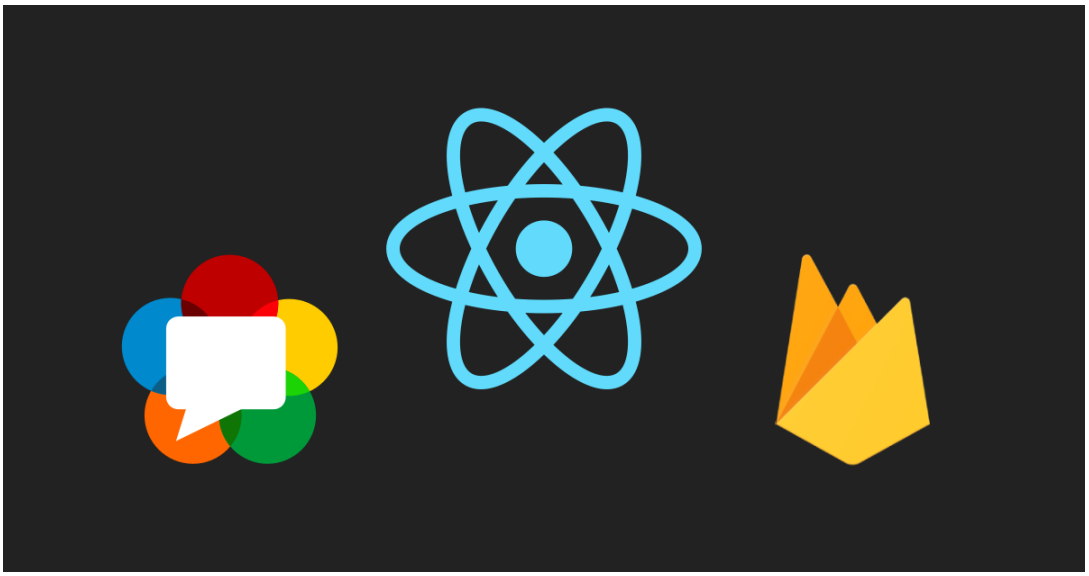
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη web εφαρμογής chat σε ReactJS»



Του φοιτητή
Στέφανου Αυλωνίτη
Αρ. Μητρώου: 164637

Επιβλέπων
Ιγνάτιος Δεληγιάννης
Βαθμίδα: Καθηγητής

Ημερομηνία 20/06/2022

Τίτλος Δ.Ε. Ανάπτυξη web εφαρμογής chat σε ReactJS
Κωδικός Δ.Ε. 21179
Ονοματεπώνυμο φοιτητή Στέφανος Αυλωνίτης
Ονοματεπώνυμο εισηγητή Ιγνάτιος Δεληγιάννης
Ημερομηνία ανάληψης Δ.Ε. 11/03/2021
Ημερομηνία περάτωσης Δ.Ε. 20/06/2022

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

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

«Στην οικογένεια μου»

Πρόλογος

Η καθημερινή χρήση των κοινωνικών μέσων δικτύωσης και η επικοινωνία μέσα από ηλεκτρονικές εφαρμογές μηνυμάτων ήταν ο αρχικός λόγος για τον οποίο επέλεξα αυτή την Διπλωματική Εργασία καθώς ήμουν περίεργος να ανακαλύψω πως φτιάχνεται μία τέτοια εφαρμογή από το μηδέν. Επιπλέον η επιλογή της ReactJS ως την βασική τεχνολογία ήταν για την καλύτερη εκμάθηση και κατανόηση της ανάπτυξης μιας σύγχρονης διαδικτυακής εφαρμογής. Η χρήση του Firebase ως την υποδομή της εφαρμογής μου έδωσε την ευκαιρία να έρθω σε επαφή με τις υπηρεσίες που προσφέρει, να αφοσιωθώ στην ανάπτυξη της web εφαρμογής και παράλληλα να αντιμετωπίσω τις δυσκολίες που προκύπτουν κατά ανάπτυξη μίας εφαρμογής χωρίς κανονικό back-end. Τέλος το WebRTC για την υλοποίηση των κλήσεων, μου επέφερε σημαντικές γνώσεις που αφορούν τις διασυνδέσεις πραγματικού χρόνου μεταξύ εφαρμογών και πως αυτές επιτυγχάνονται χωρίς κάποιον ενδιάμεσο server.

Περίληψη

Το αντικείμενο της παρούσας Διπλωματικής Εργασίας είναι η ανάπτυξη μίας διαδικτυακής εφαρμογής για την επικοινωνία μέσω γραπτών μηνυμάτων ή κλήσεων και την εύρεση τυχαίων ατόμων για επικοινωνία. Έγινε χρήση της βιβλιοθήκης ReactJS για την υλοποίηση της εφαρμογής, τις τεχνολογίες που παρέχει η πλατφόρμας Firebase για την βασική υποδομή της εφαρμογής και του WebRTC για την επίτευξη των κλήσεων. Επίσης χρησιμοποιήθηκαν διάφορα πρότυπα σχεδίασης με κυριότερο το αρχιτεκτονικό πρότυπο Domain Driven Design σύμφωνα με το οποίο αναπτύχθηκε και σχεδιάστηκε η εφαρμογή και το πρότυπο Model-View-Intent το οποίο χρησιμοποιήθηκε για την οργάνωση και διαχείριση του ReactJS επιπέδου της εφαρμογής. Η γλώσσα Sass βοήθησε στην οργάνωση και στην εφαρμογή των styles της εφαρμογής και σε συνδυασμό με το Material Design δημιουργήθηκε η τελική διεπαφή χρήστη που παρουσιάζεται. Με τα παραπάνω δημιουργήθηκε μία μοντέρνα εφαρμογή τόσο εμφανισιακά όσο και αρχιτεκτονικά που προσφέρει δυνατότητες επικοινωνίας στους χρήστες.

«Development of a chat app using ReactJS»

«Stefanos Avlonitis»

Abstract

The aim of this thesis is the development of a messaging web application with voice and video communication features that also provides users the ability to communicate with random people. The application was developed using ReactJS, Firebase services as the infrastructure and WebRTC for voice and video communication. Furthermore, a variety of design patterns were used, most notably the Domain Driven Design according to which the entire application was designed and developed. Also, the Model-View-Intent applied to organize and manage the ReactJS application layer. The Sass language was used to organize and apply styles, combined with Material Design the final user interface created. Finally, a modern designed application was developed by giving the users the ability to communicate with each other.

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον κ. Δεληγιάννη για την εμπιστοσύνη που μου έδειξε για την ανάληψη και ολοκλήρωση αυτής της διπλωματικής. Καθώς επίσης, την οικογένεια μου που με υποστήριξε σε όλη την διάρκεια των σπουδών μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
DDD	Domain Driven Design
UI	User Interface
MVI	Model View Intent
DI	Dependency Injection

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή

Η παρακάτω διπλωματική εργασία έχει να κάνει με την ανάπτυξη μίας chat εφαρμογής σε web για την επικοινωνία ατόμων από διάφορα μέρη του κόσμου. Παρέχεται η δυνατότητα αναζήτησης τυχαίων ατόμων για επικοινωνία μέσω μηνυμάτων όπως και η επιλογή προσθήκης στην λίστα φίλων για πιο συχνή επικοινωνία. Επίσης η επικοινωνία επεκτείνεται μέσω κλήσεων ήχου και βίντεο ώστε οι χρήστες να έχουν την δυνατότητα να έρθουν σε πιο άμεση επαφή.

1.2 Σύντομη περιγραφή αρχιτεκτονικής της εφαρμογής

Η εφαρμογή είναι γραμμένη με χρήση της βιβλιοθήκης ReactJS για την δημιουργία των διεπαφών χρήστη και συνολικά η εφαρμογή χρησιμοποιεί την γλώσσα TypeScript που προσφέρει επιπλέον λειτουργίες στην JavaScript. Για την υλοποίηση των κλήσεων και των βίντεο κλήσεων χρησιμοποιείται το WebRTC και για τις back-end διαδικασίες γίνεται χρήση των υπηρεσιών του Firebase.

Κατά την ανάπτυξη της εφαρμογής δόθηκε βάση στην αρχιτεκτονική, πως η εφαρμογή θα είναι το δυνατόν συντηρήσιμη στο μέλλον και πως ο κώδικας θα είναι σωστά δομημένος. Για αυτό η εφαρμογή είναι χωρισμένη κατά βάση σε 3 μέρη, το View επίπεδο που έχει να κάνει με τις διεπαφές που είναι γραμμένες σε ReactJS, το επίπεδο που υλοποιεί την επιχειρηματική λογική (Business Logic) της εφαρμογής και τέλος το επίπεδο Presentation/Παρουσίασης το οποίο αναλαμβάνει την διαχείριση των διεπαφών υλοποιώντας το Model-View-Intent πρότυπο και την μετέπειτα επικοινωνία με το Business επίπεδο. Το Business επίπεδο αναπτύχθηκε ακολουθώντας το Domain Driven Design και είναι χωρισμένο σε Microservices.

Για την κυκλοφορία των δεδομένων στην εφαρμογή χρησιμοποιείται η βιβλιοθήκη RxJS που με την βοήθεια της η εφαρμογή είναι αντιδραστική (reactive). Τέλος γίνονται χρήση διάφορων patterns πέρα από αυτά που αναφέρθηκαν με σημαντικότερο το Dependency Injection pattern που βοηθά στην αρχιτεκτονική της εφαρμογής.

1.3 Περιγραφή κεφαλαίων

Η εργασία αποτελείται από τα παρακάτω κεφάλαια.

Στο πρώτο κεφάλαιο γίνεται μία σύντομη περιγραφή της εφαρμογής, της αρχιτεκτονικής που ακολουθεί η εφαρμογή μαζί με τις τεχνολογίες και τα πρότυπα που χρησιμοποιούνται.

Στο δεύτερο κεφάλαιο αναλύεται η ReactJS και τα βασικά σημεία που χρησιμοποιούνται στην εφαρμογή.

Το τρίτο κεφάλαιο αφορά την γλώσσα TypeScript, αναλύει τι επιπλέον προσφέρει στις λειτουργίες τις JavaScript αλλά και στην ReactJS.

Στο τέταρτο κεφάλαιο, αναλύονται οι δυνατότητες της RxJS μέσω της οποίας ταξιδεύουν τα δεδομένα στην εφαρμογή, οι operators και ποιοι χρησιμοποιούνται στην εφαρμογή.

Στο πέμπτο κεφάλαιο γίνεται εκτενής ανάλυση των υπηρεσιών του Firebase που γίνονται χρήση στην εφαρμογή και ο τρόπος που γίνεται η προσπέλαση τους.

Στο έκτο κεφάλαιο, περιγράφονται οι λειτουργίες και οι δυνατότητες που παρέχει το WebRTC και πως επιτυγχάνονται οι κλήσεις ανάμεσα στους χρήστες.

Το έβδομο κεφάλαιο έχει να κάνει με το πρότυπο Model-View-Intent, περιγράφεται ο τρόπος που λειτουργεί και πως υλοποιείται μέσα στην εφαρμογή.

Στο όγδοο κεφάλαιο αναφέρεται το Dependency Injection πρότυπο, πως χρησιμοποιείται και τι προσφέρει στην εφαρμογή.

Το ένατο κεφάλαιο έχει να κάνει με το Domain Driven Design, περιγράφεται η βασική του ιδέα, οι λόγοι που είναι χρήσιμο και ο τρόπος που υιοθετήθηκε και εφαρμόστηκε σε συνδυασμό με τα Microservices.

Στο δέκατο κεφάλαιο αναφέρονται οι τρόποι που εφαρμόστηκαν τα styles στις διεπαφές και γίνονται παρουσίαση μέσω εικόνων οι λειτουργίες και οι διεπαφές της εφαρμογής.

Τέλος στο ενδέκατο κεφάλαιο αναφέρονται τα συμπεράσματα που προέκυψαν από την δημιουργία της εφαρμογής όπως επίσης και μερικές επεκτάσεις της εφαρμογής που μπορούν να εφαρμοστούν στο μέλλον.

Κεφάλαιο 2ο: ReactJS

2.1 Εισαγωγή

Η ReactJS είναι μια ανοικτού κώδικα JavaScript βιβλιοθήκη για την ανάπτυξη διεπαφών. Η ανάπτυξη της στηρίζεται κατά μεγάλο βαθμό στην κοινότητα και αυτό σημαίνει ότι η βιβλιοθήκη αναπτύσσεται και εξελίσσεται σύμφωνα με τις ανάγκες που προκύπτουν από τους χρήστες που την χρησιμοποιούν. Παρότι υπάρχουν αρκετές εναλλακτικές λύσεις για την ανάπτυξη διεπαφών όπως είναι η Angular ή η Vue, η ReactJS έχει καταφέρει να εδραιωθεί και να χρησιμοποιείται τόσο από εταιρείες όσο και από προγραμματιστές για μικρές εφαρμογές.

Από τα σημαντικότερα χαρακτηριστικά που παρουσιάζει η ReactJS είναι η εκ φύσεως component-based λογική, το εύκολο συντακτικό, η εύκολη δημιουργία διεπαφών, το Virtual DOM και η γρήγορη εκμάθηση της.

