέχει η μέθοδος που είναι υπεύθυνη για να αρχικοποιήσει τα events όπου θα ακούει ο WebSocket

server, όπως η αποστολή μηνύματος από ένα χρήστη σε ένα άλλο ή η ανταλλαγή των candidates για την εκκίνηση μιας κλήσης ήχου. Παρακάτω παρατίθεται το documentation για τα events που ακούει ο client και ο server.

4.6.3.1 UI WebSocket Events

Event Name	Description	Response Event
'connection'	Ελέγχει αν το JWT είναι έγκυρο. Αν είναι, αποθηκεύει τον χρήστη ως συνδεδεμένο και τον τοποθετεί στα κατάλληλα webscoket δωμάτια. Διαφορετιά, αποσυνδέει την σύδεση με το WebSocket απευθείας.	'authorization'
'message-send'	Στέλνει μήνυμα σε Personal Chat ή σε ένα Text Channel ενός Group.	'message-receive'
'message-read'	Αποθηκεύει το τελευταίο μήνυμα που διάβασε ο χρήστης ως ‘Διαβάστηκε’, είτε σε ένα Personal Chat είτε σε κάποιο Text Channel.	'message-read'
'message-delete'	Διαγράφει το μήνυμα ενός χρήστη ειδοποιεί τα κατάλληλα WebSocket δωμάτια.	'message-deleted'
‘disconnect'	Κατά την αποσύνδεση του χρήστη από το socket, ξεκινάει αντίστροφη μέτρηση 30 δευτερολέπτων για να ειδοποιείσει τα κατάλληλα δωμάτια πως ο χρήστη αποσυνδέθηκε. Αν ο χρήστης ξανασυδεθεί στο WebSocket μέσα σε αυτόν τον χρόνο, σταματάει και η μέτρηση.	

4.6.3.2 API WebSocket Events

Event Name	Description	Response Type
'authorization'	Ενημερώνει τον χρήστη πως η σύνδεση με το WebSocket ήταν επιτυχής, δηλαδή πως το JWT του ήταν έγκυρο.	'succeeded'
'message-receive'	Σε αυτό το event ο χρήστης δέχεται τα μηνύματα που στέλνουν άλλοι χρήστες σε κάποιο Personal Chat ή σε κάποιο Text Channel και τον αφορούν.	PersonalChat Message
'message-read'	Σε αυτό το event ο χρήστης δέχεται την σχέση ενός άλλου χρήστη, και το τελευταίο μήνυμα που διάβασε, ώστε να ενημερώνεται για τον κάθε χρήστη τι έχει διαβάσει.	PersonalChat UserPivot
'message-deleted'	Σε αυτό το event ο χρήστης δέχεται ένα μήνυμα που έχει διαγραφεί από κάποιον άλλο χρήστη.	TextChannel Message

	Σκοπός του δηλαδή είναι η απόκρυψη αυτού του μηνύματος από το UI.	
'something-changed'	Ενημερώνει τους χρήστες πως έχει γίνει κάποια αλλαγή που τον αφορά, πχ, αλλαγή ονόματος ενός χρήστη ή αλλαγή ονόματος ενός Group.	User
'connected-user-in-voice-channel'	Ενημερώνει τα μέλη ενός Group πως κάποιος χρήστης συνδέθηκε σε κάποιο Voice Channel. Σκοπός του είναι ο χρήστης να βλέπει σε πραγματικό χρόνο ποιοι συνδέονται στα Voice Channels.	VoiceChannel
'disconnect-user-from-voice-channel'	Ενημερώνει τα μέλη ενός Group πως κάποιος χρήστης αποσυνδέθηκε από ένα Voice Channel.	VoiceChannel
'update-group'	Ενημερώνει τα μέλη ενός Group πως έχει γίνει αλλαγή σε κάποιες πληροφορίες του.	Group
'update-text-channel'	Ενημερώνει τους χρήστες για τις αλλαγές που έγιναν σε κάποιο Text Channel.	TextChannel
'new-invite'	Ενημερώνει τους χρήστες πως κάποιος τους έστειλε πρόσκληση συμμετοχής σε κάποιο Group.	GroupInvite

4.6.4 Apollo Server

Σε αυτό το υπο-κεφάλαιο θα δούμε πως αρχικοποιείται ο Apollo server και πως τον παραμετροποιούμε σύμφωνα με τις ανάγκες της εφαρμογής.

```
const apolloServer = new ApolloServer({
  schema,
  context ({ ctx }: { ctx: Context }): CustomContext {
   return {
    ctx,
    em: connection.em.fork(),
    io: io
   }
  },
  plugins: [
   ApolloServerPluginDrainHttpServer({ httpServer })
  ]
 })

 await apolloServer.start()
```

Απόκομμα Κώδικα 28: Αρχικοποίηση Apollo server

Αρχικά, πρέπει να καθορίσουμε το GraphQL schema που δημιουργήσαμε προηγουμένως. Στην πορεία, μετατρέπουμε το προκαθορισμένο Context στο δικό μας τύπου CustomContext ώστε να το χρησιμοποιούμε κατά την εκτέλεση ενός Query ή Mutation.

```
export interface CustomContext {
 ctx: Context
 em: EntityManager
 user?: User
 io: Server
}
```

Απόκομμα Κώδικα 29: Interface του CustomContext

Τα πεδία του νέου CustomContext, όπως βλέπουμε στην φωτογραφία, είναι τα εξής:

- ctx: Είναι το αρχικό Context και σκοπός του είνια να παίρνουμε πληροφορίες για το request του χρήστη
- em: Είναι ο διαχειριστής της βάσης δεδομένων (Entity Manager). Ουσιαστικά περνώντας τον em στο context, χρησιμοποιείται συνεχώς μία σύνδεση στην βάση για όλα τα requests χωρίς να χρειάζεται να την τερματίζουμε στο τέλος κάποιας λειτουργίας.
- user: Είναι τύπου User και σκοπός του είναι να αποθηκεύεται στο context του request ο χρήστης που το εκτελεί.
- io: Πρόκειται για τον WebSocket server και χρειάζεται κατά την εκτέλεση κάποιου Mutation που εκτελείται από έναν χρήστη το οποίο επεξεργάζεται πληροφορίες που ενδιαφέρουν άλλους χρήστες, πχ η κατάσταση σύνδεσης ενός φίλου των χρηστών.

Όπως βλέπουμε, στο πεδίο των plugins, προσθέτουμε το plugin που είναι υπεύθυνο για την ενσωμάτωση του koa server στον Apollo server και τέλος, τον εκκινούμε με την εντολή apolloServer.start().

4.6.5 Http Middlewares

Για την σωστή λειτουργία του API κατά την εκτέλεση των request, πρέπει να εκτελούνται κάποιες διαδικασίες πριν την πραγματική εκτέλεση τους. Αυτό επιτυγχάνεται μέσω των middlewares, τα οποία εκτελούνται το ένα μετά το άλλο μέχρι την εκτέλεση της λειτουργίας.

```
 app.use(cors())

 app.use(jwt({ secret: PRIVATE_KEY, passthrough: false }))

 app.use(authAndSettStateUser(connection.em.fork()))

 httpServer.listen({ port: PORT }, () => {
  console.log(`http://${HOST}:${PORT}/graphql`)
 })
```

Απόκομμα Κώδικα 30: Δήλωση των Middlewares

Τα middlewares ορίζονται στον koa server με την εντολή app.use(”middleware”) και αυτά που χρησιμοποιούμε είναι τα εξής:

- cors(): Το middlware cors ανήκει στην βιβλιοθήκη ‘@koa/cors’ και σκοπός του είναι να επιτρέπει στους clients να εκτελούν requests στο API από οποιαδήποτε διεύθυνση IP.
- jwt(): Το jwt middlware ανήκει στην βιβλιοθήκη ‘koa-jwt’ και η λειτουργία που εκτελεί, είναι να βλέπει αν το Authorization header περιέχει έγκυρη τιμή. Αν δηλαδή, ο χρήστης έχει συνδεθεί στην επιτυχώς εφαρμογή και αν του επιτρέπεται να εκτελεί requests στο API. Στην παράμετρο secret τοποθετείται το ιδιωτικό κλειδί της εφαρμογής, με το οποίο το JWT υπογράφηκε, ώστε να ελεγχθεί η εγκυρότητά του. Τέλος, αποθηκεύει τις πληροφορίεςτου χρήστη, που περιέχει το JWT, στο αρχικό context του request.

- authAndSettStateUser(): Πρόκειται για custom middleware και σκοπός του είναι να πάρει τον τις πληροφορίες του χρήστη που αποθηκεύτηκαν στο αρχικό context, να ανακτίσει όλες τις πληροφορίες του χρήστη από την βάση και τέλος, να τον αποθηκεύσει στο CustomContext της εφαρμογής.

Τέλος, ο server ξεκινάει να ακούει για requests στην πόρτα που του ορίζεται.

4.7 Λειτουργίες Εφαρμογής

4.7.1 AuthFree Resolver

Ο AuthFreeResolver, περιέχει queries και mutations που δεν χρειάζεται ο χρήστης, να είναι συνδεδεμένος στην εφαρμογή για να τα εκτελέσει. Αυτά είναι

- Queries:
  - usernameExists: Χρησιμοποιείται αυτόματα από το UI όταν ο χρήστης πληκτρολογεί το όνομα χρήστη στην φόρμα εγγραφής. Εκτελείται κάθε φορά που αλλάζει το κείμενο του εν δυνάμει ονόματος χρήστη στην φόρμα και επιστρέφει την τιμή true, αν αυτό υπάρχει.
- Mutations:
  - registerUser: Χρησιμοποιείται για να εγγραφεί ο χρήστης στην εφαρμογή, παιρνόντας τις βασικές του πληροφορίες, όπως όνομα χρήστη, κωδικό, email κ.α. Αν η εγγραφή είναι επιτυχής, στέλνει mail στον χρήστη με τον σύνδεσμο επαλήθευσης του λογαριασμού του. Επίσης, ο κωδικός του χρήστη αποθηκεύεται σε κωδικοποιημένος.
  - usernameExists: Χρησιμοποιείται αυτόματα από το UI όταν ο χρήστης πληκτρολογεί το όνομα χρήστη στην φόρμα εγγραφής. Εκτελείται κάθε φορά που αλλάζει το κείμενο του εν δυνάμει ονόματος χρήστη στην φόρμα και επιστρέφει την τιμή true, αν αυτό υπάρχει.
  - loginUser: Χρησιμοποιείται κατά την σύνδεση του χρήστη στην εφαρμογή και παράγει και επιστρέφει το JWT του χρήστη που είναι απαραίτητο για την αλληλεπίδραση του με το API. Επίσης, αποθηκεύει την χρονοσήμανση της λήξεις του JWT. Τέλος, συνδέει τον χρήστη στον WebSocket server.
  - resendEmailConfirmation: Χρησιμοποιείται από τον χρήστη σε περίπτωση που δεν έχει επαληθεύσει τον λογαριασμό του, και ο σύνδεσμος που του στάλθηκε έχει λήξει.
  - forgotPasswordAction: Χρησιμοποιείται από τον χρήστη όταν έχει ξεχάσει τον κωδικό του. Πρέπει να εισάγει το email του ώστε να του σταλθεί ο κατάλληλος σύνδεσμος για την επαναφορά του κωδικού του.
  - resetPasswordAction: Χρησιμοποιείται όταν ο χρήστης ανοίξει τον σύνδεσμο επαναφοράς κωδικού και εισάγει τον νέο του κωδικό. Μαζί με τον νέο του κωδικό, στέλνεται και το μοναδικό token που βρισκόταν στον σύνδεσμο ώστε να επαληθευτεί ο χρήστης.
  - confirmChangeEmailAction: Χρησιμοποιείται αυτόματα από το UI όταν ο χρήστης ανοίξει τον σύνδεσμο που του ήρθε στο παλιό του email. Ο σύνδεσμος περιέχει ένα μοναδικό token το οποίο αποστέλλεται για την επαλήθευση του χρήστη.

4.7.2 User Resolver

- Αυτός ο resolver, όπως και οι επόμενοι, απαιτούν ο χρήστης να είναι συνδεδεμένος στην εφαρμογή, να έχουν χρησιμοποιήσει δηλαδή, το loginUser Mutation και να τοποθετούν το JWT που παράχθηκε στο Authorization Header. Ο συγκεκριμένος περιλαμβάνει όλα τα Queries και Mutations αφορούν τους χρήστες. Αυτά είναι:
- Queries:
  - getUsers: Χρησιμοποιείται με διάφορους σε διάφορα στάδια της εφαρμογής. Σκοπός του είναι να βοηθήσει τον χρήστη να αναζητήσει άλλους χρήστες για διάφορες

λειτουργίες. Δέχεται ένα πεδίο αναζήτησης ώστε να βρει τους πιο πιθανούς χρήστες μπορεί να ψάχνει ο χρήστης.
    - getAvailableUsersToAdd: Χρησιμοποιείται όταν ο χρήστης αναζητάει κάποιον άλλον χρήστη για να του στείλει αίτημα φιλίας. Σκοπός του είναι να κάνει κανονικά αναζήτηση σε όλους τους χρήστες, εκτός από αυτούς που έχει ήδη στην λίστα φίλων του.
    - getAvailableUsersToInvite: Χρησιμοποιείται όταν ο χρήστης αναζητάει κάποιον άλλον χρήστη για να του στείλει αίτημα συμμετωχή σε κάποιο Group. Σκοπός του είναι να κάνει κανονικά αναζήτηση σε όλους τους χρήστες, εκτός από αυτούς που συμμετέχουν ήδη στο Group. Ως παραμέτρους δέχεται το id του gourp και το πεδίο κείμενο αναζήτησης.
    - getLoggedUser: Χρησιμοποιείται μετά την σύνδεση του χρήστη ή κατά την διαδικασία επαναφόρτωσης του UI. Σκοπός του είναι, ο χρήστης να ανακτεί όλες τις πληροφορίες για τον εαυτό του που είναι χρήσιμες για το UI, όπως τα Groups που σημμετέχει, η λίστα φίλων του κλπ. Δεν δέχεται κάποια παράμετρο, απλά αναγνωρίζει τον χρήστη από το JWT που εμπεριέχεται στο Authorization header.
    - getFriendRequests: Χρησιμοποιείται από τον χρήστη όταν ζητήσει να δει τα αιτήματα φιλίας του. Δέχεται μία παράμετρο `forMe` τύπου boolean η οποία διαχειρίζεται από το UI πχ αν θέλει να δει τα αιτήματα που πρέπει να απαντήσει, παίρνει την τιμή true, ενώ όταν θέλει να δει τα αιτήματα που έχει στείλει, παίρνει την τιμή false.
- Mutations
    - logoutUser: Χρησιμοποιείται από τον χρήστη όταν επιθυμεί να αποσυνδεθεί από την εφαρμογή. Αποσυνδέει τον client από τον WebSocket server και διαγράφει την ημερομηνία της λήξεις του πιο πρόσφατου JWT που παράχθηκε, ώστε να το καταστήσει άχρηστο.
    - updateUser: Χρησιμοποιείται από τον χρήστη όταν επιθυμεί να ανανεώσει τα προσωπικά του δεδομένα, όπως το όνομα που εμφανίζεται στην εφαρμογή, την εικόνα προφίλ του κ.α.
    - getUserById: Χρησιμοποιείται σε διάφορες διαδικασίες της εφαρμογής. Σκοπός του είναι να επιστρέφει πληροροφίες για τον χρήστη στον οποίο ανήκει το id το οποίο εισάγεται.
    - updateUserEmail: Χρησιμοποιείται από τον χρήστη όταν επιθυμεί να αλλάξει το email του λογαριασμού του. Δέχεται το καινούριο email και στέλνει mail στο παλιό του έναν σύνδεσμο αλλαγής email με ένα μοναδικό token που παράγεται εκείνη την στιγμή.
    - updateUserPassword: Χρησιμοποιείται από τον χρήστη όταν επιθυμεί να αλλάξει τον κωδικό πρόσβασής του. Δέχεται ως παράμετρο τον νέο κωδικό, τον κωδικοποιεί και τον αποθηκεύει στην βάση.
    - connectToVoiceChannel: Χρησιμοποιείται μόλις ο χρήστης επιθυμεί να συνδεθεί σε κάποιο voice channel ώστε να μπορεί να επικοινωνήσει μέσω ήχου με άλλους χρήστες ενός Group.
    - deleteFriend: Χρησιμοποιείται από τον χρήστη όταν επιθυμεί να διαγράψει κάποιον χρήστη από την λίστα φίλων του. Οι χρήστες συνεχίζουν να έχουν το κοινό τους personal chat.

4.7.3 FriendRequst Resolver

Αυτός ο resolver εμπεριέχει τα mutations που αφορούν τα αιτήματα φιλίας. Αυτά είναι:

- Queries: Αυτός ο resolver δεν περιέχει queries, καθώς η ανάγκη για ανάκτηση των αιτημάτων φιλίας, καλύφθηκε στον User resolver καθώς είναι πληροφορία που ανακτάται με βάση τον χρήστη.
- Mutations
    - createFriendRequest: Χρησιμοποιείται από τον χρήστη όταν επιθυμεί να στείλει ένα αίτημα φιλάς σε κάποιον άλλο χρήστη. Δέχεται ως παραμέτρους το id του χρήστη που αφορά το αίτημα φιλίας και το μήνυμα που επιθυμεί να το συνοδεύει.

- cancelFriendRequest: Χρησιμοποιείται όταν ο χρήστης επιθυμεί να ακυρώσει το αίτημα φιλίας που έχει στείλει σε κάποιον άλλον χρήστη. Δέχεται ως παράμετρο το id του αιτήματος φιλίας.
- answerFriendRequest: Χρησιμοποιείται όταν ένας χρήστης επιθυμεί να απαντήσει κάποιο αίτημα φιλίας. Δέχεται ως παράμετρο την απάντηση που είναι τύπου boolean και το id του αιτήματος. Όταν η απάντηση έχει την τιμή true, τότε το αίτημα γίνεται αποδεκτό, οι δύο χρήστες που αφορά το αίτημα προστίθενται στην λίστα φίλων του άλλου και παράγεται το προσωπικό (personal) τους chat στην βάση.