2.2 Τι είναι το Virtual DOM (VDOM);

Το Virtual DOM είναι ο μηχανισμός τον οποίο η React χρησιμοποιεί για να εκτυπώσει στοιχεία στην οθόνη ο οποίος αυξάνει κατά πολύ την ταχύτητα συγκριτικά με τον "παραδοσιακό" πλέον τρόπο.

Ο παραδοσιακός τρόπος, όταν προκύπτει μία αλλαγή η οποία πρέπει να ερμηνευτεί στο DOM, ζωγραφίζει - δημιουργεί ξανά όλα τα στοιχεία στην οθόνη. Αυτό δημιουργεί σοβαρό πρόβλημα στην απόδοση μιας εφαρμογής καθώς πρέπει να ξαναγίνουν οι ίδιοι υπολογισμοί από την αρχή σε κάθε αλλαγή που προκύπτει είτε αυτή σημαίνει μία αλλαγή σε έναν αριθμό είτε αυτό σημαίνει αλλαγή του χρώματος όλης της σελίδας.

Από την άλλη μεριά, το Virtual DOM κρατάει σε αντίγραφο την αναπαράσταση του πραγματικού DOM το οποίο έχει μορφή ενός δέντρου από κόμβους και σε κάθε αλλαγή που προκύπτει η React κάνει αυτή την αλλαγή στο αντίγραφο που έχει κρατήσει. Έπειτα γίνεται η σύγκριση των δύο DOM (Virtual και πραγματικό) και μόνο το κομμάτι που διαφέρει θα ζωγραφιστεί τελικά στην οθόνη.

Είναι ξεκάθαρο πως με το Virtual DOM η React αυξάνει κατακόρυφα την ταχύτητα μιας εφαρμογής καθώς στην οθόνη γίνονται μόνο οι αλλαγές που πρέπει και όχι ολόκληρη η σελίδας της εφαρμογής από την αρχή. Τέλος να σημειωθεί, πως ο προγραμματιστής δεν χρειάζεται να απασχολείται με τον τρόπο με τον οποίο η React μεταφράζει τις αλλαγές στην οθόνη και πως λειτουργεί το VDOM, καθώς όλα αυτά τα αναλαμβάνει η React για τον ίδιο.

2.3 Τι είναι το JSX;

JSX σημαίνει JavaScript and XML και είναι μία επέκταση του JavaScript συντακτικού. Με το JSX είναι εύκολο να γραφτεί JavaScript κώδικας μαζί με HTML στοιχεία. Όπως φαίνεται στην Εικόνα 2.1 υπάρχουν κάποιες εκφράσεις JSX που αποθηκεύονται σε μεταβλητές. Αρχικά ένα div στοιχείο δημιουργείται και αποθηκεύεται σε μια μεταβλητή με όνομα div, έπειτα μία μεταβλητή name που παίρνει την τιμή "Kwstas" και τέλος την μεταβλητή content να αρχικοποιείται με ένα div το οποίο περικλείει το {name}. Με την χρήση αγκιστροειδών αγκύλων { } γύρω από την μεταβλητή name, η React καταλαβαίνει ότι υπάρχει μία δομή στο εσωτερικό των αγκύλων στην οποία μπορεί να έχει πρόσβαση. Έτσι παίρνει το περιεχόμενο της μεταβλητής και το βάζει στην θέση του {name} και η μεταβλητή content θα πάρει την τιμή Kwstas . Αντί για το name θα μπορούσε στην θέση του να βρίσκεται οτιδήποτε, όπως για παράδειγμα θα μπορούσε να βρίσκεται η μεταβλητή div και η

τιμή του content να ήταν το element ή θα μπορούσε να είναι ένα function που επιστρέφει και αυτό με την σειρά του είτε μία απλή τιμή που είτε μία JSX έκφραση.

```
let div = <div>element</div>;
let name = 'Kwstas';
let content = <div>{name}</div>;
```

Εικόνα 2.1 JSX Elements

2.4 Τι είναι τα components;

Components ή συστατικά ονομάζονται κομμάτια κώδικα ενός μεγαλύτερου συνόλου μιας εφαρμογής ή ενός προγράμματος που εκτελούν διάφορες λειτουργίες και προάγουν την επαναχρησιμοποίηση τους μέσα σε αυτό το σύνολο. Τα components στην React, αντιπροσωπεύουν επαναχρησιμοποιούμενα κομμάτια κώδικα που δημιουργούν διεπαφές χρήστη, δέχονται εισόδους από άλλα components τα οποία λέγονται props (από το properties) και το κάθε ένα έχει δική του κατάσταση (State). Χωρίζονται σε δύο κατηγορίες, τους functional και class components.

Οι κύριες διαφορές τους είναι ότι αρχικά τα functional components είναι απλά JavaScript functions που επιστρέφουν JSX εκφράσεις ενώ τα class components είναι κλάσεις που επεκτείνουν το React.Component. Τα class components μέσω των μεθόδων που έχουν για την διαχείριση του κύκλου ζωής του συστατικού είναι Stateful που σημαίνει ότι διαχειρίζονται την κατάσταση του component, ενώ οι functions είναι stateless που σημαίνει ότι δεν έχουν κατάσταση και γίνεται χρήση των Hooks για να γίνουν Stateful, διαφορετικά κάθε αλλαγή για παράδειγμα σε κάποια μεταβλητή θα χαθεί όταν γίνει ανανέωση στο component. Επίσης όταν το component είναι Stateful, το State μπορεί να είναι ένα οποιοδήποτε object ή μία οποιαδήποτε μεταβλητή και ο μοναδικός που μπορεί να επηρεάσει αυτό το object είναι το συγκεκριμένο component. Η πιο σημαντική διαφορά είναι ότι τα functional components είναι πιο γρήγορα σε απόδοση από τα class components για αυτό και χρησιμοποιούνται περισσότερο. Από την άλλη μεριά και τα 2 (functional και class components) δέχονται props και μπορούν να τα χρησιμοποιήσουν στον κύκλο ζωής τους, τα πρώτα (functional) δέχονται τα props σαν παράμετρο μεθόδου και τα δεύτερα σαν παράμετρο στον constructor της κλάσης. Είναι σημαντικό να αναφερθεί ότι τα props είναι μόνο για ανάγνωση (read-only), αυτό σημαίνει ότι ένα component δεν μπορεί να επηρεάσει άμεσα ένα prop που δέχεται. Με τα read-only props, επιτυγχάνεται και η ασφάλεια του State ενός Component καθώς ακόμα κι αν το State δοθεί σε ένα παιδί - Component (Child), το δεύτερο μπορεί μόνο να το διαβάσει.

Στην Εικόνα 2.2 φαίνονται δύο components, το ένα functional και το άλλο class που κάνουν την ίδια λειτουργία. Είναι ξεκάθαρο ότι στην περίπτωση του function είναι πιο γρήγορο να γραφτεί ενώ στην περίπτωση του class είναι πιο ευανάγνωστο και μπορεί να γίνει κατανοητό και από κάποιον που δεν γνωρίζει React.

```
class CounterClass extends React.Component {
    constructor(props) {
        super(props);
        this.state = { counter: 0 };
    }

    componentDidMount() {
        document.title = `Counter: ${this.state.counter}`;
    }

    componentDidUpdate() {
        document.title = `Counter: ${this.state.counter}`;
    }

    render() {
        return (
            <div>
                <h1>Counter: {this.state.counter}</h1>
                <button onClick={() => this.setState( state: { count: this.state.counter + 1 })}>
                    Click
                </button>
            </div>
        );
    }
}
```

```
function CounterFunction() {
    const [counter, setCounter] = useState( initialState: 0);

    useEffect( effect: () => {
        document.title = `Counter: ${counter}`;
    });

    return (
        <div>
            <h1>Counter: {counter}</h1>
            <button onClick={() => setCounter( value: counter + 1)}>
                Click me
            </button>
        </div>
    );
}
```

Εικόνα 2.2 Functional vs Class Component

2.5 Τι είναι τα hooks;

Τα Hooks είναι μέθοδοι που χρησιμοποιούνται για να δώσουν σε function components επιπλέον δυνατότητες που αφορούν το State και τον κύκλο ζωής τους. Για παράδειγμα όπως προαναφέρθηκε ένα function component είναι stateless, έτσι με κάθε ανανέωση οι τυχόν μεταβλητές που είχαν καθοριστεί χάνουν την τιμή τους, για αυτό υπάρχουν κάποια hooks όπως είναι το useState hook που καθορίζει το state μίας μεταβλητής και δεν χάνει την τιμή του στις ανανεώσεις του component. Κατά κύριο λόγο όλα τα hooks έχουν παρόμοιο συντακτικό, δέχονται κάποιες παραμέτρους όπως ένα object, μία τιμή ή ένα function και επιστρέφουν μία συγκεκριμένη τιμή, object ή και λίστα από objects.

Πέρα από τα hooks που παρέχει η βιβλιοθήκη, υπάρχει δυνατότητα και για δημιουργία hook από τον προγραμματιστή (custom) σύμφωνα με τις εκάστοτε ανάγκες. Κατά την δημιουργία ενός hook (custom hook) υπάρχει δυνατότητα χρησιμοποίησης των ήδη υπάρχων hooks και επέκταση τους.

Τα hooks που χρησιμοποιούνται κυρίως στην εφαρμογή και περιγράφονται παρακάτω είναι τα useState, useEffect, useRef τα οποία παρέχει εκ φύσεως η React, το useParams hook που παρέχεται από την βιβλιοθήκη react-router-dom και το useViewComponent το οποίο είναι ένα custom hook.

2.5.1 useState

Το useState προσφέρει State λογική σε έναν functional component. Δέχεται μία παράμετρο η οποία είναι η αρχική τιμή, που μπορεί να οριστεί είτε ως τιμή οποιουδήποτε τύπου ή στις περιπτώσεις που ο υπολογισμός της αρχική τιμής είναι υπολογιστικά ακριβός ένα function που επιστρέφει την τιμή αυτή. Επιστρέφει μία λίστα με δύο στοιχεία, το πρώτο είναι η τιμή που είναι αμετάβλητη και για αυτό το δεύτερο είναι ένα function που καλώντας το γίνεται update η τιμή αυτή.

2.5.2 useEffect

Το useEffect hook προσφέρει στο component δυνατότητες που έχουν να κάνουν με τον κύκλο ζωής του, για παράδειγμα μπορεί χρησιμοποιηθεί για να εκτελεστεί κάτι αυτόματα μετά από κάθε αλλαγή σε μία τιμή ή κατά το τέλος της ζωής του component. Το useEffect δέχεται δύο παραμέτρους, ένα function στο οποίο γίνεται κάποια ενέργεια και μία προαιρετική λίστα από objects για παρατήρηση αλλαγών που είναι κομμάτια του component όπως props ή state. Η λίστα παρατήρησης αλλαγών χρησιμοποιείται

ώστε κάθε φορά που αλλάζει κάποια από τις τιμές των objects που βρίσκονται στην λίστα να τρέχει η ενέργεια που βρίσκεται μέσα στο function της πρώτης παραμέτρου. Αν η λίστα παρατήρησης παραμείνει άδεια τότε η ενέργεια θα εκτελεστεί μόνο την πρώτη φορά, δηλαδή μετά το πρώτο render του component. Αν από την άλλη στην θέση της λίστας δεν δοθεί τίποτα τότε η ενέργεια θα εκτελείται σε κάθε αλλαγή που προκύπτει και συγκεκριμένα μετά από κάθε render. Τέλος, με το useEffect υποστηρίζεται ο καθαρισμός από τυχόν listeners ή subscriptions ή κάποια σύνδεση με κάποια εξωτερική εφαρμογή πριν καταστραφεί το component, έτσι επιστρέφοντας ένα function μέσα στην ενέργεια της πρώτης παραμέτρου τότε το useEffect γνωρίζει ότι πρέπει να εκτελέσει αυτό το function προτού καταστραφεί το component.

2.5.3 useRef

Το useRef κρατάει ένα αντικείμενο στην .current ιδιότητα του και επιστρέφει πάντα το ίδιο, συνήθως χρησιμοποιείται στην θέση του getElementById function του document για να μειωθούν οι επιπλέον έλεγχοι στο DOM. Έχει μία είσοδο ως παράμετρο που μόλις του δοθεί θα θέσει στην .current ιδιότητα την τιμή της εισόδου. Διαφορετικά αν σε ένα οποιοδήποτε HTML στοιχείο περαστεί ένα ref prop και σαν τιμή ένα αντικείμενο ref τότε η .current ιδιότητα του ref θα δείχνει σε αυτό το στοιχείο.

2.5.4 useParams

Το useParams επιστρέφει ένα αντικείμενο με τις URL παραμέτρους σε μορφή κλειδιού/τιμής (key/value).

2.5.5 useViewComponent

Το useViewComponent είναι ένα custom hook που χρησιμοποιούν τα components για να επικοινωνήσουν με το επίπεδο παρουσίασης το οποίο θα αναλυθεί στο κεφάλαιο του MVI. Ως παραμέτρους δέχεται ένα function που επιστρέφει το στιγμιότυπο του Presenter, την View μεταβλητή του component και επιστρέφει την κατάσταση State και το στιγμιότυπο του Presenter. Είναι γενικού τύπου hook (generic) και το generic ορίζει τον τύπο των επιστρεφόμενων αντικειμένων. Χρησιμοποιεί τα useEffect και useState hooks, εξασφαλίζοντας ότι ο component που το χρησιμοποιεί θα είναι πάντα ενημερωμένος με τις τελευταίες αλλαγές που προκύπτουν και την σωστή επικοινωνία με τον Presenter.

2.6 Επίλογος

Παραπάνω αναλύθηκε εν συντομία τι είναι η React και πως αντιλαμβάνεται και αντιμετωπίζει τις αλλαγές με το Virtual DOM. Παρουσιάστηκε το συντακτικό JSX που παρέχει και ποιες δυνατότητες προσφέρει για τον συνδυασμό JavaScript και HTML. Επίσης τα function και class components σχολιάστηκαν εκτενώς και παρουσιάστηκαν οι λειτουργίες των function components τόσο με, όσο και χωρίς τα Hooks που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής.

Κεφάλαιο 3ο: TypeScript

3.1 Τι είναι η TypeScript

Η TypeScript είναι ένα υπερσύνολο της JavaScript, αυτό σημαίνει πως οποιοσδήποτε κώδικας γραμμένος σε JavaScript είναι σωστός και σε TypeScript αλλά όχι το ανάποδο. Η TypeScript υποστηρίζει όλες τις λειτουργίες της JavaScript, είναι object-oriented και προσθέτει στατικούς τύπους στην JavaScript όπως το string, το number, τον τύπο των μεθόδων όπως το void. Οι στατικοί τύποι βοηθούν αρκετά κατά την διάρκεια της ανάπτυξης μιας εφαρμογής στον εντοπισμό λαθών που είναι αρκετά συχνό φαινόμενο καθώς τα λάθη (errors) γίνονται φανερά κατά την διαδικασία της μετάφρασης. Η TypeScript μεταφράζεται (transpile) σε JavaScript, έτσι μία εφαρμογή μπορεί μόλις γίνει transpile να ενσωματωθεί σε μία Vanilla JavaScript εφαρμογή χωρίς κανένα πρόβλημα. Το γεγονός ότι η TS είναι υπερσύνολο της JavaScript βοηθά κάποιον που χρησιμοποιεί πρώτη φορά TS να ενσωματωθεί γρήγορα καθώς το μοναδικό πράγμα που έχει ως προαπαίτηση είναι η JavaScript. Τέλος η TS μπορεί να γίνει transpile σε οποιαδήποτε έκδοση της JavaScript χρειαστεί (backwards compatibility) και αυτό είναι χρήσιμο γιατί μία λειτουργία γραμμένη στην πιο πρόσφατη έκδοση της TypeScript μπορεί να χρησιμοποιηθεί σε εφαρμογές με παλαιότερη έκδοση της JavaScript.

3.2 TypeScript και ReactJS

Η ReactJS όπως λέει και το όνομα της γράφεται με την γλώσσα JavaScript, παρόλα αυτά μπορεί να χρησιμοποιηθεί η TypeScript και να της προσφέρει τις λειτουργίες της ώστε να γίνει η ανάπτυξη πιο εύκολη. Με την TypeScript είναι πιο εύκολο όπως αναφέρθηκε να ανιχνευθούν τα λάθη, έτσι στην React μπορούν να αποφευχθούν τα περισσότερα type errors που προκύπτουν. Σημαντικό παράδειγμα είναι τα Properties ενός component, που πολλές φορές λόγω των δυναμικών τύπων της JS ο προγραμματιστής έρχεται αντιμέτωπος με προβλήματα που έχουν να κάνουν με τους τύπους των Props. Το πρόβλημα αυτό λύνεται με την βοήθεια των interfaces και των στατικών τύπων της TS, το component είναι πιο συμπαγές καθώς μπορεί να ορίσει συγκεκριμένα ποιες και αν έχει εισόδους.

Κεφάλαιο 4ο: RxJS

4.1 Εισαγωγή

Ολόκληρη η εφαρμογή είναι αντιδραστική (reactive), δηλαδή φτιαγμένη με τρόπο τέτοιο ώστε να αντιδρά στις αλλαγές που προκύπτουν. Για την υλοποίηση της αντιδραστικής φύσης της εφαρμογής γίνεται χρήση της βιβλιοθήκης RxJS.

Η βιβλιοθήκη RxJS είναι μία υλοποίηση της βιβλιοθήκη ReactiveX για JavaScript και χρησιμοποιείται για την δημιουργία προγραμμάτων που είναι ασύγχρονα και κατά κύριο λόγο σε προγράμματα που αντιδρούν με συγκεκριμένο τρόπο σε συμβάντα που προκύπτουν. Υλοποιεί το πρότυπο του παρατηρητή (Observer pattern) και προσφέρει έτσι ασύγχρονη διαχείριση των δεδομένων μέσω ακολουθιών.

Για την καλύτερη κατανόηση θα αναφερθούν κάποιες έννοιες που είναι σημαντικές για την συνέχεια. Αρχικά υπάρχει η έννοια του Observable, το οποίο είναι μία ροή δεδομένων που από μέσα της ταξιδεύουν δεδομένα τα οποία ταξιδεύουν σε απροσδιόριστο χρόνο (ασύγχρονα), έπειτα υπάρχει η έννοια του παρατηρητή (Observer) ο οποίος παρατηρεί ένα Observable και τον ενδιαφέρουν τα δεδομένα που ταξιδεύουν από μέσα του. Τέλος υπάρχει η έννοια της εγγραφής (Subscription) όπου ο Observer εγγράφεται σε ένα Observable για να μάθει τα δεδομένα του και όταν πλέον δεν τον ενδιαφέρουν αυτά τα δεδομένα μπορεί να κάνει απεγγραφή. Στην συνέχεια θα περιγραφούν εκτενέστερα οι παραπάνω έννοιες όπως επίσης και οι operators που έχουν σημαντικό ρόλο στην ροή των δεδομένων.

4.2 Τι είναι τα Observables

Το Observable είναι μία συνεχόμενη ροή δεδομένων που μοιάζει με μία λίστα στην οποία προστίθενται αργά τιμές. Έχει αρχή και τέλος, δημιουργείται με την κλάση Observable που παρέχει η RxJS και εκπέμπει νέες τιμές με την μέθοδο next. Αυτές τις τιμές - δεδομένα για να τις προσπελάσει κάποιος πρέπει να κάνει εγγραφή σε αυτή την δομή και να γίνει ο παρατηρητής. Το Observable τελειώνει με δύο τρόπους, είτε καλώντας την μέθοδο complete που σηματοδοτεί το τέλος ροής των δεδομένων ή μπορεί να προκύψει κάποιο λάθος (Error) καλώντας την μέθοδο error και περνώντας το λάθος σαν παράμετρο. Στις δύο περιπτώσεις, το Observable τελειώνει εκεί και δεν πρόκειται να εκπέμψει άλλη τιμή, συνεπώς ο Observer δεν πρόκειται να λάβει άλλη τιμή. Είναι σημαντικό να σημειωθεί πως ο Observer μπορεί να κάνει απεγγραφή από την ροή δεδομένων οποιαδήποτε στιγμή και δεν είναι απαραίτητο να ολοκληρωθεί το Observable αλλά ούτε και να λάβει δεδομένα.

4.3 Τι είναι το Subject

Το Subject είναι ένα Observable με την διαφορά ότι ένα Observable εκπέμπει τα δεδομένα του σε έναν Observer ενώ το Subject μπορεί να εκπέμψει δεδομένα σε όσους Observers τον παρατηρούν. Αυτή του η δυνατότητα είναι πολύ χρήσιμη καθώς σε μία εφαρμογή όπως αυτή μπορούν διάφορα σημεία της εφαρμογής να κάνουν εγγραφή στο ίδιο Subject και να είναι ενήμερα ταυτόχρονα, για παράδειγμα, όταν κατά την διάρκεια μιας συνομιλίας το άτομο αποσυνδεθεί από την εφαρμογή τότε το σημείο του chat που δείχνει αν είναι διαθέσιμος και το αντίστοιχο σημείο στο side-bar πρέπει να ενημερωθούν και αυτό επιτυγχάνεται με την χρήση ενός Subject.

4.4 Τι είναι το Subscription

Όταν ένας Observer εγγράφεται σε ένα Observable τότε επιστρέφεται ένα αντικείμενο τύπου Subscription. Το Subscription έχει την unsubscribe μέθοδο η οποία χρησιμοποιείται για να απεγγραφεί ο Observer από την ροή δεδομένων. Είναι σημαντικό να αναφερθεί ότι το unsubscribe από μία ροή δεδομένων είναι πολύ σοβαρή διαδικασία καθώς αν κάποιος observer συνεχίσει να παρατηρεί μία ροή σε στιγμές που δεν είναι αναγκαίο μπορεί να προκληθούν απώλειες μνήμης (memory leaks).

4.5 Operators

Οι Rx Operators είναι functions που ουσιαστικά χρησιμοποιούνται για να αλλάξουν την ροή δεδομένων ανάλογα με τις ανάγκες που προκύπτουν. Για παράδειγμα ένας operator που λέγεται filter θα φιλτράρει τις τιμές που δεν περνάνε στον λογικό έλεγχο που ορίζει το filter.

Υπάρχουν δύο ειδών operators, οι operators δημιουργίας και οι operators σωλήνωσης (pipeable operator).

4.5.1 Operators δημιουργίας

Η λειτουργία των operators δημιουργίας είναι να δημιουργήσουν ένα νέο Observable που θα εκπέμψει κάποιες τιμές, όπως είναι το of operator το οποίο φτιάχνει ένα Observable και εκπέμπει την τιμή που θα πάρει ως παράμετρο. Έτσι το of (undefined) θα εκπέμψει την τιμή undefined.

4.5.2 Operators σωλήνωσης (Pipeable Operator)

Τα operators διασωλήνωσης είναι τα operators που μπορεί να δημιουργήσουν μία διασωλήνωση στην εκπεμπόμενη ροή δεδομένων χρησιμοποιώντας το pipe σε μία ροή.

Ένας τέτοιος operator παίρνει μία ροή δεδομένων και επιστρέφει μία καινούρια από την οποία συνεχίζει η ροή. Σε έναν pipeable operator μπορεί να επιστρέφεται ένα observable που επιστρέφει τιμές που έρχονται από ένα εξωτερικό API ή σε περίπτωση που προκύψει κάποιο λάθος μπορεί να γίνει εγγραφή ξανά στο αρχικό Observable ώστε σταματήσει απρόσμενα η ροή των δεδομένων. Πολλά operators όπως το map και το filter είναι εμπνευσμένα από τα γνωστά map και filter functions της JavaScript. Παρακάτω περιγράφονται οι σημαντικότεροι operators που χρησιμοποιούνται στην εφαρμογή.

4.5.2.1 mergeMap

Το mergeMap επιστρέφει ένα observable που αρχικά εφαρμόζει την λειτουργία του mergeMap στις τιμές του αρχικού observable επιστρέφοντας μία νέα εσωτερική ροή για κάθε τιμή και έπειτα επιστρέφει ένα observable ενώνοντας όλες τις τιμές αυτές.

4.5.2.2 concatMap

Κάνει την ίδια διαδικασία με το mergeMap απλά σε περίπτωση που ένα εσωτερικό observable αργήσει να τελειώσει τότε το concatMap κρατάει την σωστή σειρά με την οποία έγιναν οι ροές.

4.5.2.3 tap

Το tap επιστρέφει το ίδιο ακριβός observable άθικτο, χρησιμοποιείται για να κάνει λειτουργίες που δεν επηρεάζουν την ροή των δεδομένων όπως να κάνει console.log τις τιμές που περνάνε από την ροή.

4.5.2.4 map

Δημιουργεί ένα Observable που εκπέμπει τις τιμές του αρχικού Observable μετασχηματίζοντας αυτές σύμφωνα με το function που δέχεται ως παράμετρο το map operator.

4.5.2.5 filter

Αφήνει να προχωρήσουν στην ροή δεδομένων μόνο οι τιμές του Observable που διασωληνώθηκε (piped) οι οποίες ικανοποιούν τον έλεγχο τον οποίο εφαρμόζει.

4.5.2.6 take

Το take operator χρησιμοποιείται ώστε αν ένα observable εκπέμπει συνεχόμενα τιμές ο observer να σταματήσει να παρατηρεί μετά τον αριθμό που ορίζει το take. Για παράδειγμα το take(2) σημαίνει ότι θα πάρει τις πρώτες 2 τιμές που θα εκπέμψει το Observable και μετά θα κάνει απεγγραφή.

4.5.2.7 retryWhen

Το retryWhen είναι διαφορετικό από τα υπόλοιπα operators καθώς εκτελείται μόνο όταν προκύπτει κάποιο λάθος και δέχεται ένα observable με λάθη το οποίο κάνει ξανά εγγραφή στο αρχικό Observable ώστε να μην τερματιστεί η ροή δεδομένων (error handling).