4.7.4 PersonalMessages Resolver

Αυτό ο resolver περιέχει μόνο μία λειτουργία. Αυτή είναι το query getPersonalMessages ByPersonalChatId. Σκοπός της είναι να επιστρέφει σελιδοποιημένα τα μηνύματα ενός personal chat. Δέχεται ως παράμετρο το id ενός personal chat και τα δεδομένα για που θα διαμορφώσουν την σελίδα.

4.7.5 Group Resolver

Αυτός ο resolver περιέχει όλες τις διαδικασίες που χρειάζονται για την διαχείρηση και την αλληλεπίδραση με ένα Group. Αυτές είναι:

- Queries:
  - getGroups: Χρησιμοποιείται όταν ο χρήστης μετακινείται στην σελίδα των Groups. Σκοπός του είναι να επιστρέφει βασικές πληροφορίες ενός Group, όπως το όνομα του, και την εικόνα του ώστε να εμφανιστούν στην λίστα των Groups που συμμετέχει ο χρήστης.
  - getGroupById: Χρησιμοποιείται όταν ο χρήστης έχει επιλέξει να μετακινείται στην σελίδα ενός Group. Δουλεύει συμπληρωματικά με το query getGroups, επιστέφει δηλαδή περισσότερες πληροφορίες για του Group καθώς δέχεται ως δεδομένα εισαγωγής το id του Group που επέλεξε από την λίστα. Μαζί με τις πληροφορίες του Group, επιστρέφει και τις βασικές πληροφορίες των Text Channels ώστε να μπορεί ο χρήστης να ανακτήσει περισσότερες πληροφορίες όταν επιλέξει κάποιο από αυτά.
- Mutations:
  - createGroup: Χρησιμοποιείται όταν ένας χρήστης επιθυμεί να δημιουργήσει ένα Group. Δέχεται ως παραμέτρους το όνομα που επιθυμεί να έχει το Group, την εικόνα του και ένα χρώμα σε δεκαεξαδική μορφή όπου χρησιμοποιείται ως θέμα του Group. Επίσης, δημιουργεί αυτόματα το πρώτο text channel με το όνομα General, όπου ο ιδιοκτήτης μπορεί να το αλλάξει στην πορεία. Τέλος, ο δημιουργός προστίθεται στην λίστα με τα μέλη του Group.
  - updateGroup: Χρησιμοποιείται από τον ιδιοκτήτη ενός Group όταν επιθυμεί να αλλάξει τις βασικές πληροφορίες του Group. Αφού οι αλλαγές γίνουν επιτυχώς, στέλνονται οι αλλαγές στα συνδεδεμένα μέλη του Group ώστε να ενημερωθούν σε πραγματικό χρόνο για αυτές.
  - deleteGroup: Χρησιμοποιείται από τον ιδιοκτήτη ενός Group όταν επιθυμεί να διαγράψει ένα Group. Δέχεται ως παράμετρο μόνο το id του Group και η διαδικασία διαγράφει τα Group, τα TextChannels και τα VoiceChannels του.
  - transferOwnershipToUser: Χρησιμοποιείται από τον ιδιοκτήτη ενός Group όταν επιθυμεί να μεταφέρει την ιδιοκτησία του Group σε άλλον χρήστη. Δέχεται ως παράμετρο το id του Group που επιθυμεί να μεταφέρει καθώς και το id του χρήστη που θα γίνει ο νέος ιδιοκτήτης του.

4.7.6 GroupInvite Resolver

Αυτός ο resolver περιέχει όλες τις λειτουργίες που αφορούν τις προσκλήσεις συμμετοχής σε Group. Αυτές είναι:

- Queries:
    - getGroupInvites: Χρησιμοποιείται από τον χρήστη όταν επιλέξει να δει είτε τις προσκλήσεις που του έχουν στείλει για ένα Group, είτε αυτές που έχει στείλει ο ίδιος. Αυτό γίνεται με τον ίδιο τρόπο του Friend Request, δέχεται δηλαδή την παράμετρο forMe που είναι τύπου boolean.
- Mutations:
    - createGroupInvite: Χρησιμοποιείται από τον χρήστη όταν θέλει να στείλει πρόσκληση συμμετοχής σε ένα Group, σε έναν άλλον χρήστη. Ως παραμέτρους δέχεται το id του Group, το id του προσκεκλημένου χρήστη και το μήνυμα που επιθυμεί ο αποστολέας να συνοδεύει την πρόσκληση. Προσκλήσεις για ένα Group μπορεί να στέλνει μόνο ο ιδιοκτήτης εκτός αν ο ίδιος έχει δώσει αυτήν την δυνατότητα και στα υπόλοιπα μέλη του. Αυτό γίνεται αποθηκεύοντας την τιμή true στο πεδίο invitationPermissionUsers.
    - cancelGroupInvite: Χρησιμοποιείται από το άτομο που δημιούργησε την πρόσκληση ώστε να την ακυρώσει. Ως παράμετρο δέχεται μόνο το id της πρόσκλησης.
    - answerGroupInvite: Χρησιμοποιείται από τον χρήστη όταν επιθυμεί να απαντήσει σε μια πρόσκληση συμμετωχής σε Group. Δέχεται ως παραμέτρους το id της πρόσκλησης και την απάντηση του στην παράμετρο answer που είναι τύπου boolean. Αν η τιμή του answer είναι true, τότε ο χρήστης γίνεται μέλος του Group.

4.7.7 TextChannel Resolver

Αυτό ο resolver περιλαμβάνει όλες τις λειτουργίες που αφορούν τα κανάλια κειμένου ενός Group. Αυτά είναι:

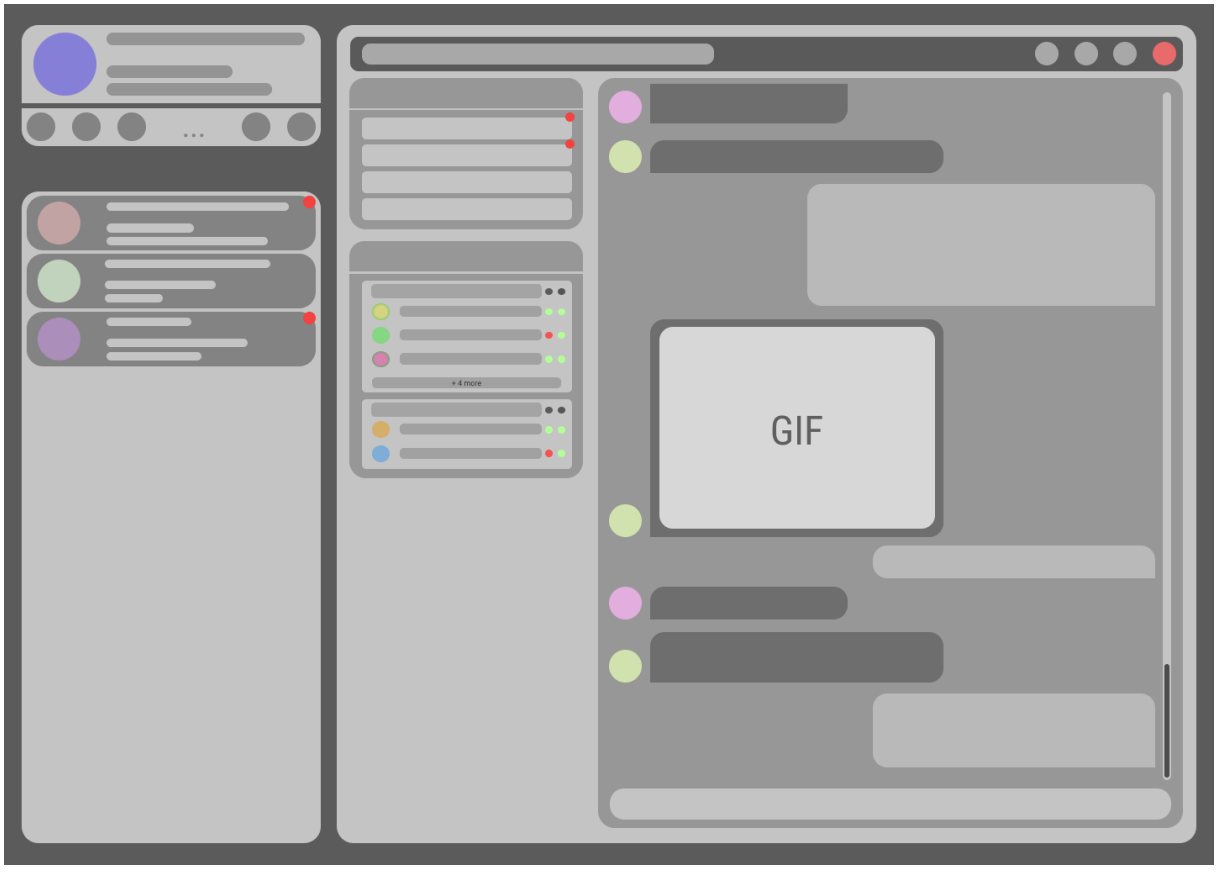
- Queries:
    - getTextChannelById: Χρησιμοποιείται από ένα μέλος του Group όταν επιλέξει να δει ένα Text Channel. Δέχεται ως παράμετρο το id του Text Channel που επέλεξε από την λίστα ο χρήστης, ώστε να επιστρέψει περαιτέρω πληροφορίες.
    - getPaginatedTextChannelMessagesByTextChannelId: Χρησιμοποιείται από ένα μέλος του Group όταν έχει ανοιχτό ένα Text Channel και περιηγείται προς τα παλαιότερα μηνύματα του. Επιστρέφει, δηλαδή, τα μηνύματα που υπάρχουν στο Text Channel σε σελιδοποιημένη μορφή.
- Mutations:
    - createTextChannel: Χρησιμοποιείται από τον ιδιοκτήτη ενός Group όταν επιθυμεί να δημιουργήσει ένα Text Channel. Δέχεται ως παραμέτρους το id του Group που θα το φιλοξενεί, το όνομα του και την επιλογή slowMode που είναι τύπου number || null. Ο σκοπός του slowMode είναι, όταν περιέχει τιμή, επιτρέπει σε κάθε χρήστη να στείλει μήνυμα στο Text Channel ανά τόσα δευτερόλεπτα, όσα η τιμή του.
    - updateTextChannel: Χρησιμοποιείται από τον ιδιοκτήτη ενός Group όταν επιθυμεί να επεξεργαστεί τις βασικές πληροφορίες του Text Channel, όπως το όνομα και το slowMode.
    - deleteTextChannel: Χρησιμοποιείται από τον ιδιοκτήτη ενός Group όταν επιθυμεί να διαγράψει ένα Text Channel από ένα Group. Μαζί με την διαγραφή του, διαγράφονται και τα μηνύματα που βρισκόντουσαν σε αυτό.

Επειδή θέλαμε να προσθέσουμε την λειτουργία του ‘Διαβάστηκε’ στα Text Channels, δημιουργήσαμε και έναν πίνακα που αντιστοιχίζει τα μέλη του Group σε κάθε Text Channel. Αυτός ο πίνακας αποθηκεύει το Text Channel, το μέλος του Group και το τελευταίο μήνυμα που διάβασε. Έτσι, κάθε χρήστης μπορεί να δει το τελευταίο μήνυμα που διάβασε το κάθε μέλος του Group ώστε να γνωρίζει μέχρι ποιο σημείο είναι ενημερωμένος για κάποια συζήτηση.

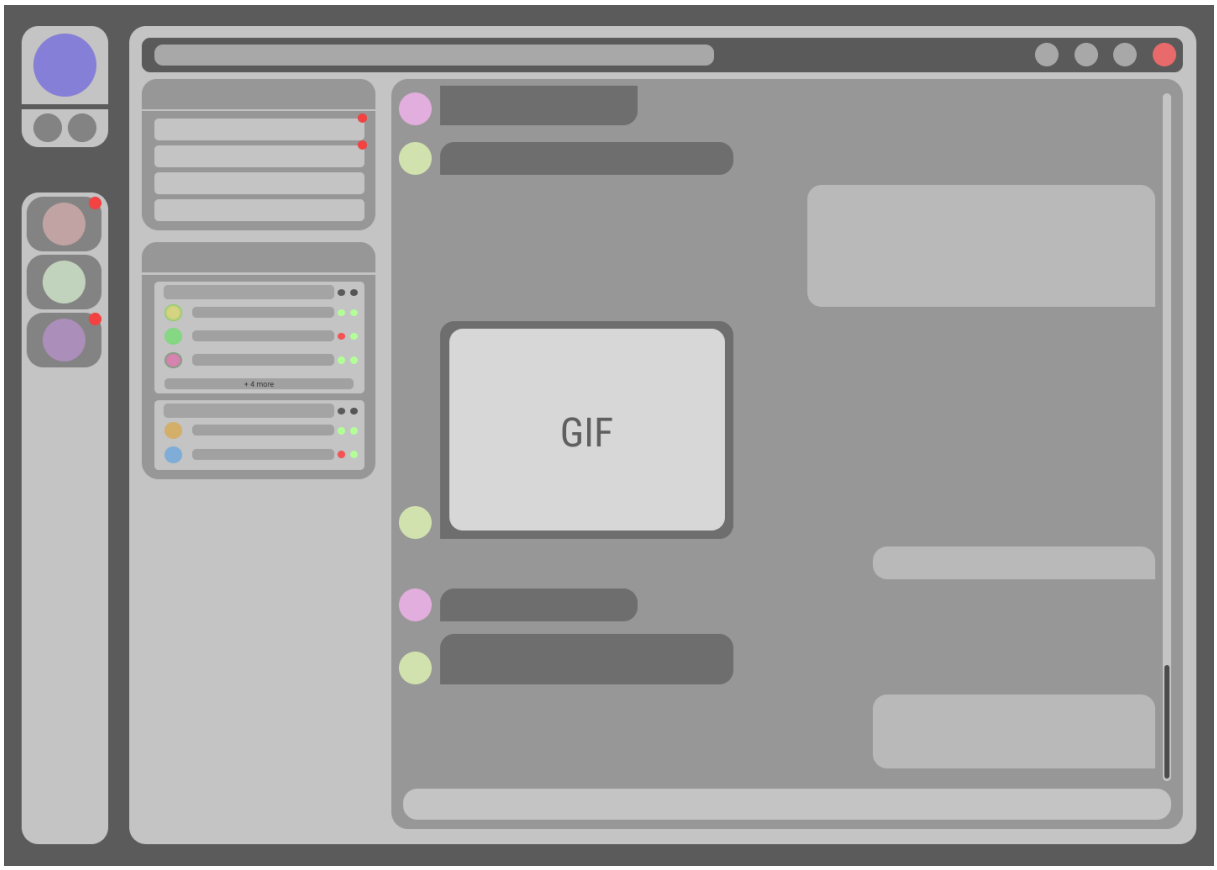
Κεφάλαιο 5ο: Υλοποίηση Front-End

5.1 Σχεδίαση του UI

Για την δημιουργία ενός UI είναι πολύ σημαντικό να ληφθεί υπόψιν η σχεδίαση του, προτού ξεκινήσει η ανάπτυξη σε κώδικα. Έτσι, η βασική ιδέα της μορφής του front-end ξεκίνησε από την σχεδίαση της στην Web εφαρμογή Figma. Το Figma είναι ένα εύκολο και χρήσιμο εργαλείο το οποίο επιτρέπει την σχεδίαση και δημιουργία mock ups (προσχεδίων) UI για Web, Desktop ή Mobile εφαρμογές. Το βασικό κομμάτι του προσχεδίου είχε σαν στόχο την εύκολη και γρήγορη εμφάνιση όλης της χρήσιμης πληροφορίας με όσο το δυνατών λιγότερη πλοήγηση του χρήστη. Η τελική μορφή του mock up παρουσιάζεται στην Εικόνα 4 και στην Εικόνα 5. Ο σκοπός του σχεδίου ήταν κατά βάση να καθοδηγήσει την υλοποίηση του κώδικα, έτσι ώστε να μην χρειαστεί κατά την συγγραφή του να υπάρχουν πολλές σκέψεις για το πως θα παρουσιάζεται η κάθε πληροφορία. Ωστόσο είναι σημαντικό να σημειωθεί πως δεν δημιουργήθηκε έτσι ώστε να ακολουθηθεί κατά γράμμα, αλλά ως μία βασική μορφή καθοδήγησης κατά την υλοποίηση. Όπως παρουσιάζεται και παρακάτω με επεξήγηση αλλά και εικόνες στο κεφάλαιο 7.2, θα παρατηρήσετε πως η τελική μορφή της εφαρμογής δεν ακολουθεί πλήρως το προσχέδιο που δημιουργήθηκε. Αυτό κυρίως γιατί κατά την ανάπτυξη παρατηρήθηκε πως κάποια τμήματα του προσχεδίου δεν λήφθηκαν σωστά υπόψιν, πράγμα που οδηγούν το τελικό προϊόν να διαφέρει με αυτό του mock up.



Εικόνα 4: Προσχέδιο UI 1



Εικόνα 5: Προσχέδιο UI 2

5.2 Υλοποίηση

Αφότου ολοκληρώθηκε η σχεδίαση του UI, επόμενο βήμα έχει η δημιουργία του project. Το περιβάλλον συγγραφής κώδικα που επιλέχθηκε είναι το VS Code. Ο συγκεκριμένος code editor επιλέχθηκε λόγω της άψογης υποστήριξης που παρέχει για web εφαρμογές. Είναι πολύ ελαφρύς και δίνει την δυνατότητα παραμετροποίησης με την χρήση των plugins (επιπρόσθετων) που προσφέρει ο ίδιος ο editor αλλά ακόμη περισσότερο η κάθε κοινότητα που το χρησιμοποιεί. Στην περίπτωση της Vue υπάρχει πολύ μεγάλη πληθώρα από εργαλεία που παρέχονται για να γίνει εύκολη η συγγραφή.

Για την έναρξη του project χρησιμοποιήθηκε το CLI της βιβλιοθήκης Quasar που αναφέρεται και στο κεφάλαιο 2.8. Το CLI (Command Line Interface) είναι ένα εργαλείο που προσφέρει η βιβλιοθήκη για την εκκίνηση νέων project. Η δημιουργία γίνεται βάσει των επιλογών του χρήστη και το αποτέλεσμα του είναι ένας φάκελος με μία συγκεκριμένη δομή αρχείων η οποία ομαδοποιεί όλα τα αρχεία τα κώδικα με τον τρόπο που η βιβλιοθήκη μπορεί να τα χειρίζεται. Επίσης στο τελικό αποτέλεσμα υπάρχουν εγκατεστημένες όλες οι απαραίτητες βιβλιοθήκες που ο χρήστης επέλεξε κατά την εκτέλεση του CLI. Για την εκκίνηση του CLI, θα χρειαστεί να εκτελεστεί η εντολή `yarn create quasar` σε μια γραμμή εντολών. Κατά την εκτέλεση ο χρήστης θα ερωτηθεί για βασικές πληροφορίες του project όπως ο φάκελος του, το όνομα, το όνομα του συγγραφέα κλπ. Μετά ακολουθούν ερωτήσεις για τις τεχνολογίες που θα εισάγει, πχ για το styling επιλέχθηκε η τεχνολογία SASS, ως βασική γλώσσα επιλέχθηκε η TypeScript, επίσης προτέθηκε και ο State Manager Pinia, η τεχνολογία Axios για τα request στο back-end και ως διαχειριστής πακέτων επιλέχθηκε το Yarn.

Αφότου ολοκληρωθεί η εγκατάσταση όλων των πακέτων αυτών δημιουργείται ο φάκελος με όλα τα απαραίτητα αρχεία. Η μορφής της δομής αυτής παρουσιάζεται στην Εικόνα 6. Από την δομή της εικόνας

η quasar απαιτεί τους φακέλους boot, components, css, layouts, pages και stores τα οποία βρίσκονται όλα κάτω από τον φάκελο src. Από αρχεία δημιουργεί επίσης τα App.vue, quasar.d.ts και shims-vue.d.ts κάτω από τον φάκελο src, αλλά και έξω από αυτόν τα αρχεία eslintrc.js, index.html, package.json, postcss.config.js, tsconfig.json και Read.me. Ακολουθεί η επεξήγηση του καθενός:

- **Φάκελος boot**: Στο φάκελο boot υπάρχουν αρχεία μορφής TypeScript τα οποία τρέχουν κατά την εκκίνηση του front. Μόλις ξεκινήσει ο εξυπηρετητής του front να τρέχει, σε εκείνο το σημείο τρέχουν και όλα τα αρχεία που υπάρχουν στο φάκελο boot. Τα αρχεία αυτά έχουν πρόσβαση στο αντικείμενο Vue, το οποίο χρειάζεται πολλές φορές για την προσθήκη plugins, βιβλιοθηκών ή components τα οποία γίνονται άμεσα προσβάσιμα σε όλα τα αρχεία μορφής Vue. Στον φάκελο αυτό έχει τοποθετηθεί το αρχείο axios.ts (από την ίδια την Quasar κατά την εκτέλεση του CLI) το οποίο αναλαμβάνει την αρχικοποίηση της βιβλιοθήκης Axios, ένα αρχείο custom-auth.ts το οποίο είναι υπεύθυνο για την αρχικοποίηση των απαραίτητων λειτουργιών για το authorization της εφαρμογής καθώς επίσης και το αρχείο socket_io.ts το οποίο είναι υπεύθυνο για την τεχνολογία SocketIO που χρησιμοποιείται στην εφαρμογή για την αποστολή μηνυμάτων, την εκκίνηση και ειδοποίηση των κλήσεων αλλά και για την αυτόματη ενημέρωση πληροφοριών σε πραγματικό χρόνο.
- **Φάκελος components**: Στον φάκελο αυτό βρίσκονται τα επαναχρησιμοποιούμενα αρχεία μορφής Vue, τα οποία είναι τα component. Εδώ υπάρχει ο μεγαλύτερος αριθμός αρχείων μέσα στο project καθώς αποτελεί και το μεγαλύτερο κομμάτι της εφαρμογής. Κάποια από τα αρχεία αυτά είναι components όπως η λίστες των φίλων, το chat, οι διάλογοι που υπάρχουν στην εφαρμογή κ.α.
- **Φάκελος css**: Εδώ υπάρχουν δύο αρχεία τα οποία δημιουργεί το CLI της Quasar. Αυτά είναι το app.scss και quasar.variables.scss. Στο πρώτο, αρχικά δεν υπάρχει κάτι, αλλά ο χρήστης μπορεί να προσθέσει εδώ όλες τις μορφοποιήσεις του styling και αυτά να έχουν πρόσβαση σε όλη την εφαρμογή. Ουσιαστικά είναι το σημείο όπου τα styles έχουν global ορατότητα και αφορούν όλα τα css του project. Όπως προδίδει και η κατάληξη του αρχείο η μορφή των styles εδώ είναι τύπου SCSS, ωστόσο ο χρήστης μπορεί να συντάξει κανονικά και σε μορφή CSS. Το αρχείο quasar.variables.scss έχει κάποιες συγκεκριμένες μεταβλητές CSS η οποίες έχουν global πρόσβαση σε όλα τα styles. Στις μεταβλητές αυτές αποθηκεύονται τα βασικά χρώματα της εφαρμογής και μπορεί κανείς να τα επεξεργαστεί από εδώ και να επηρεάσει όλη την θεματική της. Οι 3 βασικές μεταβλητές χρωμάτων είναι η `$primary`. `$secondary` και `$accent`. Μετά ακολουθούν οι δευτερεύουσες όπως `$positive`, `$negative`, `$info` και `$warning`. Όλες αυτές είναι άμεσα συνδεδεμένες με τα components που προσφέρει η Quasar αλλά μπορούν επίσης να χρησιμοποιηθούν και από τις μορφοποιήσεις των CSS που δημιουργεί και ο χρήστης. Πχ με την χρήση της συνάρτησης `var(--q-primary)` μπορεί να αποδοθεί οπουδήποτε ο χρωματισμός που έχει αποθηκευτεί στην μεταβλητή `$primary`.
- **Φάκελος layouts**: Ένα πολύ βασικό κομμάτι της γενικής δομής ενός Quasar Project αποτελούν τα layouts. Αρχικά στον φάκελο αυτό υπάρχει το MainLayout.vue αρχείο, το οποίο τοποθετείται από την Quasar. Ωστόσο δεν εξυπηρετεί κάπως στο layout αυτό την εφαρμογή. Έτσι για την βασική μορφή του layout δημιουργήθηκε ένα νέο αρχείο με την ίδια ονομασία στο οποίο υπάρχει το αριστερό μενού της εφαρμογής όπως φαίνεται και στο mock up στην Εικόνα 4 και στην Εικόνα 5. Ουσιαστικά αποτελούν ένα σταθερό κομμάτι UI το οποίο δεν αλλάζει κατά την πλοήγηση του χρήστη στις διάφορες σελίδες της εφαρμογής. Ωστόσο υπάρχει και ένα δεύτερο layout που χρησιμοποιεί η εφαρμογή και αυτό είναι το layout του των σελίδων που αφορούν το authorization. Σε αυτό δεν υπάρχει το αριστερό μενού αλλά κάποια γραφιστικά τα οποία είναι κυρίως για την αισθητική των σελίδων αυτών. Περισσότερα για την εμφάνιση των layout παρουσιάζονται με εικόνες στις φωτογραφίες του κεφαλαίου 7.1 και 7.2.
- **Φάκελος pages**: Στον φάκελο pages υπάρχουν όλες οι σελίδες της εφαρμογής. Είναι μορφής vue και ουσιαστικά αποτελούν components τα οποία έχουν την μόνη διαφορά πως χρησιμοποιούνται από τον router της Vue. Δηλαδή όλα τα αρχεία σε αυτόν το φάκελο αντιστοιχίζονται σε μία διαδρομή της εφαρμογής. Οι σελίδες είναι:
- **Index.vue:** Που εμφανίζεται στην αρχική σελίδα της εφαρμογής.

- **GroupPage.vue:** Χρησιμοποιείται για την αναπαράσταση της σελίδας που απεικονίζει μία ομάδα στην οποία είναι μέλος ο χρήστης.
- **FriendPage.vue:** Παρόμοια με την GroupPage.vue παρουσιάζει τις πληροφορίες ενός φίλου του χρήστη.
- **Και ErrorPage.vue:** Η οποία χρησιμοποιείται ως σελίδα λανθασμένης πρόσβασης και εμφανίζεται όταν ο χρήστης προσπαθεί να πλοηγηθεί σε σελίδα που δεν υπάρχει.
- **Authorization Pages:** Οι σελίδες αυτές αφορούν το login, το register, την σελίδα ενεργοποίησης λογαριασμού, την σελίδα επαναφοράς κωδικού, κ.α.
- **Φάκελος router**: Δυο πολύ βασικά αρχεία της εφαρμογής βρίσκονται σε αυτόν το φάκελο. Το ένα είναι το index.ts, το οποίο αρχικοποιεί τον router της Vue. Όλες οι λειτουργίες και ρυθμίσεις του router υλοποιούνται και τοποθετούνται εδώ. Όπως για παράδειγμα οι έλεγχοι προστασίας σχετικά με το authorization του χρήστη, όπου κάθε φορά με κάθε αλλαγή σελίδας βάσει του route της εφαρμογής, εκτελείται ο έλεγχος για το αν ο χρήστης είναι συνδεδεμένος στην εφαρμογή. Σε περίπτωση που δεν είναι δρομολογείται στο login. Το δεύτερο αρχείο είναι το routes.ts. Σε αυτό υπάρχει ένας πίνακας με όλες οι διαδρομές (paths) της εφαρμογής που έχει πρόσβαση ο χρήστης. Κάθε στοιχείο διαδρομής του πίνακα αντιστοιχίζεται σε ένα αρχείο σελίδας, και κάθε ένα από αυτά μπορεί επίσης να έχει τις ρυθμίσεις του. Πχ κάθε route μπορεί να έχει ένα δικό του έλεγχο ασφαλείας, έτσι κάθε φορά προτού προσπαθήσει κάποιος να μπει σε ένα route, θα τρέξει ο έλεγχος και ανάλογα το αποτέλεσμα που θα βγάλει, θα δώσει ή θα απορρίψει την πρόσβασή του χρήστη στην σελίδα αυτή. Μια ακόμα πολύ χρήσιμή ιδιότητα που μπορεί να δοθεί σε κάθε route είναι το ποιο layout θα χρησιμοποιεί. Έτσι όλες οι σελίδες του authorization ακολουθούν το layout τους και όλες οι υπόλοιπες το βασικό layout της εφαρμογής. Τέλος, εδώ αντιστοιχίζεται και η σελίδα λανθασμένης πρόσβασης στο route `` `/:catchAll(.*)*` ``, το οποίο εμφανίζεται ουσιαστικά σε όλα τα routes που ο χρήστης προσπαθεί να εισέλθει ενώ δεν έχει οριστεί στον πίνακα των routes.
- **Φάκελος stores**: Εδώ υλοποιούνται όλα τα stores που μπορεί να χρησιμοποιεί η εφαρμογή με την βιβλιοθήκη Pinia. Συγκεκριμένα εδώ υπάρχουν τα stores που αφορούν τον συνδεδεμένο χρήστη (current user), τα call stores που ασχολούνται με τις κλήσεις, καθώς και τα group & friend stores που ασχολούνται με τα δεδομένα των ομάδων αλλά και των φίλων του συνδεδεμένου χρήστη. Όλα αυτά τα stores είναι υπεύθυνα για την δημιουργία, αποθήκευση και τροποποίηση σε όλα τα δεδομένα της εφαρμογής με συνεχή επικοινωνία με το back-end, χρησιμοποιώντας hooks που υλοποιούνται βάσει του πακέτου Axios.
- **Αρχείο App.vue**: Το App.vue είναι ένα πολύ βασικό αρχείο για όλες τις εφαρμογές που υλοποιούνται σε Vue, ασχέτως της βιβλιοθήκης Quasar. Είναι το root αρχείο της δομής της Vue και ο router ξεκινάει πάντα από εδώ. Η Quasar χρησιμοποιεί αυτό το αρχείο για να τοποθετήσει τα layout της εφαρμογή εδώ. Οπότε ανάλογα το route αλλάζει και το layout component το οποίο εμφανίζεται στο App.vue αρχείο. Επίσης, οποιαδήποτε μορφοποίηση CSS τοποθετηθεί εδώ (χωρίς την χρήση του scoped) θεωρείται global και όλα τα components μπορούν να την υιοθετήσουν. Μία ακόμη χρηστικότητα που έχει είναι το ότι μπορεί να λειτουργήσει ως boot αρχείο. Δηλαδή εδώ μπορεί κανείς να αρχικοποιήσει κάποια εξωτερική βιβλιοθήκη όπως ακριβώς γίνεται και στα boot αρχεία ή γενικότερα να τρέξει κάποιες απαραίτητες λειτουργίες που μπορεί να απαιτεί η κάθε εφαρμογή. Η εσωτερική δομή του αρχείου είναι όπως αυτή ενός component και αποτελείται από το template, το script και τα styles.
- **Αρχεία quasar.d.ts & shims-vue.d.ts**: Γενικότερα τα αρχεία με κατάληξη ".d.ts" βοηθούν στην απόδοση τύπων της TypeScript σε δεδομένα της JavaScript. Σε αυτά μπορεί να οριστούν δομές και τύποι που είναι χρήσιμα για εφαρμογές που υλοποιούνται σε TypeScript αλλά μπορεί ταυτόχρονα να χρησιμοποιούν εξωτερικές βιβλιοθήκες που έχουν υλοποιηθεί σε JavaScript. Συγκεκριμένα ωστόσο το αρχείο quasar.d.ts έχει τύπους και δομές που αφορούν άμεσα λειτουργίες, συναρτήσεις, μεταβλητές και components της βιβλιοθήκης. Και το αρχείο shims-vue.d.ts αφορά την Vue και επιτρέπει στην TypeScript να κατανοήσει την δομή των αρχείων της Vue καθώς επίσης να ενισχύσει και την σύνδεση των τύπων της με αυτές της Quasar.
- **Αρχείο eslintrc.js:** Συνήθως ένα project της Vue χρειάζεται κάποιο εξωτερικό εργαλείο για να ορίζεται μία ομοιόμορφη δομή συγγραφής στον κώδικα έτσι ώστε να είναι εύκολο να βρίσκονται λάθη και προβλήματα προτού χρειαστεί να τρέξει ο κώδικας. Για αυτόν το λόγο

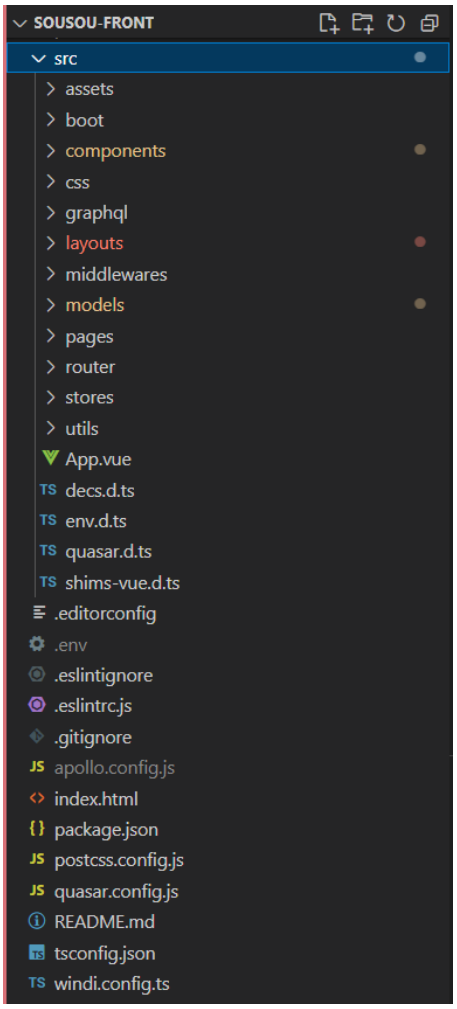
χρησιμοποιούνται βιβλιοθήκες όπως ο eslint. Αυτό που προσφέρει είναι, μια πληθώρα από σετ κανόνων, όπου συνήθως ορίζονται από τις κοινότητες που τα χρησιμοποιούν, τα οποία έχουν ως σκοπό να εμφανίζουν μηνύματα λάθους ή προειδοποιήσεις πριν αρχίσει η εκτέλεση του κώδικα. Ένα αρχείο eslintrc.js έχει μέσα διάφορες ρυθμίσεις που το αφορούν αλλά και τα σετ των κανόνων που θέλει να χρησιμοποιήσει ο καθένας. Είναι πλήρως παραμετροποιήσιμο και ο μπορεί κανείς να το κάνει όσο «αυστηρό» ή «επιεικές» θέλει. Επίσης, έχοντας επιλέξει σαν περιβάλλον συγγραφής το VS Code, μπορεί να χρησιμοποιηθεί και το επίσημο plugin του eslint, το οποίο επιτρέπει κατά την συγγραφή του κώδικα να εμφανίζονται όλα τα μηνύματα σφαλμάτων και προειδοποιήσεων σε πραγματικό χρόνο.