4.6 Επίλογος

Κλείνοντας αυτό το κεφάλαιο, περιγράφηκε ο τρόπος που γίνεται μία εφαρμογή Reactive και πως χρησιμοποιείται η βιβλιοθήκη RxJS. Η χρήση των Observables είναι πολύ σημαντική για εφαρμογές που βασίζονται στην ταχύτητα και στις ασύγχρονες διαδικασίες με τους operators να προσφέρουν σημαντικές δυνατότητες για την διαχείριση μιας ροής δεδομένων. Τέλος η χρήση των observables και των operators πρέπει να γίνεται με προσοχή για να μην παρουσιαστούν προβλήματα τόσο στην λειτουργία της εφαρμογής όσο και στην επιβάρυνση της συσκευής.

Κεφάλαιο 5ο: Firebase

5.1 Εισαγωγή

Το Firebase είναι μία πλατφόρμα της Google όπου λειτουργεί ως back-end-ως-υπηρεσία (back-end as a service) προσφέροντας λύσεις για στατιστικά, διαχείριση βάσεων δεδομένων, αυθεντικοποίηση χρηστών, ειδοποιήσεις push, φιλοξενία εφαρμογών (hosting), αποθηκευτικό χώρο και πολλές άλλες υπηρεσίες που βοηθούν στην ανάπτυξη λογισμικού. Με το Firebase γίνεται εύκολη η ανάπτυξη αξιόπιστων εφαρμογών χωρίς να χρειάζεται να αναπτυχθεί back-end υπηρεσία για την εφαρμογή και για τις διάφορες υπηρεσίες που μπορεί να προκύψουν όπως για παράδειγμα ειδοποιήσεις push, έτσι ο προγραμματιστής μπορεί να αφοσιωθεί και να ασχολείται μόνο με το front-end κομμάτι.

5.2 Τι είναι το back-end as a service

Το back-end as a service (θα αναφέρεται ως BaaS για ευκολία) απομονώνει τον προγραμματιστή στο front-end είτε αυτό είναι μία εφαρμογή Android, IOS ή Web παρέχοντας του κάποια API για την διαχείριση και την παραμετροποίηση των back-end υπηρεσιών που προσφέρει ο πάροχος.

5.3 Authentication

Για την εγγραφή και την σύνδεση των χρηστών στην εφαρμογή χρησιμοποιείται το Authentication του Firebase. Το Authentication παρέχει διάφορους τρόπους για την αυθεντικοποίηση των χρηστών όπως είναι το Google και το Facebook, έτσι για την εγγραφή του χρήστη στην εφαρμογή και στην συνέχεια την σύνδεση του σε αυτή χρησιμοποιείται ο Google provider.

5.4 Firebase Console

Το Firebase παρέχει ένα web γραφικό περιβάλλον που ονομάζεται Firebase Console στο οποίο υπάρχουν τόσο οι διαθέσιμες υπηρεσίες όσο και στατιστικά, δεδομένα της κάθε υπηρεσίας. Με το Console ο προγραμματιστής μπορεί να διαχειριστεί τις υπηρεσίες που χρησιμοποιεί όπως τις βάσεις δεδομένων, βλέποντας και αναλύοντας τα δεδομένα και θέτοντας κανόνες για την προσπέλαση αυτών των δεδομένων από την εκάστοτε εφαρμογή που αναπτύσσει. Παρόλες τις ευκολίες που διαθέτει το Console, ο χρήστης μπορεί να το χρησιμοποιήσει μόνο για την αρχική παραμετροποίηση της εφαρμογής και από εκεί και έπειτα να χρησιμοποιεί τις υπηρεσίες μόνο από τον εφαρμογή του.

5.5 Βάσεις δεδομένων

Το Firebase προσφέρει δύο επιλογές NoSQL βάσεων δεδομένων, το Firestore και το Realtime Database. Και οι δύο περιπτώσεις φροντίζουν τα δεδομένα να είναι συγχρονισμένα σε όλους τους clients που τα χρησιμοποιούν, έχουν εκτός σύνδεσης υποστήριξη που σημαίνει ότι αν γίνει κάποια αλλαγή την στιγμή που ο χρήστης έχει χάσει την σύνδεση του από το διαδίκτυο, μόλις το δίκτυο επανέλθει θα γίνει συγχρονισμός μεταξύ βάσης - client. Στην εφαρμογή χρησιμοποιούνται και οι δύο βάσεις οι οποίες λύνουν διαφορετικά προβλήματα που προέκυψαν.

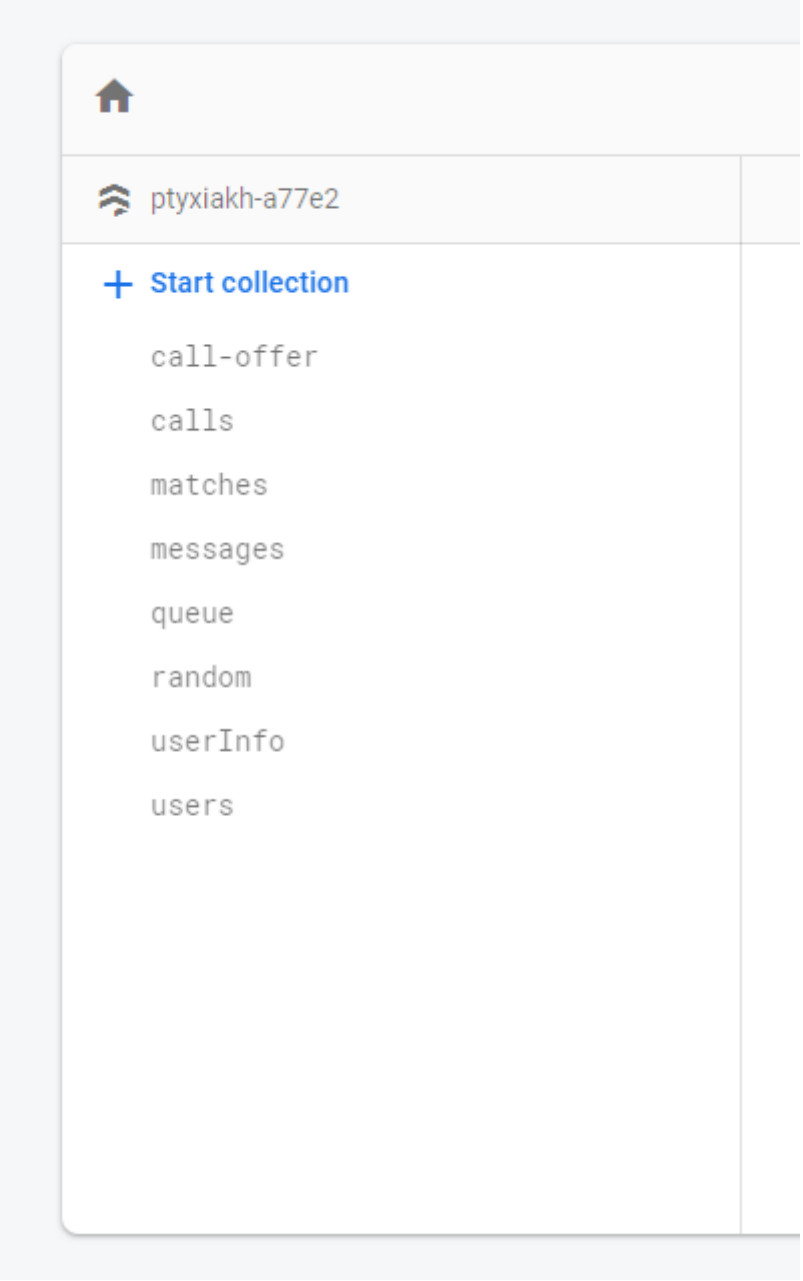
5.5.1 Firestore

Το Firestore αποθηκεύει τα δεδομένα σε μορφή εγγράφων, κάθε έγγραφο έχει τα πεδία του που μπορούν να είναι οποιουδήποτε τύπου όπως φαίνεται στην Εικόνα 5.2. Τα έγγραφα οργανώνονται σε συλλογές (collections), οι συλλογές είναι δομές στις οποίες οργανώνονται και αποθηκεύονται τα έγγραφα μέσω των οποίων μπορούν να γίνουν ερωτήματα στο Firestore. Επίσης κάθε έγγραφο πέρα από τα πεδία που

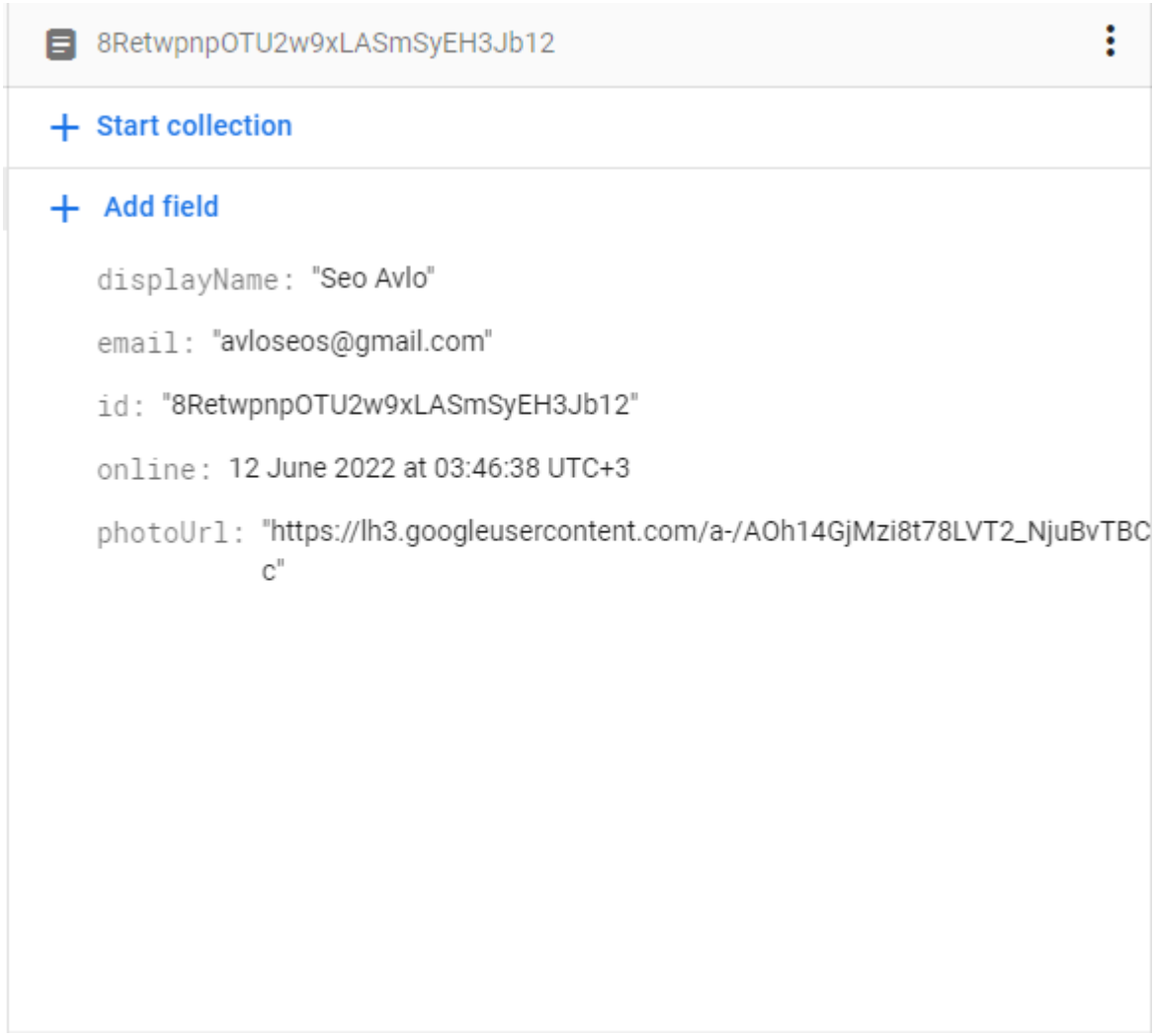
έχει μπορεί να περιέχει υπο-συλλογές. Αυτό σημαίνει ότι οι δομές που οργανώνονται και αποθηκεύονται τα δεδομένα είναι ιεραρχική και αυτό παρατηρείται καθώς μία συλλογή περιέχει πολλά έγγραφα, τα έγγραφα μπορούν να περιέχουν υπο-συλλογές και αυτές με την σειρά τους πολλά έγγραφα.

Στην εφαρμογή το Firestore χρησιμοποιείται για την αποθήκευση των δεδομένων, όπως για παράδειγμα μηνύματα, πληροφορίες χρηστών, σχέσεις μεταξύ χρηστών, κλήσεις.

Στην Εικόνα 5.1 φαίνονται τα collections που χρησιμοποιεί η εφαρμογή.



Εικόνα 5.1 Console - Firestore Collections



Εικόνα 5.2 Console User Document

5.5.1.1 users

Στο collection users, αποθηκεύονται έγγραφα τύπου User. Το κάθε έγγραφο περιέχει τα σημαντικά στοιχεία του χρήστη κατά την εγγραφή του στην εφαρμογή τα οποία αντλούνται από τον Google λογαριασμό του. Τα στοιχεία που αποθηκεύονται είναι το όνομα του (πεδίο displayName), email, το μοναδικό id του χρήστη, η ημερομηνία της τελευταίας φοράς που συνδέθηκε (πεδίο online) και η διεύθυνση URL της φωτογραφίας προφίλ του.