- **Αρχείο index.html:** Στο αρχείο αυτό υπάρχει η κλασική δομή μιας σελίδας html και σε αυτά ορίζονται οι ετικέτες head και body. Δεν υπάρχει κάποια συγκεκριμένη παραμετροποίηση την οποία μπορεί κανείς να κάνει εδώ, χρησιμοποιείται αποκλειστικά από την Quasar για να εκκινήσει την εφαρμογή (mount) μέσω μιας ετικέτας div με γνώρισμα `id="q-app"`. Ωστόσο, αν για κάποιο λόγο χρειάζεται να προστεθεί δυναμικά στην εφαρμογή κάποιο JavaScript ή CSS αρχείο, γίνεται με την χρήση της -γνωστής από την HTML- ετικέτας «link».
- **Αρχείο package.json:** Όπως σε κάθε υλοποίηση μιας JavaScript εφαρμογής έτσι και εδώ υπάρχει το βασικότερο αρχείο package.json. Το αρχείο αυτό χρησιμοποιείται για να δηλώσει την ονομασία και την έκδοση της εφαρμογής, το είδος σύνταξης της JavaScript, και άλλες πολύ βασικές πληροφορίες που την αφορούν. Το κυριότερο κομμάτι της είναι ο ορισμός των πακέτων μέσω του package manager. Στην συγκεκριμένη περίπτωση ο επιλεγμένος package manager είναι το Yarn. Εκτελώντας την εντολή yarn add και μετά την ονομασία του πακέτου (προαιρετικά και την έκδοση) προστίθεται το πακέτο στην εφαρμογή και καταγράφεται στο αρχείο package.json. Κατά την δημιουργία ενός Quasar Project, η βιβλιοθήκη προσθέτει ένα μεγάλο σύνολο πακέτων που είναι απαραίτητο για την εφαρμογή, πχ: Axios, Pinia, Eslint κλπ. Σε αυτά μπορεί κανείς να δει, να κατεβάσει ή να ενημερώσει τις εκδόσεις των πακέτων πάλι μέσω του package manager.
- **Αρχείο quasar.config.js**: Το αρχείο αυτό είναι υπεύθυνο για διάφορες ρυθμίσεις που αφορούν την Quasar. Για παράδειγμα, plugins, boot αρχεία που θέλει να χρησιμοποιεί, διάφορες βιβλιοθήκες και συναρτήσεις που προσφέρει η quasar κ.α.
- **Αρχείο tsconfig.json**: Στο αρχείο tsconfig.json ορίζονται διάφορα γνωρίσματα για το πως η TypeScript θα κάνει compile τον κώδικα.

Τα υπόλοιπα αρχεία και φάκελοι που βρίσκονται στο project έχουν δημιουργηθεί μετέπειτα και δεν είναι απαραίτητα σε κάθε υλοποίηση της Quasar. Ωστόσο, είναι απαραίτητα για την δική μας υλοποίηση της εφαρμογής. Συγκεκριμένα έχουμε:

- **Φάκελος assets**: Στον φάκελο αυτό τοποθετούνται επαναχρησιμοποιήσιμα εξωτερικά αρχεία τύπου json, ήχου ή εικόνας. Στην δική μας περίπτωση αποτελείται από τις εικόνες του logo της εφαρμογής.
- **Φάκελος graphql**: Ο φάκελος αυτός περιέχει όλα τα απαραίτητα αρχεία με όλα τα queries και τα mutations που χρησιμοποιεί το front-end για να επικοινωνεί με το back-end, όπως περιγράφονται και στο κεφάλαιο 2.4.
- **Φάκελος middlewares**: Στα middlewares τοποθετούνται τα αρχεία που περιέχουν συναρτήσεις που τρέχουν ως ενδιάμεσοι. Δηλαδή, στην δική μας περίπτωση υπάρχει το αρχείο auth.ts στο οποίο έχει υλοποιηθεί η ο έλεγχος πρόσβασης του χρήστη. Τρέχει σε κάθε αλλαγή του route της εφαρμογής, πράγμα που το κάνει μία ενδιάμεση συνάρτηση του router. Κάθε φορά που εκτελείται γίνεται ο έλεγχος αν ο χρήστης είναι συνδεδεμένος εκείνη την στιγμή ή αν έχει λήξει το token του. Ανάλογα το αποτέλεσμα της συνάρτησης ο router ξέρει τι να κάνει προτού ολοκληρωθεί η δρομολόγηση του χρήστη.
- **Φάκελος models**: Σε αυτόν τον φάκελο τοποθετούνται όλα τα interfaces και τα types που δημιουργήθηκαν και χρησιμοποιούνται στην εφαρμογή.
- **Φάκελος utils**: Ο φάκελος αυτός είναι ένα βοηθητικός φάκελος για μικροσυναρτήσεις που μπορεί να χρησιμοποιούνται σε διάφορα σημεία της εφαρμογής. Πχ: μία συνάρτηση που δημιουργεί μία ειδοποίηση σε περιπτώσεις σφάλματος.

- **Αρχείο windi.config.ts**: Η βιβλιοθήκη της windi απαιτεί για την χρήση της, την δημιουργία ενός αρχείου για την ρύθμιση και παραμετροποίηση των διάφορων κλάσεων και μηχανισμών που προσφέρει, όπως επίσης δίνει και την δυνατότητα της επέκτασης τους.



Εικόνα 6: Δομή Quasar Project

Η υλοποίηση της εφαρμογής έγινε χρησιμοποιώντας την παραπάνω δομή και κατά βάσει προσθέτοντας components σταδιακά. Κάποια από τα components μπορεί να χρειάζεται να επικοινωνούν με το back end για αυτό, χρησιμοποιούν είτε τα stores του pinia, είτε τα sockets. Με τα stores γίνονται ουσιαστικά αιτήματα στο back-end με queries ή mutations ανάλογα την περίπτωση (πχ η λήψη των παλιών μηνυμάτων από την βάση, αποστολή αιτήματος φιλίας κλπ). Και με τα sockets ενημερώνονται «αυτόματα» σε πραγματικό χρόνο οι πληροφορίες των component που τα χρησιμοποιούν (πχ η αποστολή και λήψη νέων μηνυμάτων στο chat). Πέραν των socket και των graphql αιτημάτων στο back-end, υπάρχει και η επικοινωνία μέσω του WebRTC. Την συγκεκριμένη τεχνολογία χρησιμοποιούν τα component παραθύρων κλήσης. Εκεί κατά την εκκίνηση του παραθύρου (και της κλήσης) ξεκινάει η σηματοδότηση μέσω των socket από τον ένα χρήστη στον άλλο. Μόλις ολοκληρωθεί επιτυχώς η σηματοδότηση στην απέναντι πλευρά και αποδεχτεί την κλήση, ξεκινάει η διαδικασία μεταφοράς πληροφορίας μέσω του WebRTC απευθείας από το ένα fron-end στο άλλο. Την ώρα της κλήσης είναι δυνατών κανείς να συνεχίσει να χρησιμοποιεί τόσο τα graphql αιτήματα όσο και τα socket καθώς είναι πλήρως ανεξάρτητα μεταξύ τους. Για την μορφοποίηση των components χρησιμοποιήθηκαν οι βιβλιοθήκες των κλάσεων που προσφέρει η Quasar, καθώς και η Windi.

Κεφάλαιο 6ο: Deploy Εφαρμογής

Σε αυτό το κεφάλαιο θα δούμε πως έγινε το deploy της εφαρμογής και πως αυτό βοήθησε στην γενική ανάπτυξή της. Μετά την υλοποίηση κάποιων βασικών λειτουργιών της εφαρμογής, αποφασίσαμε να την φιλοξενήσουμε σε κάποιον εξυπηρετητή ώστε να χρησιμοποιηθεί από φίλους και γνωστούς, και να ανακαλύψουμε τυχών bugs. Εκτός από την παραγωγή, φιλοξενήθηκε και μία έκδοση ανάπτυξης, ώστε να μην χρειάζεται να τρέχει τοπικά το backend κατά την ανάπτυξη του frontend. Επίσης, η έκδοση ανάπτυξης χρησιμοποιήθηκε και ως περιβάλλον δοκιμών, καθώς, πριν την κυκλοφορία κάποιας νέας λειτουργίας ή και επίλυση κάποιου bug, πρώτα περνούσε από αυτήν την έκδοση για να δοκιμαστεί.

6.1 Linux Server

Η ανάπτυξη της περιλαμβάνει μια σχολαστική διαδικασία ρύθμισης και διαμόρφωσης ενός διακομιστή Linux για να διασφαλιστεί ένα ισχυρό και αποτελεσματικό περιβάλλον φιλοξενίας. Η επιλογή του Linux ως λειτουργικού συστήματος για τη φιλοξενία καθοδηγείται από τη δημοφιλή σταθερότητα, ασφάλεια και ευελιξία του. Η επιλεγμένη έκδοση Linux είναι η Ubuntu 18.04.6. Επιλέχθηκε για τον μεγάλο κύκλο υποστήριξης που προσφέρεται, για τον εκτενή διαχειριστή πακέτων και κυρίως για την απέραντη υποστήριξη από την κοινότητα.

6.2 Εγκατάσταση εργαλείων

Για την σωστή εκκίνηση και διαχείριση της εφαρμογής, χρειάζονται πολλά κομμάτια τα οποία δεν είναι εγκατεστημένα κατευθείαν στο Linux. Έτσι, έπρεπε να εγκαταστήσουμε διάφορα πακέτα και υπηρεσίες που χρησιμοποιεί η εφαρμογή, όπως η βάση δεδομένων, η γλώσσα προγραμματισμού της εφαρμογής κλπ. Η εγκατάστασή τους έγινε με την εντολή `apt insta ${package_name}` και τα πακέτα αυτά είναι:

- **MySQL:** Η βάση δεδομένων τρέχει τοπικά στον server που φιλοξενεί και την διαδικασία του backend
- **Nginx:** Προσφέρει λειτουργίες load balacning, web serving και reverse proxy
- **Node.js:** Γλώσσα προγραμματισμού της εφαρμογής
- **PM2:** Διαχείριση του backend process
- **npm:** Διαχειριστής πακέτων της Node.js, απαραίτητος για την ανάκτηση των πακέτων που χρησιμοποιεί η εφαρμογή.

6.3 Environment Variables

Για την εκκίνηση της εφαρμογής, έπρεπε αρχικά να ετοιμαστεί το περιβάλλον της, δηλαδή να οριστούν οι κατάλληλες τιμές στης μεταβλητές περιβάλλοντος (environment variables) ώστε να μπορεί να λειτουργήσει. Τα παραδείγματα αυτών των μεταβλητών βρίσκονται στο αρχείο .env.example το οποίο υπάρχει στο repository του backend ώστε όποιος επιθυμεί να τρέξει την εφαρμογή, να γνωρίζει τι μεταβλητές πρέπει να ορίσει. Οι τιμές των μεταβλητών, αποθηκεύονται στο αρχείο .env. Οι απαραίτητες μεταβλητές για την εφαρμογή είναι:

- **DB_HOST:** Αποθηκεύει το url που φιλοξενεί την MySQLβάση δεδομένων
- **DB_NAME:** Αποθηκεύει το όνομα της βάσης δεδομένων της εφαρμογής
- **DB_USER:** Αποθηκεύει το username του χρήστη της βάσης δεδομένων που είναι υπεύθυνος για την διαχείριση των δεδομένων της εφαρμογής.
- **DB_PASSWORD:** Αποθηκεύει τον κωδικό του χρήστη της βάσης δεδομένων
- **DB_PORT:** Αποθηκεύει την πόρτα (port) που χρησιμοποιεί η βάση δεδομένων στο server που φιλοξενείται.

- **PORT:** Αποθηκεύει την πόρτα που θα χρησιμοποιεί η διεργασία της εφαρμογής στον server, για την επικοινωνία της με τους client.
- **FRONT_URL:** Αποθηκεύει το url του front end της εφαρμογής ώστε να στέλνει links στα emails.
- **EMAIL_PORT:** Αποθηκεύει την πόρτα που χρησιμοποιεί το email service για να δέχεται events και να στέλνει τα emails.
- **EMAIL_HOST:** Αποθηκεύει το url του server που φιλοξενεί το email service.
- **EMAIL_USERNAME:** Αποθηκεύει την ηλεκτρονική διεύθυνση που θα στέλνει τα emails στους χρήστες
- **EMAIL_PASSWORD:** Αποθηκεύει τον κωδικό της ηλεκτρονικής διεύθυνσης.
- **SECONDS_FOR_LOGOUT:** Αποθηκεύει τα νανοδευτερόλεπτα που χρειάζεται για να φανεί logedout ο χρήστης, από όταν αποσυνδεθεί το socket.
- **PRIVATE_KEY:** Αποθηκεύει το ιδιωτικό ssh κλειδί ώστε να παράγει τα JWT του authorization και των emails.

Μετά τον ορισμό των κατάλληλων μεταβλητών, το backend είναι έτοιμο να τρέξει.

6.4 DB Migrations

Πριν την εκκίνηση της εφαρμογής, πρέπει δημιουργήσουμε την βάση δεδομένων, και να περάσουμε τους πίνακες και τα πεδία που χρειάζεται η εφαρμογή για να λειτουργήσει. Αυτό επιτυγχάνεται εκτελώντας την εντολή `npm migrate` η οποία εκτελεί το script “npx mikro-orm migration:up”. Αυτό το script είναι υπεύθυνο για να τρέξει όλα τα αρχεία που βρίσκονται στον φάκελο migrations και δεν έχουν εκτελεστεί ακόμα. Τα migration αρχεία περιλαμβάνουν προεγγεγραμμένο sql κώδικα, που συνήθως αποσκοπεί στην προσθήκη/αφαίρεση πινάκων και στην πρόσθεση/αφαίρεση/επεξεργασία των πεδίων τους. Έτσι, το αρχικό npm migrate σχηματίζει την αρχική βάση, και τα επόμενα την επεξεργάζονται, βάση των αναγκών της εφαρμογής.

6.5 PM2

Αφού ορίσαμε τις μεταβλητές περιβάλλοντος και δημιουργήσαμε την βάση δεδομένων, είμαστε έτοιμοι να τρέξουμε την εφαρμογή. Για την διαχείριση της διεργασίας του backend χρησιμοποιήθηκε το PM2, το οποίο δίνει πολλές ευκολίες για την εκκίνηση και την διαχείρηση της διεργασίας. Για την εκκίνηση, χρησιμοποιήθηκε η εντολή `pm2 start npm --name "sousou-api" -- start `. Αυτό που κάνει η συγκεκριμένη εντολή, είναι να δίνει το όνομα "sousou-api" στην διεργασία, και να την εκκινεί χρησιμοποιώντας την εντολή `npm start`. Το npm start εκτελεί το script “ts-node -r tsconfig-paths/register ./src/main.ts” όπου εκκινεί την εφαρμογή.

6.6 Nginx

Ο Nginx είναι ένας web server διαμεσολάβησης, δηλαδή, είναι η λειτουργία που βρίσκεται στην πρώτη γραμμή ενός server και σκοπός του κυρίως είναι η αναδρομολόγηση των request στην κατάλληλη εφαρμογή, στον ίδιο server. Για να δουλέψει χρειάζεται να ορίσεις ένα configuration file για κάθε εφαρμογή στον server. Το configuration του sousou.me είναι:

```
server {
    root /home/sousou1/sousou-front;
    index index.html index.htm index.nginx-debian.html;
    server_name sousou.me www.sousou.me;
    underscores_in_headers on;

    location /graphql {
      proxy_pass http://127.0.0.1:3333;
```

```
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_pass_header store_slug;
      proxy_cache_bypass $http_upgrade;
      proxy_pass_request_headers   on;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
    }
    location /socket.io {
      proxy_pass http://127.0.0.1:3333;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection 'upgrade';
      proxy_set_header Host $host;
      proxy_pass_header store_slug;
      proxy_cache_bypass $http_upgrade;
      proxy_pass_request_headers   on;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
    }

    location / {
        try_files $uri /index.html;
    }
}
```

Απόκομμα Κώδικα 31: Nginx configuration file

Ουσιαστικά, δείχνουμε στον Nginx ότι όταν σε ένα request βλέπει το domain sousou.me, να κειτάξει αυτό το αρχείο. Ύστερα, πρέπει να δωθεί εμφαση στις τρεις τοποθεσίες (locations) που υπάρχουν στο εύρος του server. Τα locations ορίζονται από το υπόλοιπο url του request, πχ το url https://sousou.me/graphql, θέλουμε να αναδρομολογηθεί στο GrapqhQL API. Αυτά τα locations είναι τα εξής:

- /graphql: Εδώ θέλουμε να στείλουμε όλα τα request που έχουν να κάνουν με το GraphQL API. Στο πεδίο proxy_pass περνάμε την τιμή http://127.0.0.1:3333 όπου είναι το localhost και η πόρτα που τρέχει το API.
- /socket.io: Εδώ θέλουμε να στείλουμε όλα τα request που αφορούν τα WebSockets. Στο πεδίο proxy_pass έχουμε πάλι την ίδια τιμή με το /graphql διότι και οι δύο λειτουργίες φιλοξενούνται από την ίδια διαδικασία.
- /: Εδώ θέλουμε να στέλνονται όλα τα request που αφορούν την ανάκτηση HTML. Στο πεδίο try_files αποθηκεύουμε την τιμή του URL και ακολουθεί το index.html όπου βρίσκεται το αρχικό HTML αρχείο.

6.7 Frontend

Η διεπαφή του χρήστη είναι λίγο πιο απλή στην χρήση της στο production. Το μόνο που χρειάζεται, είναι να εγκατασταθούν τα πακέτα που χρησιμοποιεί το frontend, να οριστούν οι σωστές μεταβλητές

περιβάλλοντος και να χτιστεί (build) η εφαρμογή ώστε να μεταφραστούν τα typescript αρχεία σε javascript.

Για την εγκατάσταση των πακέτων, το μόνο που χρειάζεται είναι να τρέξει η εντολή `npm install`. Αυτό κοιτάζει το package.json και το package-lock.json ώστε να εγκαταστήσει τα κατάλληλα πακέτα στην κατάλληλη έκδοση.

Οι μεταβλητές περιβάλλοντος είναι οι εξής:

- **API_URL:** Αποθηκεύει την διεύθυνση του API για τα request που θα κάνει το front-end στο back-end

Για το build της εφαρμογής, θα χρειαστεί να δημιουργηθούν τα απαραίτητα HTML και JavaScript αρχεία από τον πηγαίο κώδικα. Για να γίνει αυτό θα χρειαστεί να τρέξουμε το script του Framework της Quasar «`quasar build`». Αυτό θα μας δώσει τα αρχεία του "χτισμένου" frontend που θα φιλοξενούνται σε έναν φάκελο μέσα στον server της εφαρμογής, τα οποία δίνονται από τον Nginx όταν κάποιος εκτελέσει ένα request στο βασικό url της εφαρμογής.

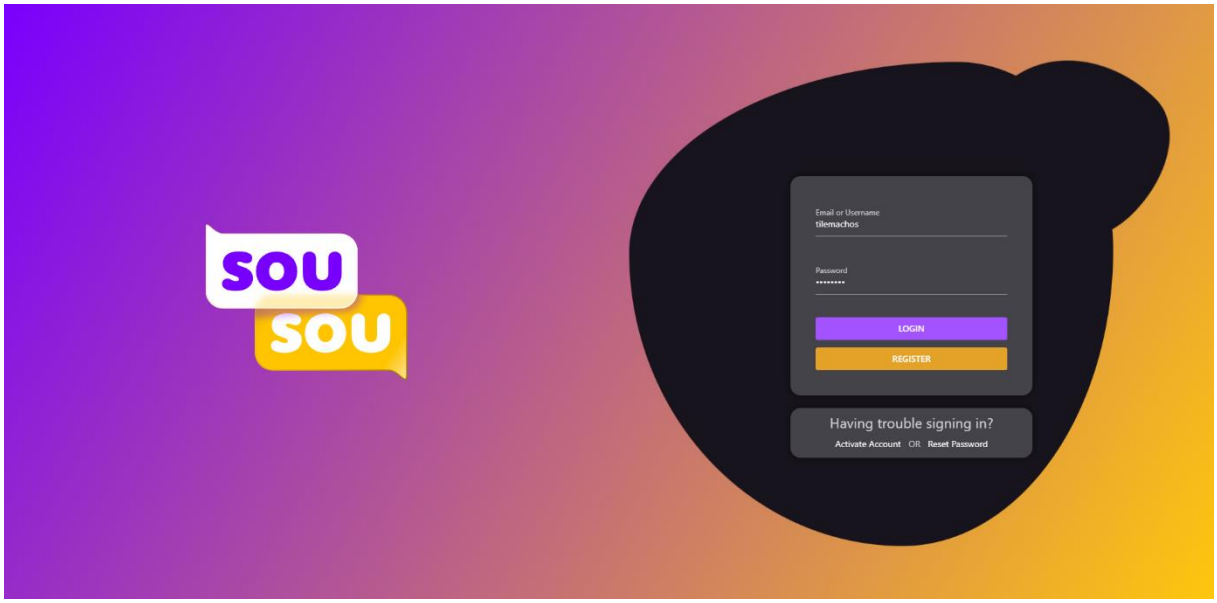
Κεφάλαιο 7ο: Χρήση Εφαρμογής

7.1 Σελίδες Εισόδου (Authorization)

Η πρώτη επαφή του χρήστη με την εφαρμογή είναι στις σελίδες εισόδου ή αλλιώς authorization σελίδες. Οι σελίδες αυτές αναλαμβάνουν την διαχείριση εγγραφής, εισόδου και εξόδου του χρήστη στην εφαρμογή.

7.1.1 Σελίδα Εισόδου (Login)

Η κυριότερη σελίδα είναι αυτή της εισόδου η οποία φαίνεται στην Εικόνα 7. Σε αυτή υπάρχει μια φόρμα η οποία ζητά από τον χρήστη να εισάγει τα στοιχεία εισόδου του. Αφού ο έχει βάλει τα σωστά στοιχεία θα αποκτήσει πρόσβαση στην εφαρμογή και θα δρομολογηθεί στην αρχική σελίδα της εφαρμογής. Σε περίπτωση όπου τα στοιχεία εισόδου δεν είναι σωστά, εμφανίζονται μηνύματα σφάλματος εισόδου στην φόρμα και ζητείται από τον χρήστη να ξανά εισάγει τα στοιχεία του εκ νέου. Από την σελίδα αυτή εκτός από είσοδο στην εφαρμογή ο χρήστη μπορεί να πλοηγηθεί σε 3 ακόμη σελίδες εισόδου. Αυτές είναι: η σελίδα εγγραφής, η σελίδα ενεργοποίησης χρήστη και η σελίδα επαναφοράς κωδικού, οι οποίες αναλύονται στο κεφάλαιο αυτό αναλυτικά.

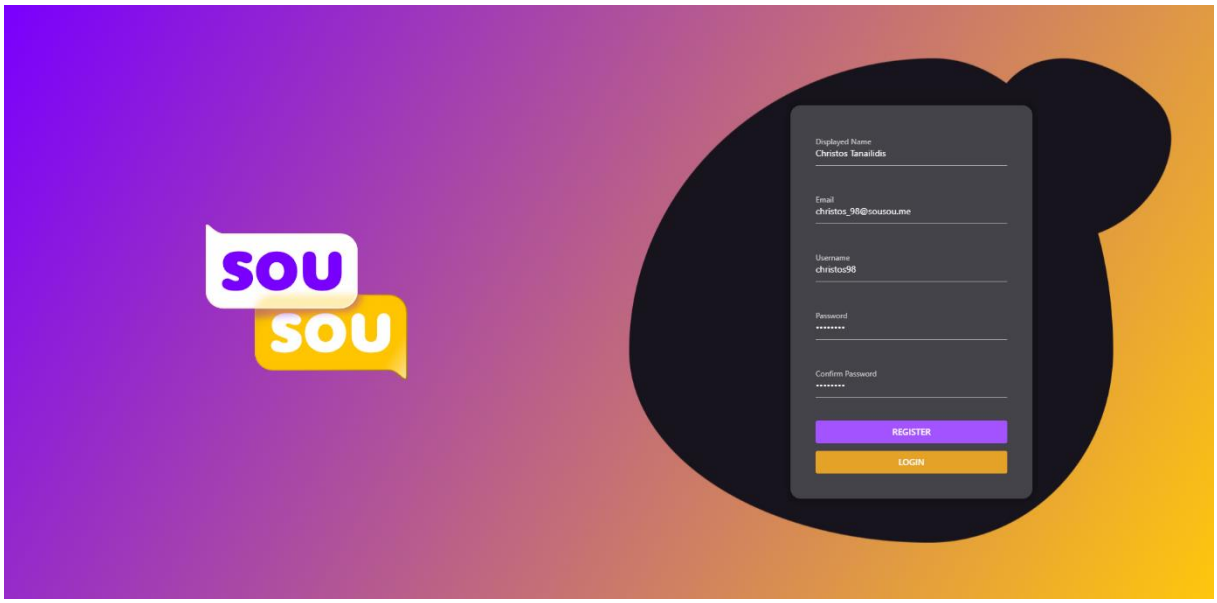


Εικόνα 7: Σελίδα Εισόδου (Login)

7.1.2 Σελίδα Εγγραφής (Register)

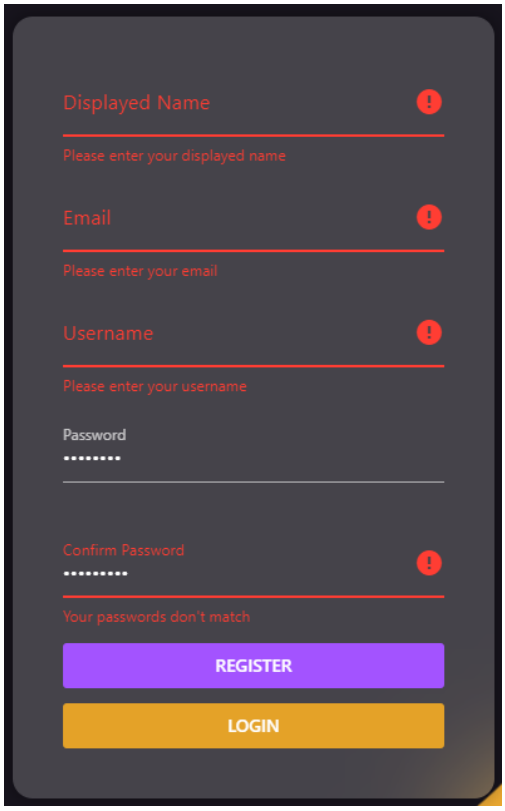
Η σελίδα εγγραφής είναι αυτή που δίνει την δυνατότητα στον χρήστη να εγγραφεί στο σύστημα της εφαρμογής για να μπορέσει αργότερα να μπει σε αυτή. Όπως και στην σελίδα εισόδου έτσι και εδώ υπάρχει μία φόρμα. Η φόρμα αυτή αποτελείται τα στοιχεία εγγραφής του χρήστη. Αυτά είναι:

- Εμφανιζόμενο όνομα χρήστη
- Email
- Username
- Password & Επανάληψη Password



Εικόνα 8: Σελίδα Εγγραφής (Register)

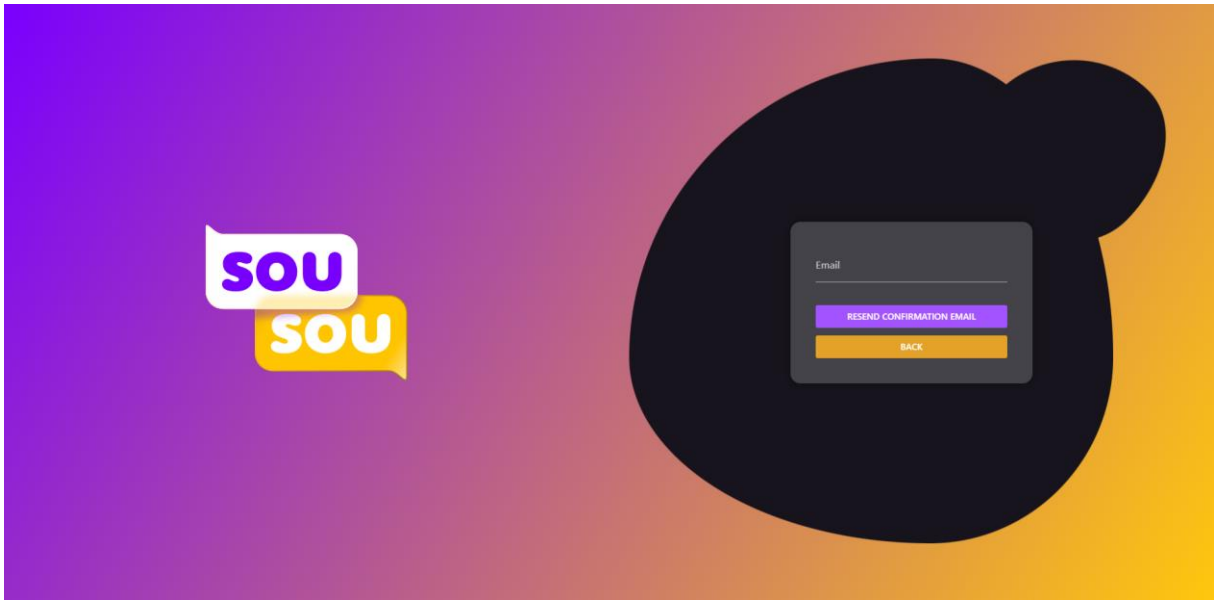
Στην φόρμα αυτή υπάρχουν κανόνες επικύρωσης (validation rules). Αυτό σημαίνει πως κάποια από αυτά τα πεδία ελέγχουν την ορθότητα εισόδου που κάνει ο χρήστης. Όλα τα πεδία της φόρμας είναι υποχρεωτικά. Έτσι σε περίπτωση που κάποιο λείπει εμφανίζεται το κατάλληλο μήνυμα. Επίσης η επανάληψη εισόδου του κωδικού πρόσβασης εμφανίζει μήνυμα λάθους αν οι κωδικοί δεν ταιριάζουν.



Εικόνα 9: Έλεγχος Ορθότητας Δεδομένων Εισόδου Εγγραφής

7.1.3 Επιβεβαίωση Εγγραφής

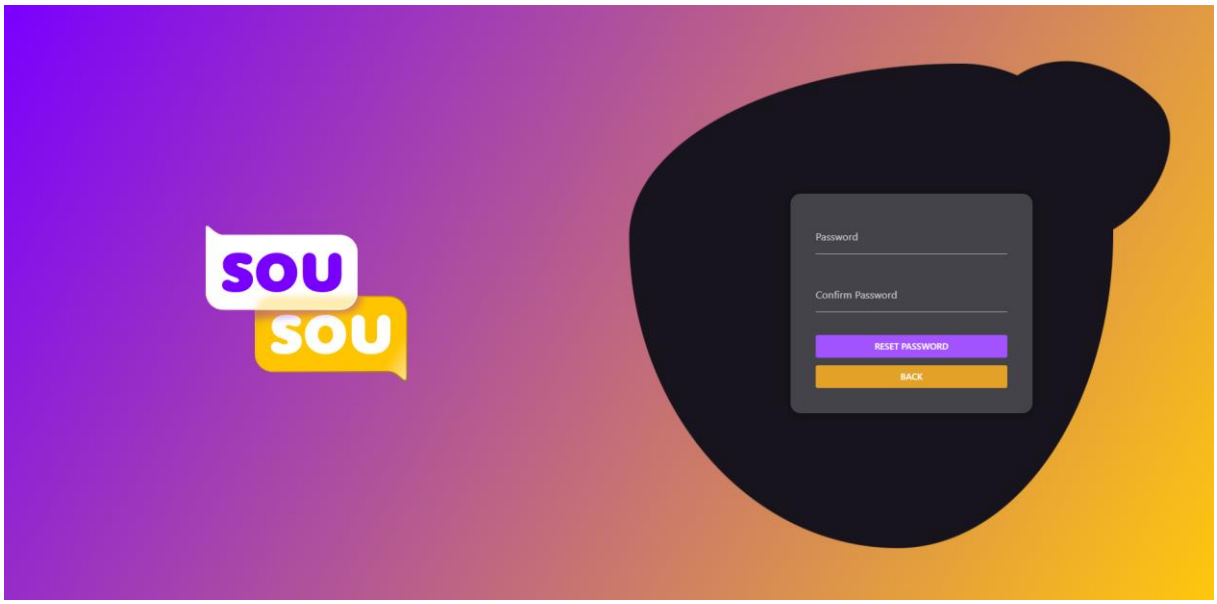
Μόλις ο χρήστης καταφέρει να ολοκληρώσει την εγγραφή του εμφανίζεται ένα μήνυμα πως η εγγραφή του ολοκληρώθηκε και θα λάβει ένα email επιβεβαίωσης. Το email αυτό περιλαμβάνει έναν σύνδεσμο επιβεβαίωσης. Μόλις το πατήσει ο χρήστης πηγαίνει στην σελίδα επικύρωσης χρήστη η οποία επιβεβαιώνει αυτόματα την εγγραφή του χρήστη, τον επαναδρομολογεί στην σελίδα εισόδου και πλέον μπορεί να προχωρήσει στην είσοδο του στην εφαρμογή. Σε περίπτωση που για κάποιο λόγο ο χρήστης δεν έλαβε αυτό το email, ή διαγράφηκε από την αλληλογραφία του, μπορεί να αιτηθεί νέο σύνδεσμο από την σελίδα επαναποστολής συνδέσμου επιβεβαίωσης (Εικόνα 10). Η σελίδα αυτή είναι προσβάσιμη από την σελίδα εισόδου. Σε αυτή ο χρήστης βάζει ξανά το email που χρησιμοποίησε για να εγγραφεί και πατάει αποστολή εκ νέου. Αν το email είναι σωστό και χρησιμοποιήθηκε όντως για εγγραφή θα αποσταλεί ένα νέο email επιβεβαίωσης.



Εικόνα 10: Σελίδα Επαναποστολής Συνδέσμου Επιβεβαίωσης

7.1.4 Επαναφορά Κωδικού

Αν ο χρήστης έχει εγγραφεί επιτυχώς στην βάση και έχει επιβεβαιωμένο λογαριασμό, έχει την δυνατότητα επαναφοράς κωδικού σε περίπτωση που τον ξέχασε. Η σελίδα αυτή είναι προσβάσιμη από την σελίδα εισόδου όπως και η επαναποστολή συνδέσμου επιβεβαίωσης. Ο χρήστης μπορεί να εισάγει το email του και αν αυτό είναι σωστό και έχει ολοκληρωθεί η εγγραφή του, θα λάβει τον σύνδεσμο επαναφοράς. Ο σύνδεσμος θα τον επαναφέρει στην εφαρμογή, συγκεκριμένα στην σελίδα επαναφοράς κωδικού (Εικόνα 11) και θα του ζητηθεί να εισάγει 2 φορές τον κωδικό του (για επιβεβαίωση). Αφού ολοκληρώσει την επαναφορά του κωδικού του πλέον μπορεί να εισέλθει στην εφαρμογή με τον νέο του κωδικό.