5.5.1.2 messages

Στην συλλογή messages αποθηκεύονται τα μηνύματα των χρηστών. Ένας χρήστης έχει μία εγγραφή στην messages, κάθε εγγραφή χρήστη περιέχει συλλογές από συνομιλίες στις οποίες αποθηκεύονται έγγραφα μηνυμάτων. Τα έγγραφα έχουν τα πεδία from που δείχνει το id του αποστολέα του μηνύματος και αντίστοιχα το πεδίο to που δείχνει το παραλήπτη. Επίσης έχουν ένα πεδίο για την καταγραφή της ώρας της αποστολής (timestamp πεδίο), ένα πεδίο που δείχνει τον τύπο του μηνύματος (type) και το πεδίο message που περιέχει το ίδιο το μήνυμα.

5.5.1.3 userInfo

Στο userInfo αποθηκεύονται έγγραφα που αφορούν τις πληροφορίες και τις σχέσεις που έχει ο χρήστης με άλλους χρήστες (για παράδειγμα οι φίλοι). Οι χρήστες έχουν από μία εγγραφή στο userInfo που περιέχει το όνομα του (πεδία firstName και lastName), το id του, την διεύθυνση της φωτογραφίας προφίλ του (photo) και ένα αντικείμενο με τους φίλους που έχει (friends). Το αντικείμενο friends περιέχει πληροφορίες που έχουν να κάνουν με τον φίλο όπως είναι το όνομα του, το id του, αν είναι συνδεδεμένος και πότε συνδέθηκε τελευταία φορά, η διεύθυνση της φωτογραφίας προφίλ του, καθώς και το τελευταίο μήνυμα και η ώρα του μηνύματος.

5.5.1.4 queue

Όταν ένας χρήστης θέλει να συνομιλήσει με κάποιον άγνωστο πρέπει να μπει στην ουρά αναμονής μέχρι να βρεθεί άτομο για συνομιλία, έτσι ο χρήστης κάνει μία εγγραφή στο collection queue με το id του ώστε το random cloud function (θα περιγραφεί η λειτουργία του παρακάτω) να τον ταιριάξει με κάποιον άλλο χρήστη για συνομιλία.

5.5.1.5 random

Στο random collection κάθε χρήστης έχει από ένα έγγραφο το οποίο παρατηρεί για αλλαγές ώστε αν υπάρξει κάποια αλλαγή στο έγγραφό και προστεθεί το πεδίο randomId να ενημερωθεί ότι βρέθηκε τυχαίο άτομο για συνομιλία. Το έγγραφο πέρα από το πεδίο randomId που είναι το id του τυχαίου ατόμου, έχει και το πεδίο friendRequest το οποίο εμφανίζεται μόνο όταν ένας από τους δύο στείλει αίτημα φιλίας στο άλλον. Το friendRequest πεδίο είναι αντικείμενο που έχει τα πεδία from που δείχνει από ποιον έγινε το αίτημα φιλίας και το πεδίο response που δείχνει την κατάσταση του αιτήματος φιλίας (για παράδειγμα unresponded).

5.5.1.6 matches

Στο matches collection εγγράφονται τα στοιχεία των χρηστών μετά από κάθε ταίριασμα μιας νέας συνομιλίας, ώστε να χρησιμοποιηθεί σωστά από το random cloud function. Κάθε έγγραφο περιέχει το id που προκύπτει από την ένωση των ids των δύο χρηστών και την ώρα που άρχισε η συνομιλία (match).

5.5.1.7 call-offer

Το call-offer χρησιμοποιείται από κάθε χρήστη για να δεχτεί κλήσεις. Κάθε χρήστης έχει μία εγγραφή με το id του, η εγγραφή περιέχει ένα collection το οποίο ονομάζεται calls. Στο calls ο χρήστης περιμένει για μία νέα εγγραφή ή αλλιώς μία προσφορά για κλήση (call offer). Η προσφορά κλήσης περιέχει το id του χρήστη που έκανε την προσφορά (πεδίο userId), την χρονική στιγμή που έκανε ο χρήστης την προσφορά (πεδίο timestamp), ένα πεδίο που λέγεται voiceOnly και δείχνει τον τύπο της κλήσης αν δηλαδή είναι κλήση ήχου ή κλήση βίντεο και τέλος ένα πεδίο offer του οποίου τα περιεχόμενα έχουν να κάνουν με το πρωτόκολλο WebRTC μέσω του οποίου επιτυγχάνονται οι κλήσεις και θα αναλυθεί σε επόμενο κεφάλαιο.

5.5.1.8 calls

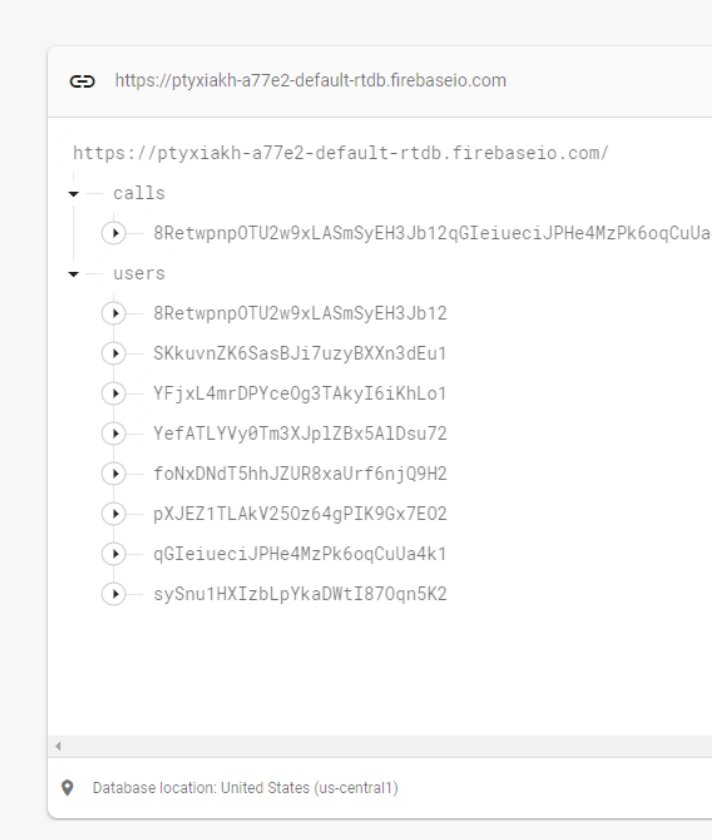
Το calls collection (είναι διαφορετικό από το calls πεδίο του call-offer εγγράφου) είναι το μέρος στο οποίο οι χρήστες κάνουν την διαδικασία του signaling δηλαδή την ανταλλαγή δεδομένων ώστε να επιτευχθεί η WebRTC κλήση και η ανταλλαγή πληροφοριών κατά την διάρκεια της κλήσης, όπως ανοιχτό - κλειστό μικρόφωνο/κάμερα. Ένα calls έγγραφο περιέχει τα ίδια πεδία που περιέχει ένα έγγραφο call-offer και επιπρόσθετα περιέχει: ένα αντίστοιχο πεδίο του offer που λέγεται answer και έχει να κάνει με την απάντηση του χρήστη, ένα πεδίο που δείχνει την χρονική στιγμή που δόθηκε η

απάντηση (answerTimestamp), δύο ίδια πεδία που κρατάνε πληροφορία για την κατάσταση του μικροφώνου και της κάμερα κάθε συμμετέχοντα της κλήσης (πεδία caller και callee), το πεδίο finishInfo που χρησιμοποιείται στον τερματισμό της κλήσης, με πεδία status και user τα οποία δείχνουν ποιος τερμάτισε την κλήση (user) και η πληροφορία του τερματισμού της κλήσης (status). Για την καλύτερη οργάνωση των κλήσεων άρα και της συλλογής calls η κλήσεις ανάμεσα σε δύο χρήστες καταγράφονται κάτω από την υπό-συλλογή κλήσεων στο έγγραφο με id το αποτέλεσμα της ένωσης των ids των συμμετεχόντων.

5.5.2 Real Time Database

Η Firebase Realtime Database αποθηκεύει τα δεδομένα σε μορφή JSON και είναι φτιαγμένη με τέτοιο τρόπο ώστε να επιτρέπει μόνο αλλαγές ή ερωτήματα τα οποία εκτελούνται γρήγορα. Επίσης τα δεδομένα συγχρονίζονται αυτόματα με τους συνδεδεμένους χρήστες στην βάση δεδομένων και αυτό επιτυγχάνεται με του μειωμένου χρόνου ερωτήματα που επιτρέπει και έτσι οι χρήστες λαμβάνουν μία πραγματικού χρόνου εμπειρία. Τα JSON objects περιέχουν κλειδιά και τιμές, μία τιμή μπορεί να είναι ένας απλός τύπος όπως ένας αριθμός ή να περιέχει επιπλέον JSON objects.

Στην Εικόνα 5.3 φαίνονται τα JSON objects που χρησιμοποιούνται στην εφαρμογή.



Εικόνα 5.3 Real Time DB

5.5.2.1 users object

Στο users json object αποθηκεύεται ένα object με το id του χρήστη που συνδέεται στην εφαρμογή, αυτό το object έχει ένα κλειδί που λέγεται online και σαν τιμή την κατάσταση που βρίσκεται ο χρήστης δηλαδή αν είναι συνδεδεμένος η τιμή είναι true και αν όχι η τιμή είναι false. Μόλις ο χρήστης αποσυνδεθεί από την εφαρμογή άρα και από την Realtime Database το online κλειδί θα πάρει την τιμή

false και έτσι θα ενημερωθούν οι υπόλοιποι κόμβοι που ενδιαφέρονται για αυτή την πληροφορία. Για την αλλαγή της τιμής σε false γίνεται χρήση της OnDisconnect κλάσης που επιτρέπει την εκτέλεση μιας γρήγορης αλλαγής στην βάση από την στιγμή που ο χρήστης αποσυνδεθεί. Με την OnDisconnect επιτυγχάνεται οι χρήστες να δέχονται τα σωστά δεδομένα για την κατάσταση ενός άλλου χρήστη καθώς ακόμα και αν ο χρήστης αντί για αποσύνδεση κλείσει την καρτέλα του browser που βρίσκεται τότε η σύνδεση με την βάση δεδομένων θα χαθεί και έτσι το online θα γίνει false. Η κατάσταση online απασχολεί και κάποια collection του Firestore, για αυτό παρακάτω θα αναλυθεί πως γίνεται ο συγχρονισμός των δύο βάσεων με χρήση των Firebase Cloud Functions.

5.5.2.2 calls object

Το calls JSON object έχει παρόμοια δομή με το calls collection, περιέχει διάφορα objects που έχουν να κάνουν με τις κλείσεις ανάμεσα σε δύο χρήστες με κλειδί το id που προκύπτει από τα id τους. Έπειτα έχει ένα calls object που με την σειρά του περιέχει τις κλείσεις των δύο χρηστών. Τα πεδία των call objects σε αυτή την περίπτωση δεν έχουν σημασία όπως είχαν στο users object, αυτό συμβαίνει γιατί η JSON διαδρομή calls χρησιμοποιείται μόνο για να δείξει ότι μία κλήση τέλειωσε απρόσμενα. Όταν ένας χρήστης κλείνει την καρτέλα που έχει ανοιχτεί και είναι σε κλήση με κάποιον τότε η κλήση πρέπει να τερματιστεί, για αυτό τον λόγο με την χρήση της OnDisconnect κλάσης και των Cloud Functions αρχικά θα γίνει μία εγγραφή στο calls object που θα δείχνει ότι η κλήση με το συγκεκριμένο id έχει τερματιστεί και έπειτα τον συγχρονισμό αυτής της πληροφορίας με το calls collection του Firestore θα αναλάβουν τα Cloud Functions.

5.6 Rules

Τα Firebase Rules βοηθούν στην επιβολή κανόνων για την προσβασιμότητα των δεδομένων. Για να γίνει κάποιο ερώτημα ή κάποια αλλαγή στα δεδομένα θα πρέπει ο χρήστης που κάνει αυτή τη λειτουργία να εφαρμόζει όλους τους κανόνες που υπάρχουν για παράδειγμα στο συγκεκριμένο collection στο Firestore database. Μερικά παραδείγματα κανόνων είναι: σε κάποια δεδομένα πρέπει να έχουν πρόσβαση μόνο χρήστες που είναι συνδεδεμένοι με την εφαρμογή ή ένας χρήστης θα μπορούσε να διαβάσει μόνο δεδομένα από ένα collection στο οποίο αναφέρεται το id του. Από την άλλη μεριά κάποια δεδομένα πρέπει να είναι ανοιχτά σε όλους τους χρήστες (συνδεδεμένους και μη) όπως για παράδειγμα η κατάσταση της εφαρμογής σε μία σελίδα που περιγράφει αν η εφαρμογή είναι σε θέση να χρησιμοποιηθεί.