Εικόνα 11: Σελίδα Επαναφοράς Κωδικού

7.2 Διεπαφή Χρήστη (User Interface - UI)

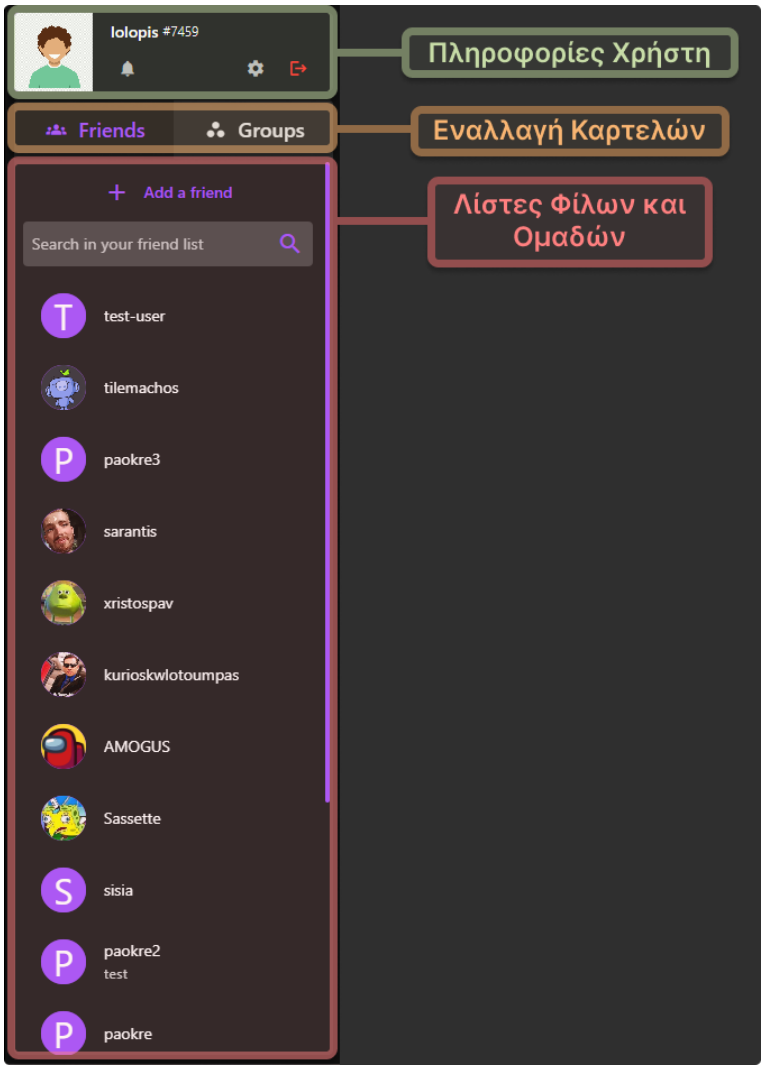
7.2.1 Αρχική Σελίδα

Κατά την είσοδο στην εφαρμογή εμφανίζεται στον χρήστη η αρχική σελίδα. Η αρχική σελίδα δεν περιέχει κάποια πληροφορία παρά μόνο το logo της εφαρμογής. Ωστόσο, μόλις ο χρήστης αποκτήσει πρόσβαση στην εφαρμογή, αλλάζει η διάταξη της σελίδας και του δίνεται η δυνατότητα πλοήγησης από το «μενού πλοήγησης». Το μενού πλοήγησης είναι αυτό που επιτρέπει στον χρήστη να αποκτήσει πρόσβαση και στις υπόλοιπες σελίδες της εφαρμογής, να δει πληροφορίες για τον ίδιο, για τους φίλους του και τις ομάδες στις οποίες είναι μέλος. Επίσης μέσω αυτού μπορεί να πραγματοποιήσει βασικές λειτουργίες της εφαρμογής όπως η επεξεργασία του προφίλ του, να δημιουργήσει αιτήματα φιλίας, ομάδες κ.α. Στο κεντρικό σημείο της εφαρμογής, όπως φαίνεται και στην Εικόνα 12, βρίσκεται η αρχική σελίδα που αλλάζει δυναμικά, μεταξύ όλων των σελίδων της εφαρμογής, μέσω της διεύθυνσης URL. Όταν ο χρήστης θα θελήσει να ανοίξει κάποια συζήτηση φίλου ή ομάδας, όλες οι πληροφορίες και οι ενέργειες της κάθε σελίδας (φίλου ή ομάδας) θα βρίσκονται σε αυτό το σημείο. Σε αντίθεση με το μενού πλοήγησης που βρίσκεται σταθερά στο αριστερό μέρος της εφαρμογής, το υπόλοιπο κομμάτι είναι δυναμικό και εναλλάσσεται όταν ο χρήστης πλοηγείτε στην εφαρμογή.

Εικόνα 12: Αρχική σελίδα εφαρμογής

7.2.2 Μενού πλοήγησης

Το μενού πλοήγησης χωρίζεται σε δύο κύρια σημεία: στα στοιχεία χρήστη μαζί με τις ενέργειες χρήστη, και στις λίστες φίλων και ομάδων. Ο διαχωρισμός έγινε με αυτόν το τρόπο, έτσι ώστε ο χρήστης να έχει απευθείας πρόσβαση στο προφίλ, στις ρυθμίσεις και στις ειδοποιήσεις του ανά πάσα στιγμή, αλλά ταυτόχρονα να μπορεί να πλοηγηθεί με ευκολία στις σελίδες των φίλων και των ομάδων του. Η Εικόνα 13 απεικονίζει την διάταξη του μενού πλοήγησης.



Εικόνα 13: Διαχωρισμός μενού πλοήγησης

7.2.2.1 Πληροφορίες Χρήστη

Οι πληροφορίες χρήστη βρίσκονται στο επάνω μέρος του μενού πλοήγησης και αποσκοπούν στην εμφάνιση χρήσιμων και βασικών πληροφοριών του χρήστη, καθώς επίσης και στην σελίδα των ρυθμίσεων του. Ακολουθεί μια μικρή επεξήγηση του κάθε στοιχείου στο τμήμα των πληροφοριών χρήστη, όπως απεικονίζεται και στην Εικόνα 14.

- **Εικόνα Προφίλ**: Όπως απεικονίζεται και στην Εικόνα 14: Πληροφορίες χρήστηΕικόνα 14 στα αριστερά του τμήματος των πληροφοριών του χρήστη υπάρχει η εικόνα προφίλ που επιλέγει ο χρήστης από τις ρυθμίσεις του. Σε περίπτωση που δεν επιλεγεί κάποια εικόνα αναγράφεται το πρώτο γράμμα του εμφανιζόμενου ονόματος του.
- **Εμφανιζόμενο Όνομα & Κωδικός Χρήστη**: Μετά ακολουθεί το εμφανιζόμενο του όνομα (display name) σε συνδυασμό με τον κωδικό χρήστη (code) ο οποίος ακολουθείται μετά από μία δίεση.
- **Ειδοποιήσεις**: Κάτω από αυτά τα στοιχεία του χρήστη υπάρχει το κουμπί των ειδοποιήσεων. Το κουμπί αυτό απεικονίζεται με ένα καμπανάκι και όταν υπάρχει κάποια ειδοποίηση για αίτημα φιλίας ή πρόσκληση για κάποια ομάδα γίνεται μοβ. Η συγκεκριμένη λειτουργία γίνεται δυναμικά μέσω των socket επικοινωνίας. Αυτό σημαίνει πως ανά πάσα στιγμή όταν ο χρήστης χρησιμοποιεί την εφαρμογή μπορεί να του έρθει κάποια ειδοποίηση, χωρίς να χρειάζεται επαναφόρτιση (refresh) της σελίδας. Πατώντας το εμφανίζεται ένα μικρό μενού στο οποίο ο χρήστης μπορεί να ανοίξει τα παράθυρα των αιτημάτων φιλίας που λαμβάνει ο ίδιος καθώς και

τις προσκλήσεις που του κάνουν σε ομάδες. Από αυτά τα παράθυρα μπορεί να δει, να διαβάσει τα μηνύματα των αιτημάτων και να αποδεχτεί ή να τα απορρίψει ανάλογα.

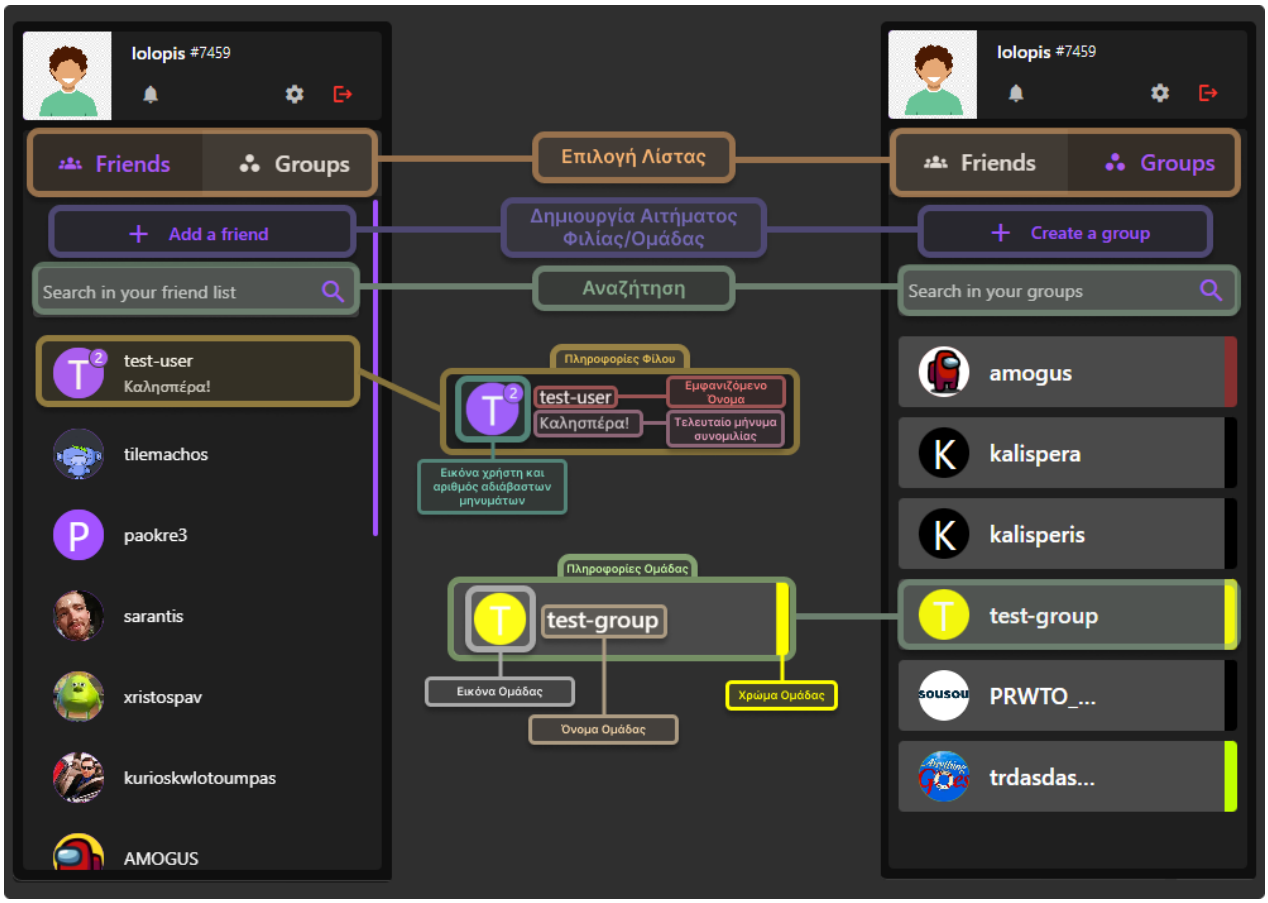
- **Ρυθμίσεις**: Οι ρυθμίσεις χρήστη βρίσκονται στο κουμπί με το γρανάζι το οποίο ανοίγει το παράθυρο των ρυθμίσεων. Το παράθυρο (ρεφερνς εικόνας) χωρίζεται σε δύο μέρη: την πλοήγηση κατηγοριών και τις ρυθμίσεις κατηγοριών. Μόλις ο χρήστης επιλέξει μια κατηγορία από τα αριστερά η σελίδα αλλάζει στο δεξί μέρος. Οι δύο κατηγορίες που υπάρχουν εδώ είναι:
- οι γενικές, που αφορούν το username του χρήστη, το εμφανιζόμενο όνομα του και την εικόνα προφίλ του
- και οι προτιμήσεις του χρήστη, στις οποίες ο χρήστης μπορεί να ρυθμίσει την ένταση του ήχου εισόδου του (μικρόφωνο), την κώφωση μικροφώνου ή ηχείου κ.α. (εικόνες για όλα αυτά).
- **Αποσύνδεση**: Η αποσύνδεση, με κόκκινο χρωματισμό, για να διαφέρει από τα υπόλοιπα κουμπιά, βρίσκεται στην κάτω και δεξιά πλευρά των πληροφοριών χρήστη. Το κουμπί αυτό κατά το πάτημα του, αποσυνδέει τον χρήστη και ουσιαστικά χάνεται η πρόσβαση στην εφαρμογή. Ο χρήστης επαναδρομολογείται στην σελίδα εισόδου και ζητείται νέα είσοδος από αυτόν.

Εικόνα 14: Πληροφορίες χρήστη

7.2.2.2 Λίστες Φίλων και Ομάδων

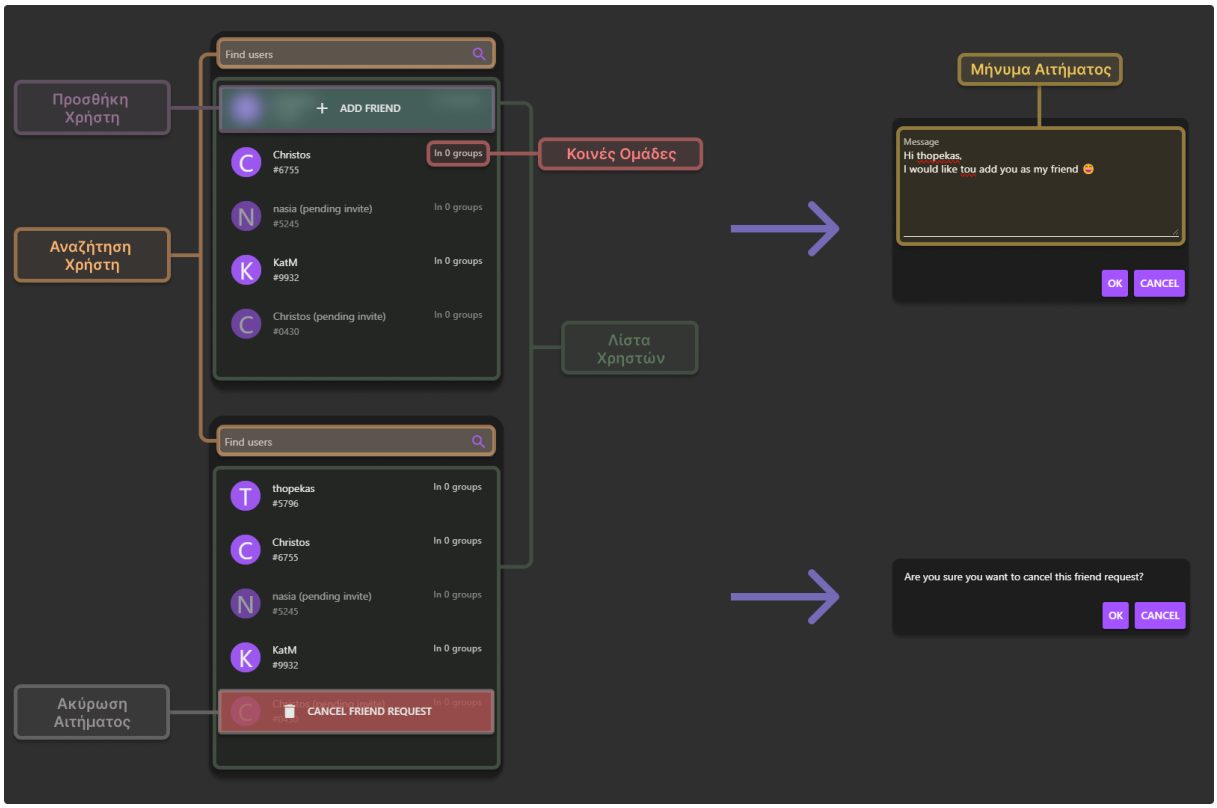
Οι λίστες φίλων και ομάδων (Εικόνα 15) βρίσκονται κάτω από τις πληροφορίες χρήστη. Καταλαμβάνουν το μεγαλύτερο μέρος του μενού πλοήγησης, καθώς αποτελούν και το μέσο κατά το οποίο πλοηγείτε ο χρήστης στα προφίλ των φίλων και των ομάδων του.

- **Επιλογή Λίστας:** Στο επάνω μέρος βρίσκονται σταθερά τα κουμπιά επιλογής της εμφανιζόμενης λίστας. Οι επιλογές αποτελούνται από τους «φίλους» και τις «ομάδες». Ανάλογα με την επιλογή του χρήστη εμφανίζεται και η αντίστοιχη λίστα.



Εικόνα 15: Λίστες Φίλων και Ομάδων

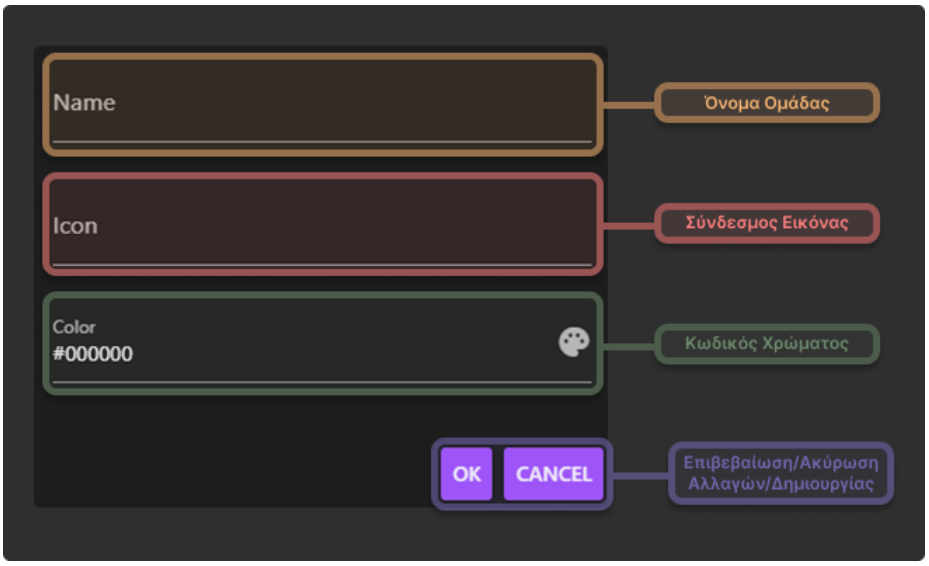
- **Κουμπί προσθήκης φίλου**: Το κουμπί αυτό, ανοίγει το παράθυρο αναζήτησης, στο οποίο υπάρχει δυνατότητα εύρεσης ανάμεσα σε όλους του χρήστες της εφαρμογής που δεν είναι ήδη στην λίστα φίλων του συνδεδεμένου χρήστη (Εικόνα 16). Επίσης σε αυτή τη λίστα μπορεί να δει κανείς και τους χρήστες που έχει αποσταλεί ήδη κάποιο αίτημα φιλίας.
- **Ακύρωση Αιτήματος**: Στη περίπτωση που ο χρήστης έχει ήδη αιτηθεί, τότε δίνεται η επιλογή ακύρωσης αιτήματος, στο οποίο μετά την επιλογή, εμφανίζεται ένα μικρό παράθυρο επιβεβαίωσης προτού ολοκληρωθεί η ακύρωση.
- **Αίτημα Φιλίας**: Στην περίπτωση που δεν έχει αποσταλεί κάποιο αίτημα φιλίας, τότε εμφανίζεται η επιλογή προσθήκης φίλου στην οποία παρουσιάζεται ένα παράθυρο εισαγωγής μηνύματος, στο οποίο ο χρήστης μπορεί να εισάγει προαιρετικά κάποιο μήνυμα για να το λάβει ο παραλήπτης στις ειδοποιήσεις του μαζί με το αίτημα φιλίας.



Εικόνα 16: Παράθυρο Δημιουργίας Αιτήματος Φιλίας

- **Κουμπί Δημιουργίας Ομάδας**: Το κουμπί δημιουργία ομάδας (Εικόνα 17), ανοίγει το παράθυρο δημιουργίας και επεξεργασίας ομάδας στο οποίο ο χρήστης εισάγει τις απαραίτητες πληροφορίες όπως:
- το όνομα της,
- τον σύνδεσμο για την εικόνα της,
- και το κύριο χρώμα της ομάδας.

Στην φόρμα αυτή και τα 3 πεδία είναι απαραίτητα, αλλιώς σε περίπτωση που δεν συμπληρωθούν εμφανίζονται μηνύματα λανθασμένης εισαγωγής για το καθένα και δεν επιτρέπεται η αποστολή του αιτήματος δημιουργίας ομάδας.



Εικόνα 17: Παράθυρο Δημιουργίας/Επεξεργασίας Ομάδας

- **Λίστες Φίλων & Ομάδων**: Οι λίστες φίλων και ομάδων που εμφανίζονται κατά την επιλογή του χρήστη, απεικονίζουν τους όλους τους χρήστες που βρίσκονται στην λίστα φίλων του συνδεδεμένου χρήστη και οι ομάδες, όλες τις ομάδες που έχει δημιουργήσει ο ίδιος καθώς και αυτές που είναι μέλος, όπως φαίνεται και στην Εικόνα 15. Στην λίστα φίλων, κάθε φίλος εκτός από τις βασικές πληροφορίες του όπως η εικόνα και το εμφανιζόμενο του όνομα μπορεί κανείς να δει και το τελευταίο μήνυμα που υπάρχει στην συζήτηση τους. Επίσης σε περίπτωση ο χρήστης-φίλος στείλει νέο μήνυμα στο συνδεδεμένο χρήστη, εμφανίζεται ένα συννεφάκι πάνω από τον χρήστη-φίλο που αναγράφει τον αριθμό των νέων μηνυμάτων. Με αυτόν τον τρόπο μπορεί κανείς να καταλάβει πόσα νέα αδιάβαστα μηνύματα έχει κάποιος από έναν φίλο του. Κατά την επιλογή ενός φίλου από το μενού πλοήγησης, στην δεξιά σελίδα ανοίγει η σελίδα φίλου και κατά την επιλογή ομάδας ανοίγει η σελίδα ομάδας. Περισσότερα για τις σελίδες της εφαρμογής περιγράφονται και αναλύονται στο κεφάλαιο 7.2.3. Ο τρόπος με τον οποίο γίνεται η μετάβαση στην σελίδα αφότου επιλεγεί ο χρήστης, είναι μέσω του router της Vue, ο οποίος χρησιμοποιεί την διεύθυνση URL της εφαρμογής και προσθέτει σημεία διαδρομής (paths) και γνωρίσματα (props), τα οποία μεταφράζουν το τι εμφανίζεται στην κεντρική σελίδας της εφαρμογής. Οπότε με ένα κλικ του χρήστη σε ένα φίλο από το μενού πλοήγησης, η εφαρμογή δρομολογεί την κεντρική σελίδα στην σελίδα φίλου. Ένα παράδειγμα μιας διαδρομής (path) σε μία σελίδα φίλου:

`sousou.me/friend/0230689c-d7c7-4cb3-b0e3-72768a3794fb`

Στο παραπάνω παράδειγμα βλέπουμε πως μετά το `sousou.me/` το οποίο ορίζεται και ως η «αρχική σελίδα», ακολουθεί το σημείο διαδρομής `friends/` και μετά ακολουθεί το γνώρισμα του ID του χρήστη-φίλου που έχει επιλεγεί. Με το σημείο `friends/` ο έχουμε ορίσει πως ο router πρέπει να ανοίξει την σελίδα φίλου και με το γνώρισμα του ID δίνεται η απαραίτητη πληροφορία στην σελίδα για να εμφανίζει τα στοιχεία του συγκεκριμένου χρήστη. Παρόμοια μορφή δομής ισχύει και για τις ομάδες:

`sousou.me/group/9980b7e6-487e-4563-bd6b-d997ac793437`

Η μόνη διαφορά με την προηγούμενη διαδρομή είναι το σημείο διαδρομής `group/` το οποίο είναι και αυτό που εμφανίζει την σελίδα ομάδας αντί αυτή του φίλου.

Υπάρχει επίσης και δυνατότητα της άμεσης πρόσβασης στις σελίδες χωρίς να χρειάζεται η χρήση του UI για πλοήγηση. Δηλαδή αν ο χρήστης εισάγει τις παραπάνω διαδρομές στο URL του περιηγητή ιστού, ενώ είναι συνδεδεμένος και έχει χρησιμοποιεί ID χρήστη που είναι φίλοι ή ID

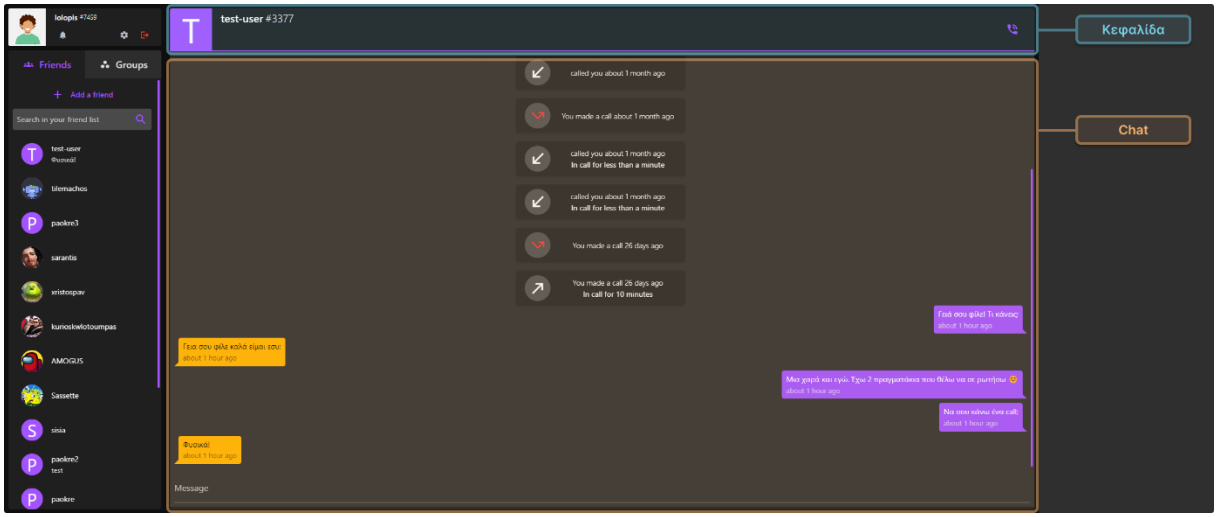
ομάδας που είναι μέλος, τότε θα αποκτήσει απευθείας πρόσβαση στην σελίδα που αναζητά. Σε κάθε άλλη περίπτωση η εφαρμογή εμφανίζει μηνύματα λάθους στον χρήστη.

7.2.3 Σελίδες Εφαρμογής

Οι κύριες σελίδες της εφαρμογής είναι δύο: η Σελίδα προβολής φίλου και η σελίδα προβολής ομάδας. Η κάθε σελίδα εμφανίζει, με ελαφρώς διαφορετική δομή, πληροφορίες για χρήστες και ομάδες αντίστοιχα. Ωστόσο υπάρχει ακόμη μία σελίδα, η σελίδα λανθασμένης πρόσβασης. Εμφανίζεται όταν γίνεται κάποιο λάθος στην διαδρομή του URL της εφαρμογής. Δηλαδή όταν ο χρήστης προσπαθεί να μπει σε κάποια σελίδα η οποία δεν έχει οριστεί στον router της vue.

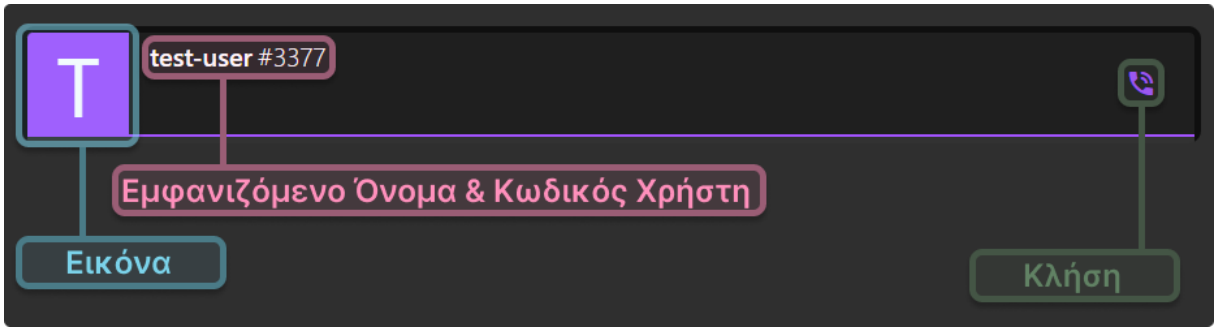
7.2.3.1 Σελίδα Προβολής Φίλου

Η σελίδα φίλου χωρίζεται σε δύο μέρη: την κεφαλίδα πληροφορίων και το chat όπως παρουσιάζεται στην Εικόνα 18.



Εικόνα 18: Σελίδα Προβολής Φίλου

Η κεφαλίδα πληροφοριών αποτελείται από την εικόνα, το εμφανιζόμενο όνομα και τον κωδικό του χρήστη όπως φαίνεται και στην Εικόνα 19. Στην δεξιά μεριά υπάρχει το κουμπί κλήσης μέσω φωνής. Το κουμπί κλήσης είναι αυτό που εκκινεί την αίτηση κλήσης και στέλνει στον παραλήπτη (χρήστη-φίλο) την ειδοποίηση κλήσης. Μόλις πατηθεί το κουμπί κλήσης, εμφανίζεται στον αποστολέα το παράθυρο κλήσης στο οποίο υπάρχουν: Οι πληροφορίες χρήστη όπως η εικόνα και το κείμενο κλήσης, η επιλογές ήχου όπως κώφωση μικροφώνου και η ρύθμιση έντασης ήχου και τέλος το κουμπί τερματισμού κλήσης (1ο μέρος από την Εικόνα 20).

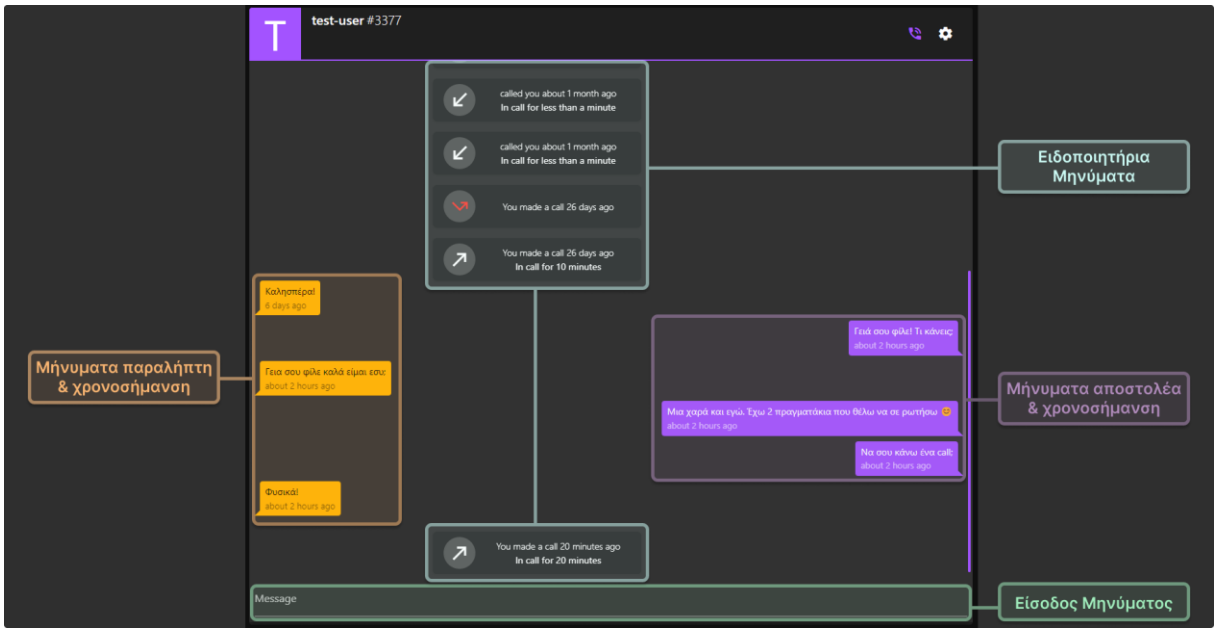


Εικόνα 19: Κεφαλίδα Σελίδας Προβολής Φίλου

Ο παραλήπτης της κλήσης μόλις λάβει την ειδοποίηση κλήσης βλέπει και αυτός το δικό του παράθυρο κλήσης. Η μόνες διαφορές είναι το κείμενο της κλήσης που αναγραφεί πως κάποιος τον καλεί και τα κουμπιά απάντησης και απόρριψης κλήσης (2ο μέρος από την Εικόνα 20). Αν ο χρήστης απορρίψει την κλήση κλείνουν και τα δύο παράθυρα και η κλήση τερματίζεται. Αν ο χρήστης αποδεχτεί το αίτημα της κλήσης, τότε ξεκινάει η επικοινωνία μεταξύ των δύο και τα παράθυρα τους πλέον αποτελούνται από το κείμενο κλήσης που αναγράφει πως οι δύο χρήστες βρίσκονται σε κλήση, τις λειτουργίες ήχου (που είχαν και πριν) και το κουμπί τερματισμού κλήσης (3ο μέρος από την Εικόνα 20). Τα παράθυρα κλήσης είναι ανεξάρτητα της σελίδας φίλου. Αυτό σημαίνει πως αν ο χρήστης έχει την ανάγκη να χρησιμοποιήσει την εφαρμογή και να πλοηγηθεί σε κάποια άλλη σελίδα φίλου ή ομάδας, μπορεί.

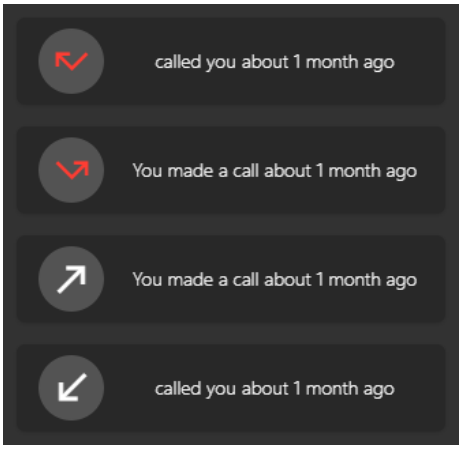
Εικόνα 20: Παράθυρα Κλήσης

Στο κεντρικό κομμάτι της σελίδας φίλου βρίσκεται το chat όπως φαίνεται και στην Εικόνα 21. Στο chat υπάρχουν όλα τα μηνύματα μεταξύ των χρηστών και διάφορα ειδοποιητήρια μηνύματα κλήσεων.



Εικόνα 21: Συνομιλία Φίλου

- Στα μηνύματα χρηστών εμφανίζονται το μήνυμα και η χρονοσήμανση αποστολής. Αν το μήνυμα είναι του συνδεδεμένου χρήστη (αποστολέα), βρίσκεται στα δεξιά της σελίδας. Αν το μήνυμα είναι του χρήστη-φίλου (αποδέκτη), τότε βρίσκεται στα αριστερά της σελίδας. Επίσης διαφοροποιούνται και χρωματικά για να είναι απευθείας κατανοητό από ποιον στάλθηκε.
- Στα ειδοποιητήρια μηνύματα κλήσεων υπάρχει διαφορετική δομή. Βρίσκονται στο κέντρο του chat, έχουν ίδιο χρωματισμό και αποτελούνται από το εικονίδιο τύπου κλήσης και το κείμενο που αναγράφει το πόσο καιρό πριν στάλθηκε το μήνυμα. Τα εικονίδια τύπου κλήσης είναι 4 και απεικονίζονται στην Εικόνα 22. Ο σκοπός τους είναι να διατηρούν το ιστορικό των κλήσεων μέσα στο chat.