5.7 Storage

Το Firebase Storage είναι μία υπηρεσία αποθηκευτικού χώρου που επιτρέπει την αποθήκευση βίντεο και εικόνων. Όπως και στα database, υπάρχει δυνατότητα επιβολής κανόνων είτε για το ανέβασμα (upload) είτε για το κατέβασμα (download) περιεχομένου. Κατά το ανέβασμα κάποιου περιεχομένου, το Storage δημιουργεί μία ηλεκτρονική διεύθυνση URL για την προσπέλαση του. Η εφαρμογή έχει δυνατότητα αποστολής φωτογραφιών ανάμεσα σε χρήστες, αυτό επιτυγχάνεται με το upload των φωτογραφιών στο storage από τον αποστολέα και έπειτα με τις διευθύνσεις URL για την προσπέλαση τους.

5.8 Hosting

Το Firebase Hosting προσφέρει ένα περιβάλλον για την φιλοξενία εφαρμογών και χρησιμοποιήθηκε για την φιλοξενία της εφαρμογής. Η διαδικασία του deploy ενός app στο Firebase είναι πολύ απλή και γρήγορη, ο προγραμματιστής όπως και στις υπόλοιπες λειτουργίες που παρέχει το Firebase δεν χρειάζεται να έχει επιπλέον γνώση πάνω στο αντικείμενο και χρειάζεται μόνο να δώσει μερικές απλές εντολές. Τέλος, το hosting δίνει επιλογές όπως αυτόματο deploy της εφαρμογής μετά από κάποια

αλλαγή που προκύπτει σε κάποιο Version Control System όπως είναι το Git το οποίο μπορεί κατά περιπτώσεις να είναι αρκετά βοηθητικό, με τον χρήστη να μην χρειάζεται να κάνει χειροκίνητα το deploy μετά από μία νέα έκδοση της εφαρμογής.

5.9 Cloud Functions

Τα Cloud Functions στο Firebase είναι functions που επιτρέπουν backend λειτουργίες. Δίνουν την δυνατότητα είτε για δημιουργία λειτουργιών που καλούνται χειροκίνητα από την πλευρά του client ή για λειτουργίες που τρέχουν αυτόματα (triggered) όταν συμβαίνει κάποια αλλαγή στην βάση (Firestore ή Realtime database). Για τις ανάγκες της εφαρμογής δημιουργήθηκαν τα random, online και callEnd Cloud Functions που θα αναλυθούν παρακάτω.

5.9.1 Απευθείας κλήση Cloud Function

Η Υπάρχουν αρκετοί τρόποι για να καλεστεί και να ξεκινήσει η λειτουργία ενός function όπως είναι η κλήση με HTTP ερωτήματα, η κλήση μέσω του client με το API του Firebase ή και να ρυθμιστούν ώστε να τρέχουν σύμφωνα με ένα συγκεκριμένο πρόγραμμα ώρας και ημέρας.

5.9.2 Κλήση Cloud Function με έναυσμα (trigger)

Για την αυτόματη κλήση των functions από trigger δηλαδή όταν προκύπτει κάποια αλλαγή στις βάσεις δεδομένων υπάρχουν 4 διαφορετικοί τρόποι – εναύσματα, οι οποίοι είναι το onCreate, onUpdate, onDelete, onWrite.

5.9.2.1 onCreate

Μία function που χρησιμοποιεί την onCreate, εκτελείται μόνο όταν δεδομένα γράφονται για πρώτη φορά στην περίπτωση του Realtime DB και αντίστοιχα όταν κάποιο έγγραφο γράφεται για πρώτη φορά στο Firestore.

5.9.2.2 onUpdate

Εκτελείται όταν δεδομένα που ήδη υπάρχουν αλλάζουν τιμή στην Realtime και αντίστοιχα όταν γίνεται κάποια αλλαγή σε ήδη υπάρχων έγγραφο στο Firestore.

5.9.2.3 onDelete

Εκτελείται όταν δεδομένα διαγράφονται στην Realtime και αντίστοιχα όταν διαγράφεται ένα έγγραφο στο Firestore.

5.9.2.4 onWrite

Η onWrite εκτελείται όταν εκτελούνται μία από τις προηγούμενες τρεις functions.

5.9.3 Functions που γράφτηκαν

Παρακάτω θα αναλυθούν οι random, online και η callEnded functions.

5.9.3.1 random function

Μία σημαντική λειτουργία της εφαρμογής είναι όταν ένας χρήστης θέλει να συνομιλήσει με κάποιον τυχαίο και αυτό επιτυγχάνεται μέσω του random Cloud Function. Όταν ένας χρήστης θέλει να επικοινωνήσει η εφαρμογή κάνει μία εγγραφή στην συλλογή queue με το id του, στην συνέχεια περιμένει και όταν βρεθεί κάποιο άτομο για συνομιλία τότε ενημερώνεται μέσω της εγγραφής του στην συλλογή random ότι βρέθηκε τυχαία συνομιλία. Η δουλειά της random είναι αρχικά να είναι ενήμερη για νέες εγγραφές στο queue collection μέσω της onCreate και να δημιουργήσει μία συνομιλία (match) για δύο χρήστες. Μόλις εκτελεστεί πρέπει να ελέγξει ότι υπάρχουν και άλλα άτομα στην λίστα ώστε να

προχωρήσει στους υπόλοιπους ελέγχους ώστε να γίνουν match δύο χρήστες. Έπειτα πρέπει να προσπελάσει έναν προς έναν τους χρήστες του queue με σειρά αύξουσα από τον παλαιότερο στην λίστα και να δει αν ισχύουν οι παρακάτω κανόνες:

1. Δεν είναι φίλοι
2. Δεν έχουν ξανά συνομιλήσει τα τελευταία 5 λεπτά, παίρνοντας την εγγραφή από την συλλογή matches με το id των δύο χρηστών και ελέγχοντας την ημερομηνία της τελευταίας τους συνομιλίας αν αυτή υπάρχει.

Μόλις ελεγχθούν και τα παραπάνω, ο πρώτος χρήστης που θα περάσει τους ελέγχους θα είναι το match δηλαδή ο χρήστης που θα συμμετέχει στην συνομιλία. Τέλος η random function βάζει αρχικά μία εγγραφή με την ημερομηνία στην συλλογή matches ώστε να μην ξαναγίνουν άμεσα match, διαγράφει και τους δύο χρήστες από την λίστα, ενημερώνει τις εγγραφές των δύο χρηστών στην random συλλογή και τέλος ενημερώνει τις συνομιλίες των δύο με μήνυμα ότι έχουν συνδεθεί.

5.9.3.2 online function

Το function online φροντίζει να είναι ενημερωμένη η κατάσταση (συνδεδεμένος ή όχι) ενός χρήστη στην βάση δεδομένων. Όταν ένας χρήστης μπαίνει στην εφαρμογή κάνει μία εγγραφή στο users object της Realtime DB και το ενημερώνει θέτοντας το online κλειδί σε true ότι είναι συνδεδεμένος και αντίστοιχα όταν βγαίνει από την εφαρμογή ενημερώνει πάλι το online ότι αποσυνδέθηκε δηλαδή το θέτει σε false. Το online function είναι υπεύθυνο να είναι ενήμερο για αυτή την αλλαγή που προέκυψε στην Realtime DB μέσω του onWrite και να ενημερώσει το Firestore για αυτή την αλλαγή. Έτσι όταν το online function δεχτεί μία αλλαγή στην κατάσταση ενός χρήστη θα πάρει την λίστα φίλων αυτού του χρήστη και θα ενημερώσει το αντίστοιχο userInfo εγγραφό τους, βάζοντας το friends πεδίο στο object που αντιστοιχεί στο id του την νέα κατάσταση.

5.9.3.3 callEnded function

Όπως αναφέρθηκε ήδη το calls object υπάρχει ώστε να αντιμετωπίζεται ένα απρόσμενο τέλος κλήσης όπως είναι το κλείσιμο της καρτέλας του browser κατά την διάρκεια της κλήσης. Για αυτό το λόγω όταν μία κλήση διακόπτεται απρόσμενα γίνεται μια εγγραφή στο callEnded object της Realtime DB και από εκεί και έπειτα η callEnded function μέσω της onCreate ενημερώνεται για το τέλος μιας κλήσης και ενημερώνει το αντίστοιχο έγγραφο στο Firestore για το τέλος της κλήσης.

Τέλος είναι σημαντικό να αναφερθεί ότι οι callEnd και η online Cloud Functions παίζουν τον ρόλο του συγχρονιστεί καθώς συγχρονίζουν τα δεδομένα μεταξύ των δύο βάσεων χωρίς να χρειάζεται να γίνει χρήση ενός εξωτερικού παράγοντα.

5.10 Επίλογος

Σε αυτό το κεφάλαιο αναλύθηκαν οι διάφορες υπηρεσίες που προσφέρει η Google μέσω του Firebase και οι ευκολίες που προκύπτουν με την χρήσης τους. Οι Firestore και Realtime DB προσφέρουν εναλλακτικές λύσεις που διαφέρουν από την κλασικές σχεσιακές βάσεις και μπορούν να χρησιμοποιηθούν εύκολα, γρήγορα αλλά και με ασφάλεια με τους κανόνες (Firebase Rules). Επίσης το Storage αφήνει τους προγραμματιστές να υποστηρίζουν δεδομένα διαφορετικού τύπου στην εφαρμογή τους δίνοντας ευκαιρίες για ανάπτυξη επιπλέον λειτουργιών (features). Η υπηρεσία του Authentication δίνει την δυνατότητα για εγγραφή χρηστών στην εφαρμογή από διαφορετικούς κόσμους (Google, Facebook κ.λπ.) χωρίς επιπλέον δυσκολία και με το Firebase Hosting για την φιλοξενία της εφαρμογής μπορούν να δημιουργηθούν ολοκληρωμένες εφαρμογές μόνο με την χρήση των υπηρεσιών του Firebase αλλά και μεμονωμένα χρησιμοποιώντας μόνο υπηρεσίες που έχει ανάγκη το εκάστοτε project. Τέλος τα Cloud Functions δίνουν την δυνατότητα για επέκταση των λειτουργιών του Firebase γράφοντας backend λειτουργίες για την επίτευξη των αναγκών που δεν είναι εύκολα υλοποιήσιμες χωρίς εξωτερικό παράγοντα.

Κεφάλαιο 6ο: WebRTC

6.1 Εισαγωγή

Το WebRTC είναι ένα API που προσφέρει peer to peer δυνατότητες πραγματικού χρόνου για μεταφορά εικόνας, ήχου και δεδομένων. Είναι ανοιχτού κώδικα και συντηρείται από την WebRTC ομάδα της Google. Υποστηρίζεται από όλους τους browsers όπως επίσης υπάρχουν αντίστοιχες υλοποιήσεις για Android και iOS, έτσι επιτρέπεται στους προγραμματιστές να αναπτύξουν εφαρμογές πραγματικού χρόνου αρκετά εύκολα. Στην συγκεκριμένη εφαρμογή γίνεται χρήση του WebRTC για την επίτευξη κλήσης ήχου και βίντεο.

Αυτό που κάνει το WebRTC ξεχωριστό είναι η κύρια του δυνατότητα ανταλλαγής δεδομένων όπως εικόνα και ήχος από browser σε browser χωρίς την άμεση διαμεσολάβηση κάποιου server. Ένας server χρησιμοποιείται για την διαδικασία του signaling και είναι γνωστός ως signaling server. Κατά την διαδικασία του signaling οι δύο πλευρές ανταλλάσσουν δεδομένα που έχουν να κάνουν με την σύνδεση τους και όχι με τα δεδομένα που μεταφέρονται.

6.2 Πως επιτυγχάνεται μία WebRTC επικοινωνία ανάμεσα σε δύο πλευρές (clients);

Αρχικά η μία πλευρά η οποία είναι ο caller δηλαδή αυτός που ξεκινάει την κλήση, δημιουργεί μία προσφορά ή offer όπως λέγεται και το αποθηκεύει στον Signaling Server. Το offer περιγράφει την peer to peer σύνδεση και περιέχει πληροφορίες για αυτή. Η απέναντι πλευρά δηλαδή ο callee θα διαβάσει το offer από τον Signaling Server και θα απαντήσει με την σειρά της με την δημιουργία μιας απάντησης (answer) και την αποθηκεύσει αυτής στον Server ώστε να το διαβάσει ο caller. Από εκείνο το σημείο οι δύο πλευρές ανταλλάσσουν συντεταγμένες μέσω του Server για να συνδεθούν η μία στην άλλη. Οι συντεταγμένες λέγονται ICE (Interactive Connectivity Establishment) και είναι χρήσιμές καθώς η ανίχνευση των συσκευών είναι δύσκολη από διαφορετικά δίκτυα για διάφορους λόγους όπως οι δυναμικές IP. Έτσι το WebRTC σε κάθε πλευρά χρησιμοποιεί τις συντεταγμένες κάνοντας ερωτήματα σε έναν STUN Server για να αποφασίσει ποια συντεταγμένη είναι η καλύτερη για την σύνδεση.

6.3 Βασικά σημεία του WebRTC

Κατά την WebRTC peer to peer επικοινωνία όπως έγινε ήδη κατανοητό, εμπλέκονται διάφορες κλάσεις, έννοιες και τεχνολογίες. Παρακάτω θα αναφερθούν τα σημαντικότερα αντικείμενα του WebRTC που χρησιμοποιήθηκαν στην εφαρμογή και είναι απαραίτητα για την επίτευξη μιας peer σύνδεσης.

6.3.1 Media

Για την μεταφορά εικόνας, ήχου και δεδομένων χρησιμοποιείται το αντίστοιχο υλικό της συσκευής για την άντληση αυτής την πληροφορίας όπως είναι το μικρόφωνο ενός κινητού. Το WebRTC μπορεί να χρησιμοποιήσει το μικρόφωνο, την κάμερα όπως και την οθόνη της συσκευής για την μετάδοσης τους, μέσω του αντικειμένου navigator.mediaDevices.

Για την άντληση είτε του μικροφώνου είτε της κάμερα υπάρχει η μέθοδος getUserMedia του navigator.mediaDevices και αντίστοιχα η getDisplayMedia για την άντληση της οθόνης της συσκευής. Με την κλήση των παραπάνω μεθόδων επιστρέφεται ένα αντικείμενο τύπου MediaStream το οποίο περιέχει την πληροφορία της συσκευής όπως για παράδειγμα ο ήχος από την συσκευή του μικροφώνου. Προτού επιστραφεί το MediaStream ο browser θα εμφανίσει στον χρήστη ένα ερώτημα για άδεια

χρήσης της συγκεκριμένης συσκευής, αν ο χρήστης αρνηθεί ή για κάποιο λόγο το υλικό δεν είναι διαθέσιμο προς χρήση τότε θα προκύψει αντίστοιχο σφάλμα.

6.3.1.1 Media Constraints

Οι μέθοδος getUserMedia δέχεται μία παράμετρο τύπου MediaStreamConstraints που ορίζει τους περιορισμούς που θα έχει το MediaStream. Οι περιορισμοί ορίζονται ως ένα object με κλειδιά και τιμές όπως το object {video: true, audio: true} το οποίο χρησιμοποιείται στις κλήσεις της εφαρμογής και ορίζει ότι το WebRTC θέλει να χρησιμοποιήσει την κάμερα και το μικρόφωνο της συσκευής. Επιπλέον περιορισμοί θα μπορούσε να είναι η ποιότητα του μικροφώνου ή οι διαστάσεις της εικόνας που επιστρέφει η κάμερα.

6.3.1.2 Media Streams

Ένα MediaStream αντικείμενο αντιπροσωπεύει μία ροή περιεχομένου πολυμέσου όπως είναι ο ήχος και το βίντεο. Κάθε MediaStream περιλαμβάνει διάφορα κομμάτια που το καθένα κουβαλά ξεχωριστά την πληροφορία του κάθε μέσου, αυτό σημαίνει ότι αν ένα MediaStream περιέχει εικόνα και ήχο τότε σε άλλο κομμάτι θα περιέχεται η εικόνα και σε άλλο ο ήχος. Για την παρουσίαση και εμφάνιση των MediaStreams στον browser πρέπει να δοθούν ως πηγή (src χαρακτηριστικό) σε ένα HTML στοιχείο το οποίο τα υποστηρίζει όπως είναι τα video και audio στοιχεία.

6.3.2 RTCPeerConnection

Το RTCPeerConnection αντικείμενο αντιπροσωπεύει και διαχειρίζεται μία peer σύνδεση, η οποία συνδέει τους δύο clients για να επικοινωνήσουν. Ένας χρήστης αρχικά πρέπει να δημιουργήσει ένα αντικείμενο RTCPeerConnection δίνοντας τους ως παραμέτρους ένα RTCConfiguration που περιέχει πληροφορίες για τους STUN Servers. Στην συνέχεια ο χρήστης που ξεκινάει την σύνδεση (caller) πρέπει να δημιουργήσει ένα offer - προσφορά τύπου RTCSessionDescription και να το δώσει στην απέναντι πλευρά (callee) όπου ο callee μόλις λάβει το RTCSessionDescription αντικείμενο θα δημιουργήσει αντίστοιχα μία απάντηση - answer και θα την δώσει πίσω στον caller.

6.3.3 Offer

Η προσφορά του caller δημιουργείται με την μέθοδο createOffer του RTCPeerConnection αντικειμένου και πρέπει να καλεστεί αφού δημιουργηθεί το peer connection. Το αντικείμενο RTCSessionDescription που επιστρέφει η createOffer αντιπροσωπεύει την μία πλευρά της σύνδεσης και περιέχει ένα SDP αντικείμενο (Session Description Protocol) και έναν τύπο που δείχνει αν είναι offer ή answer. Το SDP χρησιμοποιείται στις peer to peer συνδέσεις για την περιγραφή τους.

Μόλις δημιουργηθεί η προσφορά πρέπει να ενημερωθεί το peer connection για την ύπαρξη της τοπικής περιγραφής - προσφοράς χρησιμοποιώντας την μέθοδο setLocalDescription ως εξής peerConnection.setLocalDescription(offer) και έπειτα θα αποσταλεί στον Signaling Server.

6.3.4 Answer

Η απάντηση του callee δημιουργείται παρομοίως με το offer, χρησιμοποιώντας την μέθοδος peerConnection.createAnswer() αντ' αυτού. Για να είναι σε θέση όμως ο callee να απαντήσει πρέπει πρώτα να λάβει την προσφορά από τον Signaling Server και να ενημερώσει την peer σύνδεση ότι υπάρχει απομακρυσμένη περιγραφή peer σύνδεσης (προσφορά) με την εντολή peerConnection.setRemoteDescription(newRTCSessionDescription(offer)). Αφού θέσει την

απομακρυσμένη περιγραφή τότε θα δημιουργήσει την απάντηση και θα ενημερώσει εκ νέου την σύνδεση, αυτή την φορά με την τοπική περιγραφή.

Με το πέρας της παραπάνω διαδικασίας η απάντηση θα αποσταλεί στον Signaling Server για να την διαβάσει από εκεί ο caller και να θέσει με την σειρά του την απομακρυσμένη περιγραφή.

6.3.5 Signaling Server

Παραπάνω αναφέρθηκε ο Signaling Server, η ουσιαστική λειτουργία του είναι μόνο για την ανταλλαγή πληροφοριών που έχουν να κάνουν με το πως θα πραγματοποιηθεί η σύνδεση ανταλλάσσοντας τις περιγραφές προσφοράς - απάντησης και τις συντεταγμένες ICE. Στην συγκεκριμένη εφαρμογή τον ρόλο του Signaling Server παίζει το Firebase Firestore με τα calls και call-offer collection με τα οποία γίνονται οι παραπάνω ανταλλαγές δεδομένων μεταξύ των clients.

6.3.6 ICE συντεταγμένες

Οι ICE συντεταγμένες χρησιμοποιούνται ώστε να μπορούν οι δύο πλευρές να εντοπίσουν η μία την άλλη, καθώς οι IP διευθύνσεις των συσκευών αλλάζουν συνεχώς λόγω του NAT και το τείχος προστασίας που βρίσκονται κάνει δύσκολη την ανίχνευση των συσκευών από το εξωτερικό του δίκτυο. Οι ICE συντεταγμένες κάθε client αποθηκεύονται στις υπο-συλλογές offerICECandidates και answerICECandidates της εγγραφής της κλήσης στο Firestore.

6.3.7 STUN Server

Ο STUN Server λαμβάνει ICE συντεταγμένες και βοηθά να βρεθεί η δημόσια διεύθυνση της συσκευής. Επίσης βοηθά να βρεθούν περιορισμοί που μπορεί να προκύπτουν από το δίκτυο και αν η συσκευή είναι διαθέσιμη. Να σημειωθεί ότι συνήθως χρησιμοποιούνται STUN Servers από δωρεάν και αξιόπιστους παρόχους όπως είναι η Google και οι servers που χρησιμοποιούνται στην διπλωματική είναι οι:

- stun:stun1.l.google.com:19302
- stun:stun2.l.google.com:19302

6.4 Υλοποίηση WebRTC κλήσεων στην εφαρμογή

Για τις ανάγκες της αρχιτεκτονικής της εφαρμογής και για την επίτευξη μιας κλήσης με το WebRTC δημιουργήθηκαν κάποιες βασικές έννοιες σύμφωνα με τις οποίες πραγματοποιούνται οι κλήσεις. Αρχικά δημιουργήθηκε το interface CallClient το οποίο όπως φαίνεται στην Εικόνα 6.1 έχει το πεδίο type και τις μεθόδους start και finish προς υλοποίηση και θα χρησιμοποιηθεί για την αρχή και το τέλος των κλήσεων. Υπάρχει το Signaling interface που επεκτείνεται από το WebRTCSignaling (Εικόνα 6.2) interface και έχει να κάνει με τις λειτουργίες που θα εκτελέσει ο Signaling Server. Επίσης υπάρχει η κλάση WebRTCClient που υλοποιεί το WebRTCCall (Εικόνα 6.3) και δείχνει τις λειτουργίες του WebRTC.

Ο Signaling Server θα είναι το Firestore του Firebase οπότε οι λειτουργίες του WebRTCSignaling interface θα υλοποιηθούν από την κλάση FirebaseSignaling. Το interface CallClient υλοποιείται έμμεσα δύο φορές, μία για την απάντηση μιας κλήσης (κλάση AnswerWebRTCFirebaseCall) και μία για την έναρξη μιας κλήση (κλάση OfferWebRTCFirebaseCall). Πρώτα όμως (άμεσα) υλοποιείται από την αφηρημένη κλάση WebRTCFirebaseCallClient και εκτελεί μερικές λειτουργίες που είναι κοινές στις υπόλοιπες κλάσεις που την επεκτείνουν όπως είναι η δημιουργία του WebRTCClient και τον FirebaseSignaling Server που θα χρησιμοποιηθούν στην συγκεκριμένη κλήση.

```
export interface CallClient {
    readonly type: CallType;

    start(): void;

    finish(): void;
}
```

Εικόνα 6.1 CallClient

```
export interface WebRTCSignaling extends Signaling {
    addCandidate(event: RTCPeerConnectionIceEvent): Observable<void>;

    addOffer(offerDescription: RTCSessionDescriptionInit): Observable<void>;

    addAnswer(answerDescription: RTCSessionDescriptionInit): Observable<void>;

    listenForRemoteIceCandidates(): Observable<RTCIceCandidateInit>;

    listenForRemoteAnswer(): Observable<RTCSessionDescription>;
}
```

Εικόνα 6.2 WebRTCSignaling

```
export interface WebRTCCall {

    getUserMedia(): Observable<MediaStream>;

    listenForRemoteTracks(): void;

    listenForIceCandidates(): Observable<RTCPeerConnectionIceEvent>;

    setLocalDescriptionToRTCPeerConnection(description: RTCLocalSessionDescriptionInit): Observable<void>;

    setRemoteDescriptionToRTCPeerConnection(description: RTCSessionDescriptionInit): Observable<void>;

    setRemoteIceCandidateToRTCPeerConnection(candidate: RTCIceCandidateInit): Observable<void>;

    setLocalStreamTracksToRTCPeerConnection(stream: MediaStream): void;

    createOffer(): Observable<RTCSessionDescriptionInit>;

    createAnswer(): Observable<RTCSessionDescriptionInit>;

    setVideo(video: boolean): void;

    setMic(mic: boolean): void;

    localStream(): MediaStream;

    remoteStream(): MediaStream;

    destroy(): void;
}
```

Εικόνα 6.3 WebRTCCall

6.4.1 Αρχή κλήσης

Για την αρχή της κλήσης χρησιμοποιείται η κλάση OfferWebRTCFirebaseCall και καλείται η μέθοδος start(). Έτσι αφού αντλούνται τα MediaStreams (το ένα είναι το τοπικό που προέρχεται από την συσκευή και το άλλο δημιουργήθηκε εσωτερικά στον WebRTCClient) από τον WebRTCClient μέσω της getUserMedia, δίνονται στο MediaService (θα εξηγηθεί παρακάτω η λειτουργία του), θέτονται τα κομμάτια του τοπικού MediaStream στην peer σύνδεση για να μπορέσει να τα μεταδώσει όταν γίνει το signaling. Έτσι αρχίζει η παρατήρηση για απομακρυσμένα κομμάτια από την σύνδεση ώστε να ενημερωθεί το αντικείμενο του remoteStream. Ξεκινά η παρατήρηση για τις ICE συντεταγμένες που δημιουργούνται αυτόματα και αποθηκεύονται μέσω του signaler στον server και επιπλέον αρχίζει η παρατήρηση του εγγράφου της κλήσης ώστε να ενημερώνεται ο WebRTCClient όταν προκύπτει αλλαγή στην κατάσταση του μικροφώνου και της κάμερας.