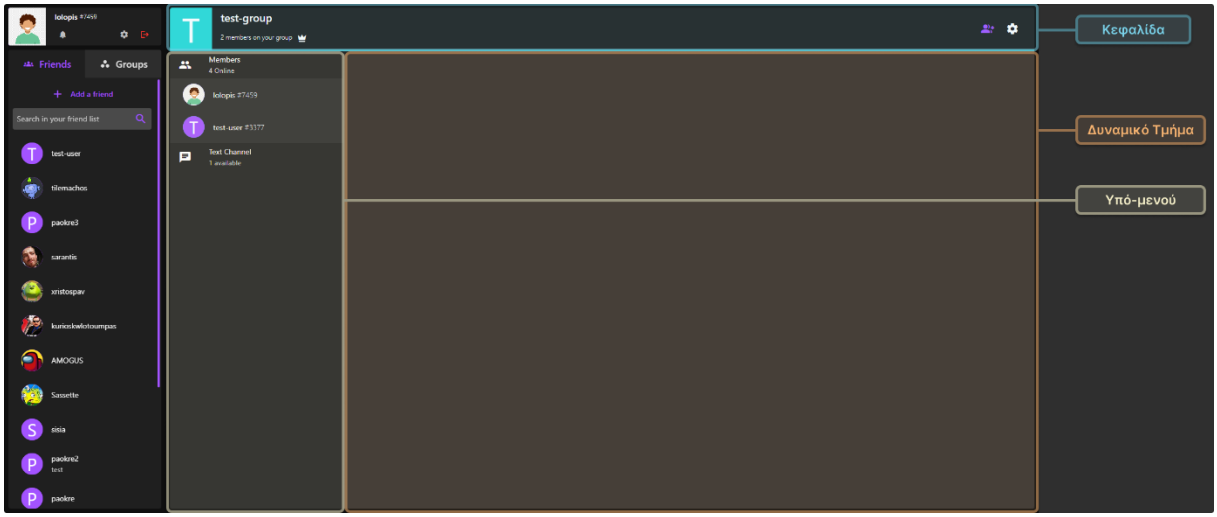
Εικόνα 22: Τύποι ειδοποιητηρίων μηνυμάτων

Όλα τα μηνύματα που υπάρχουν προτού ανοίξει το chat, έρχονται από την βάση δεδομένων σελιδοποιημένα. Ο τρόπος σελιδοποίησης γίνεται με την τεχνική virtual scroll. Η τεχνική αυτή χρησιμοποιεί το scroll του ποντικιού για να φέρει περισσότερα δεδομένα, και ουσιαστικά μόλις το σημείο της σελίδας που έχει γίνει scroll φτάσει σε ένα συγκεκριμένο σημείο (offset) ύψους (στον άξονα y) φέρνει περισσότερα μηνύματα. Τα νέα μηνύματα είναι δυναμικά και προστίθενται κάθε φορά που αποστέλλεται το κάθε ένα. Αυτό επιτυγχάνεται με την χρήση της τεχνολογίας websocket για μετάδοση

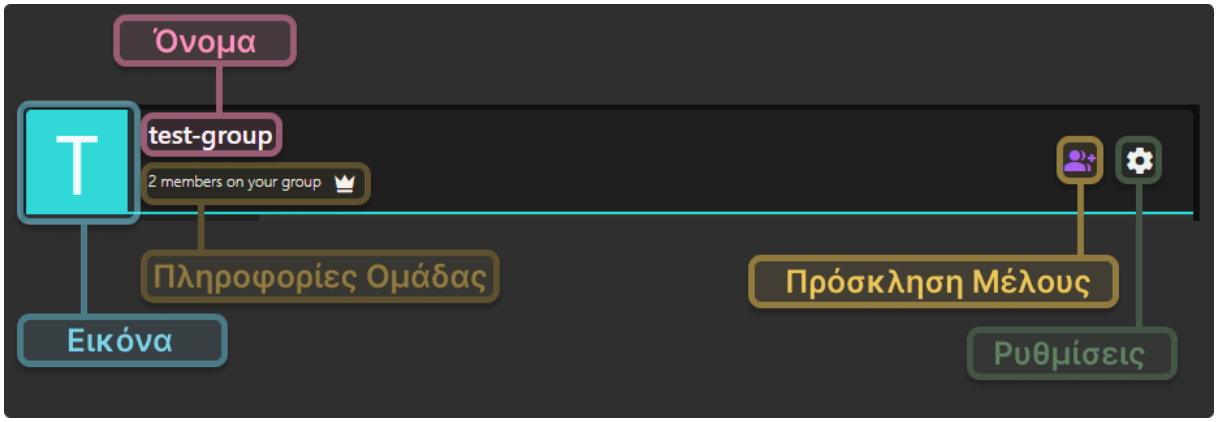
δεδομένων σε πραγματικό χρόνο. Στο κάτω μέρος της σελίδας είναι η είσοδος των μηνυμάτων (text input), όπου εκεί ο χρήστης γράφει το μήνυμα του και πατώντας το enter το αποστέλλει.

7.2.3.2 Σελίδα Προβολής Ομάδας

Οι σελίδες προβολής ομάδας είναι διαφορετικές από αυτές των χρηστών-φίλων. Ενώ χωρίζονται όπως και αυτές των φίλων, οι πληροφορίες και οι λειτουργίες που εμφανίζονται είναι διαφορετικές (Εικόνα 23). Στην κεφαλίδα της σελίδας ομάδας βρίσκετε η εικόνα και το όνομα της. Κάτω από το όνομα υπάρχει η πληροφορία του αριθμού των μελών της και σε περίπτωση που η ομάδα δημιουργήθηκε από το συνδεδεμένο χρήστη, εμφανίζεται ένα εικονίδιο «κορώνας» που το συμβολίζει. Στα δεξιά της κεφαλίδας υπάρχουν το κουμπί προσθήκης μέλους και οι ρυθμίσεις της ομάδας, τα οποία και τα δύο μπορεί να τα δει μόνον ο δημιουργός της. Η προσθήκη μέλους ανοίγει το παράθυρο αναζήτησης χρήστη ίδιο με αυτό της προσθήκης φίλου (Εικόνα 16) καθώς η διαδικασία πρόσκλησης μέλους είναι ίδια με αυτή του αιτήματος φίλιας. Οι ρυθμίσεις της ομάδας εμφανίζονται σε ένα παράθυρο στο οποίο μπορεί κανείς να αλλάξει το όνομα, την εικόνα ή το χρώμα της ομάδας. Όπως και κατά την δημιουργία, έτσι και στην επεξεργασία των δεδομένων της ομάδας όλες οι πληροφορίες είναι υποχρεωτικές.



Εικόνα 23: Σελίδα Προβολής Ομάδας

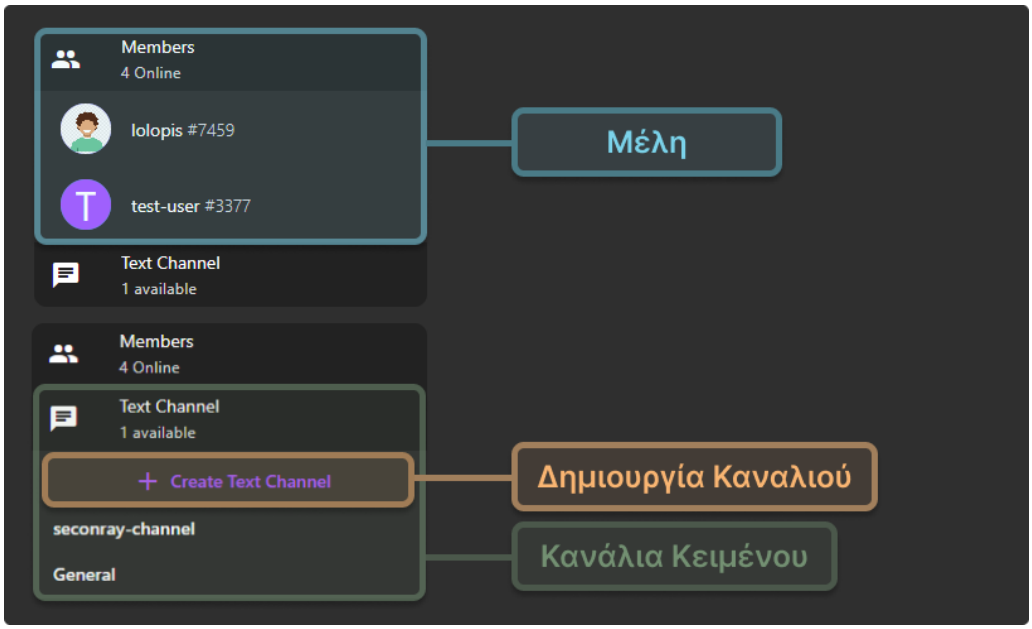


Εικόνα 24: Κεφαλίδα Σελίδας Προβολής Ομάδας

Μια αρκετά σημαντική διαφορά της σελίδας ομάδας με αυτή του φίλου, είναι η κεντρική πληροφορία που εμφανίζεται. Στις ομάδες υπάρχει ένα υπό-μενού πλοήγησης (Εικόνα 23, Εικόνα 25), το οποίο μένει σταθερό σε κάθε σελίδα ομάδας, ενώ το δεξί κομμάτι της σελίδας είναι δυναμικό και εναλλάσσετε

ανάλογα με τις επιλογές που κάνει ο χρήστης στο υπό-μενού. Το υπό-μενού πλοήγησης ομάδας αποτελείται από 2 μέρη:

- Την καρτέλα μελών
- Και τα κανάλια κειμένου (chat)



Εικόνα 25: Υπό-μενού Σελίδας Προβολής Ομάδας

Η κάρτα μελών εμφανίζει όλα τα μέλη της σελίδας, ενώ η κάρτα καναλιών κειμένου εμφανίζει όλα τα ενεργά κανάλια της ομάδας στα οποία κάποιος μπορεί να επικοινωνήσει.

Τα κανάλια κειμένου όταν επιλέγονται από το υπό-μενού, εμφανίζουν στο δεξί κομμάτι της κεντρική πληροφορίας της ομάδας, ένα chat. Το chat αυτό έχει τις ίδιες δυνατότητες με αυτά των σελίδων προβολής φίλου, με την μόνη διαφορά πως αποτελείται από πολλά άτομα. Για την συγκεκριμένη ιδιαιτερότητα, χρειάζεται ένας τρόπος να διαχωρίζονται τα άτομα μεταξύ τους όταν στέλνουν μηνύματα. Για αυτό, υπάρχει και η εικόνα του αποστολέα δίπλα από τα μηνύματα του, καθώς εμφανίζεται και το όνομα χρήστη όταν το ποντίκι περνάει πάνω από το την εικόνα του αποστολέα.

Επίσης πάνω ακριβώς από τα κανάλια κειμένου, υπάρχει το κουμπί δημιουργίας καναλιού. Το συγκεκριμένο εμφανίζεται μονάχα στους δημιουργούς της σελίδας και με αυτό μπορούν να ανοίξουν το παράθυρο δημιουργίας καναλιού. Σε αυτό υπάρχει μία φόρμα με ένα input που ζητά την ονομασία του καναλιού. Μετά την δημιουργία του η λίστα καναλιών ανανεώνεται απευθείας και οι χρήστες της ομάδας μπορούν να ξεκινήσουν να επικοινωνούν μέσω του chat αυτού.

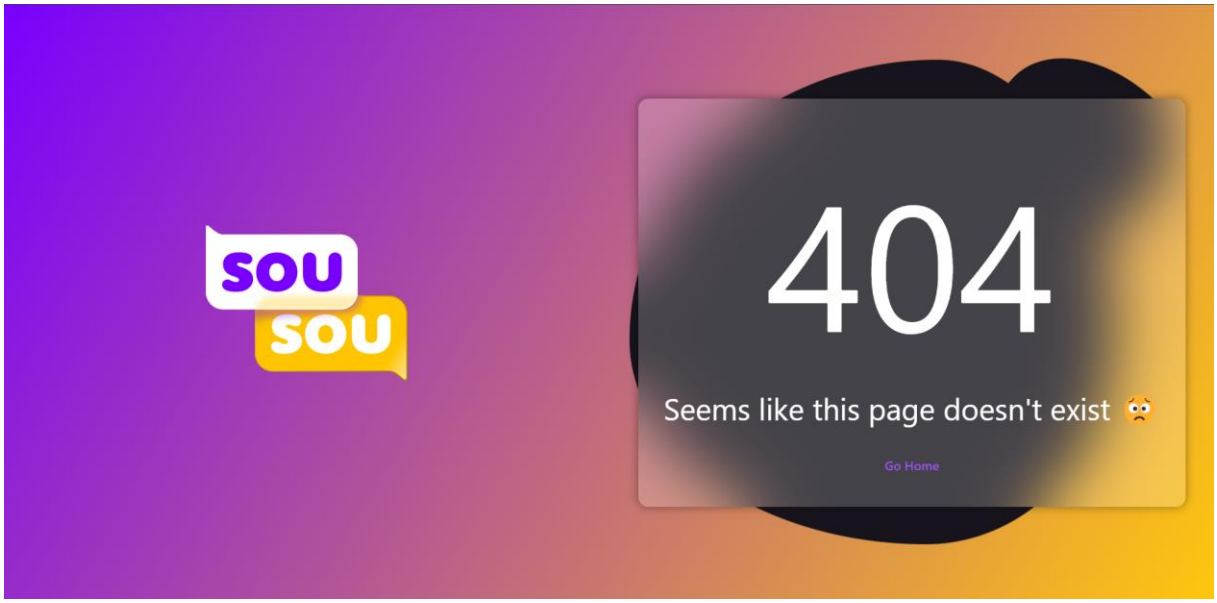
7.2.3.3 Σελίδα Λανθασμένης Πρόσβασης

Η Σελίδα λανθασμένης πρόσβασης εμφανίζεται στον χρήστη όταν αυτός προσπαθεί να μπει σε μία λανθασμένη διαδρομή του URL path. Πχ: αν ο χρήστης θελήσει να μπει στην σελίδα φίλου και κατά την πληκτρολόγηση της διαδρομής γράψει

`https://dev.sousou.me/friends/3bfec39a-3773-4116-baaf-e1f1a87d558d`

τότε δεν θα καταφέρει να μπει στην σελίδα καθώς στο τμήμα καθορισμού σελίδας έπρεπε να γράφει “friend” και όχι “friends”. Οπότε θα χρειαστεί ένας μηχανισμός για τέτοιες περιπτώσεις λάθους, τον οποίο αναλαμβάνει ο router της Vue. Σε περίπτωση που δοθεί διαδρομή, μη εγγεγραμμένη στις

διαδρομές του router, ο router θα επαναδρομολογήσει τον χρήστη στην σελίδα λανθασμένης πρόσβασης (Εικόνα 26).



Εικόνα 26: Σελίδα Λανθασμένης πρόσβασης

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] R. Jones, “Object-Oriented Programming LNCS 8586 ARCoSS.”

[2] J. Bogner and M. Merkel, “To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub,” in *Proceedings - 2022 Mining Software Repositories Conference, MSR 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 658–669. doi: 10.1145/3524842.3528454.

[3] B. Basumatary and N. Agnihotri, “Benefits and Challenges of Using NodeJS,” *International Journal of Innovative Research in Computer Science & Technology*, pp. 67–70, May 2022, doi: 10.55524/ijircst.2022.10.3.13.

[4] Tejas Kaneriya, “Advantages & Disadvantages of Node.js : Why to Use Node.js?” Accessed: Jan. 28, 2024. [Online]. Available: https://www.simform.com/blog/nodejs-advantages-disadvantages/

[5] G. Brito and M. T. Valente, “REST vs GraphQL: A controlled experiment,” in *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020*, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, pp. 81–91. doi: 10.1109/ICSA47634.2020.00016.

[6] J.-Y. Nie, Institute of Electrical and Electronics Engineers, and IEEE Computer Society, *2017 IEEE International Conference on Big Data : proceedings : Dec 11- 14, 2017, Boston, MA, USA*.

[7] B. Sredojev, D. Samardzija, and D. Posarac, “WebRTC technology overview and signaling solution design and implementation.”

[8] “Web RTC Documentation.” Accessed: Jan. 29, 2024. [Online]. Available: https://webrtc.org/

[9] G. L. Muller, “HTML5 WebSocket protocol and its application to distributed computing,” Sep. 2014.

[10] Priya Pedamkar, “WebSocket vs Socket.io.” Accessed: Jan. 28, 2024. [Online]. Available: https://www.educba.com/websocket-vs-socket-io/

[11] “Socket. IO vs. WebSocket: Keys Differences.” Accessed: Jan. 28, 2024. [Online]. Available: https://apidog.com/articles/socket-io-vs-websocket/

[12] “Vue.js history.” Accessed: Jan. 30, 2024. [Online]. Available: https://madushaprasad21.medium.com/vue-js-history-1a6b8567198f

[13] “Vue,js Documentation.” Accessed: Jan. 30, 2024. [Online]. Available: https://vuejs.org/

[14] “Vue School.” Accessed: Jan. 30, 2024. [Online]. Available: https://vueschool.io/articles/vuejs-tutorials/options-api-vs-composition-api/

[15] “Pinia Documentation.” Accessed: Jan. 30, 2024. [Online]. Available: https://pinia.vuejs.org/

[16] “Quasar Documentation.” Accessed: Jan. 30, 2024. [Online]. Available: https://quasar.dev/

[17] “WindiCSS Documentation.” Accessed: Jan. 30, 2024. [Online]. Available: https://windicss.org/

[18] Paul Dubois, *MySQL*, 5th ed. Boston, 2013.

[19] “History of MySQL.” Accessed: Jan. 28, 2024. [Online]. Available: https://dev.mysql.com

[20] “Discuss the history of MySQL.” Accessed: Jan. 28, 2024. [Online]. Available: https://www.tutorialspoint.com/discuss-the-history-of-mysql

[21] Nilesh Parashar, “What is Object-Relational Mapping?” Accessed: Jan. 28, 2024. [Online]. Available: https://medium.com/@niitwork0921/what-is-object-relational-mapping-38a9830d5056

[22] “ORM Is an Offensive Anti-Pattern.” Accessed: Jan. 28, 2024. [Online]. Available: https://www.yegor256.com/2014/12/01/orm-offensive-anti-pattern.html

[23] “What is version control?” Accessed: Jan. 28, 2024. [Online]. Available: https://www.atlassian.com/git/tutorials/what-is-version-control

[24] Ishan Gaba, “What is Version Control and What Are Its Benefits?” Accessed: Jan. 28, 2024. [Online]. Available: https://www.simplilearn.com/tutorials/devops-tutorial/version-control

[25] Suri Patel, “Why you should move from centralized version control to distributed version control.” Accessed: Jan. 28, 2024. [Online]. Available: https://about.gitlab.com/blog/2020/11/19/move-to-distributed-vcs/

[26] J. L. Prem Kumar Ponuthorai, *Version Control with Git*, 3rd ed. O’Reilly Media, Inc., 2022.

[27] K. Shingala, “JSON Web Token (JWT) based client authentication in Message Queuing Telemetry Transport (MQTT),” Mar. 2019, [Online]. Available: http://arxiv.org/abs/1903.02895