Μόλις ενεργοποιηθούν οι παραπάνω λειτουργίες, δημιουργείται η προσφορά κλήσεις, αποθηκεύεται στον Signaling Server και ενημερώνεται η peer σύνδεση για την τοπική περιγραφή. Τέλος ενεργοποιούνται επιπλέον παρατηρητές (observer) οι οποίοι περιμένουν από την απέναντι πλευρά για απάντηση ώστε να ενημερώσουν την peer σύνδεση με την απομακρυσμένη περιγραφή - απάντηση και παράλληλα περιμένουν για απομακρυσμένες ICE συντεταγμένες ώστε να ενημερώσουν εκ νέου την peer σύνδεση.

6.4.2 Απάντηση κλήσης

Για την απάντηση εισερχομένων κλήσεων είναι υπεύθυνη η κλάση AnswerWebRTCFirebaseCall και η λειτουργίας της είναι παρόμοια με την κλάση OfferWebRTCFirebaseCall. Τα πρώτα βήματα που συμβαίνουν κατά την απάντηση μιας κλήσης είναι ίδια με της αρχής της κλήσης, δηλαδή αφού γίνει κλήση της μεθόδου start και δημιουργηθούν τα MediaStreams δίνονται στο MediaService, ενημερώνεται η peer σύνδεση για το τοπικό MediaStream και ξεκινούν οι observers για τα απομακρυσμένα κομμάτια περιεχομένου (remote tracks), του εγγράφου της κλήσης και των ICE συντεταγμένων.

Έπειτα γίνεται ενημέρωση της σύνδεσης με την προσφορά του χρήστη και αρχίζει η δημιουργία της απάντησης με την μέθοδο createAnswer. Μόλις δημιουργηθεί η απάντηση, δίνεται στην σύνδεση ως τοπική περιγραφή και αποστέλλεται στον Signaling Server. Τέλος ξεκινά η παρατήρηση για απομακρυσμένες ICE συντεταγμένες όπως συμβαίνει και στην αρχή της κλήσης.

6.4.3 Media Service

Το MediaService χρησιμοποιείται για την μεταφορά περιεχομένου προς το VIEW επίπεδο και πιο συγκεκριμένα με την WebRTCMediaService υλοποίηση του, την μεταφορά των MediaStreams από το επίπεδο του WebRTCFirebaseClient στον React Component Calling. Ο λόγος ύπαρξης αυτού του Service είναι ότι η αρχιτεκτονική είναι φτιαγμένη να στηρίζεται σε αφηρημένους δεσμούς και τα MediaStreams είναι μία έννοια που είναι αρκετά συγκεκριμένη και έχει να κάνει κυρίως με τον browser. Αν τα MediaStreams κυκλοφορούσαν μέσα σε όλα τα επίπεδα της εφαρμογής χωρίς το MediaService τότε πιθανόν να υπήρχε πρόβλημα συμβατότητας αν για παράδειγμα η εφαρμογή από Web γινόταν native εφαρμογή με χρήση React Native και τα αντικείμενα τύπου MediaStream δεν υπήρχαν στην native τεχνολογία θα έπρεπε να γραφτεί μεγάλο μέρος της εφαρμογής που έχει να κάνει με τις κλήσεις από την αρχή, ενώ με την χρήση του MediaService θα χρειαζόταν να γραφτεί εκ νέου μόνο η υλοποίηση του WebRTCMediaService.

6.4.4 Επιπλέον πληροφορίες

6.4.4.1 CallOptions

Κάθε CallClient στον δομητή δέχεται ένα αντικείμενο τύπου CallOptions που περιέχει πληροφορίες για την συγκεκριμένη κλήση όπως είναι ο τύπο της κλήσης (offer/answer), τα id των συμμετεχόντων και ποιος είναι ο caller, αν η κλήση είναι μόνο ήχου ή και εικόνας και στην περίπτωση που ο CallClient είναι τύπου AnswerWebRTCFirebaseCall τότε περιέχει επιπλέον την προσφορά - offer.

6.4.4.2 CallClienFactories

Για να συνάδει η αρχιτεκτονική των κλήσεων με τους κανόνες και την φιλοσοφία της υπόλοιπης της εφαρμογής υπάρχουν κάποια Factories τύπου CallClientFactory Εικόνα 6.4 τα οποία βοηθούν στην δημιουργία των OfferWebRTCFirebaseCall και AnswerWebRTCFirebaseCall αντικειμένων και κρατούν κάθε κομμάτι της αρχιτεκτονικής αφηρημένο και απομονωμένο από συγκεκριμένες τεχνολογίες και έννοιες όπως είναι το WebRTC και το Firebase.

```
export interface CallClientFactory<C extends CallOptions> {
    readonly type: CallType
    create(options: C): CallClient;
}
```

Εικόνα 6.4 CallClientFactory

6.5 Επίλογος

Σε αυτό το κεφάλαιο αναλύθηκαν τα βασικότερα σημεία του WebRTC. Χρησιμοποιώντας τις κλάσεις και τα API που προφέρει, μπορούν να αναπτυχθούν σημαντικές λειτουργίες που αφορούν τις πραγματικού χρόνου συνδέσεις από μεταφορά εικόνας και ήχου μέχρι και μεταφορά δεδομένων ανάμεσα σε συσκευές που βρίσκονται οπουδήποτε στο Internet. Επίσης αναλύθηκε πως επιτυγχάνεται μία peer σύνδεση ανάμεσα σε δύο συσκευές και συγκεκριμένα πως επιτυγχάνονται οι κλήσεις στην συγκεκριμένη εφαρμογή.

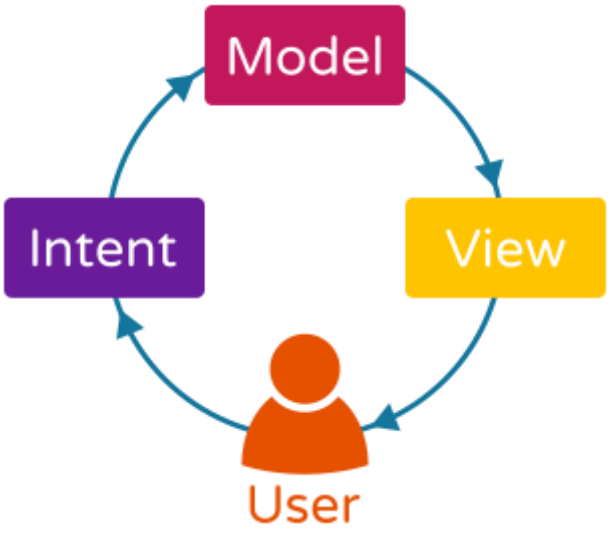
Κεφάλαιο 7ο: Model View Intent (MVI)

7.1 Εισαγωγή

Όπως ήδη αναφέρθηκε η εφαρμογή χρησιμοποιεί την βιβλιοθήκη ReactJS για την δημιουργία των user interfaces δηλαδή των διεπαφών μέσω των οποίον ο χρήστης αλληλεπιδρά με την εφαρμογή. Όπως επίσης προαναφέρθηκε η εφαρμογή είναι χωρισμένη σε τρία επίπεδα (VIEW - PRESENTATION - BUSINESS LOGIC), έτσι σε αυτό το κεφάλαιο θα ασχοληθούμε με την παρουσίαση του Model - View - Intent (MVI) pattern και πως χρησιμοποιήθηκε κατά την δημιουργία της εφαρμογής.

7.2 Τι είναι το Model-View-Intent

Το MVI (Model - View - Intent) είναι ένα αρχιτεκτονικό μοτίβο (Architectural Pattern) με σημαντικότερα χαρακτηριστικά να είναι η μονόδρομη ροή δεδομένων ανάμεσα στα επίπεδα της αρχιτεκτονικής της εφαρμογής, τα αμετάβλητα μοντέλα και ο αντιδραστικός (reactive) του χαρακτήρας στις αλλαγές που προκύπτουν. Στο MVI το Intent ή αλλιώς η πρόθεση όπως και η σημασία της λέξης δηλώνει μια πρόθεση του χρήστη να κάνει κάτι στην εφαρμογή (π.χ. το πάτημα ενός κουμπιού). Το μοντέλο (Model) πρέπει να είναι αμετάβλητο και το View να υλοποιείται από ένα κομμάτι της εφαρμογής που δημιουργεί κάποια διεπαφή όπως ένας component στην ReactJS.



Εικόνα 7.1 Model View Intent

7.3 Πως λειτουργεί και εφαρμόζεται

Κάθε View έχει μία κατάσταση (ViewState) η οποία αντιστοιχεί σε αυτό που βλέπει ο χρήστης στο View και έναν Presenter ο οποίος είναι υπεύθυνος για την δημιουργία αυτής της κατάστασης (State). Η κατάσταση ενός View είναι μια δομή η οποία περιέχει μοντέλα που έχουν να κάνουν με το συγκεκριμένο View. Στην Εικόνα 7.2 το MessagesState περιέχει μία λίστα από Message αντικείμενα (το Message είναι μοντέλο της εφαρμογής) τα οποία εκτυπώνονται στην οθόνη του Messages Component (MessagesView). Το View δημιουργεί Intents (για λόγους ευκολίας τα Intents στο MVI ονομάζονται και Events ή ViewEvents) ώστε να αλλάξει την κατάσταση του View, όπως στην Εικόνα 7.3 το SendMessageEvent θα αλλάξει την κατάσταση αποστέλλοντας μήνυμα σε έναν χρήστη δηλαδή προσθέτοντας ένα ακόμα Message αντικείμενο στην λίστα μηνυμάτων του MessagesState. Ο Presenter

είναι υπεύθυνος να λάβει τα Events και σύμφωνα με τον τύπο και την πληροφορία που περιέχει το εκάστοτε Event να δημιουργήσει μία καινούργια κατάσταση για το View. Αν για την δημιουργία της νέας κατάστασης είναι αναγκαία η δημιουργία νέων μοντέλων ο Presenter επικοινωνεί με το business επίπεδο καθώς είναι το μοναδικό επίπεδο της εφαρμογής στο οποίο γίνεται η δημιουργία των μοντέλων, έτσι μεταφράζει τα events σε μία δομή την οποία αναγνωρίζει το business επίπεδο και την παραδίδει σε αυτό με αποτέλεσμα την δημιουργία του νέου μοντέλου σύμφωνα με τα δεδομένα της δομής. Εν συνεχεία ο Presenter μετά την δημιουργία του νέου μοντέλου, δημιουργεί νέο State με το view να εμφανίζει στην οθόνη την νέα πληροφορία.

```
export class MessagesState implements State {
    private readonly _messages: Message[];

    constructor(messages: Message[]) {
        this._messages = messages;
    }

    get messages(): Message[] {
        return this._messages;
    }
}
```

Εικόνα 7.2 Message State

```
export class SendMessageEvent implements Event {
    private readonly _message: string;
    private readonly _timestamp: Date;

    constructor(message: string, timestamp: Date) {
        this._message = message;
        this._timestamp = timestamp;
    }

    get message(): string {
        return this._message;
    }

    get timestamp(): Date {
        return this._timestamp;
    }
}
```

Εικόνα 7.3 Send Message Event

Με την παραπάνω διαδικασία γίνεται ξεκάθαρο ότι έχουμε αμετάβλητα μοντέλα καθώς ένα μοντέλο δεν αλλάζει σύμφωνα με ένα intent αλλά ξαναδημιουργείται με αποτέλεσμα να επιτυγχάνεται η μονόδρομη ροή δεδομένων καθώς το μοναδικό μέρος της αρχιτεκτονικής που μπορεί να κάνει αλλαγές στην κατάσταση της εφαρμογής είναι το business. Τέλος ή αρχιτεκτονική είναι αντιδραστική καθώς ο Presenter είναι συνεχώς σε παρατήρηση (observe) για Events από το View και αντιδρά αναλόγως όταν παρατηρήσει καινούριο event, το View είναι συνεχώς σε παρατήρηση (observe) για νέο State (αντιδρώντας με την αλλαγή της κατάστασης της οθόνη) από τον Presenter. Το model επίσης αντιδρά σύμφωνα με την πληροφορία που λαμβάνει από τον Presenter με την δημιουργία του νέου μοντέλου και ο Presenter παρατηρεί συνεχώς για την δημιουργία νέου μοντέλου και αντιδρά δημιουργώντας νέα κατάσταση.

Επιπλέον, το αντικείμενο της κατάστασης βοηθά στην επιδιόρθωση πολλών προβλημάτων που παρουσιάζουν άλλα πρότυπα καθώς η πληροφορία που περιέχει το αντικείμενο μεταφράζεται στην διεπαφή που βλέπει ο χρήστης. Έτσι βοηθά τον προγραμματιστή κατά τον εντοπισμό σφαλμάτων (debugging) και την αποφυγή λογικών λαθών στο View επίπεδο, με το αμετάβλητο μοντέλο και την μονόδρομη ροή είναι πιο εύκολο να βρεθεί σε ποιο σημείο έγινε λάθος και προέκυψε αυτή η κατάσταση.

Όταν δημιουργείται ένα View, στέλνει στον Presenter ένα ScreenViewEvent το οποίο δείχνει στον Presenter ότι αυτό το View δημιουργήθηκε και πλέον είναι σε θέση να λάβει καταστάσεις, για παράδειγμα το MessagesScreenViewEvent δείχνει στον MessagesPresenter ότι το MessagesComponent δημιουργήθηκε και είναι έτοιμο για νέα States.

Ακόμα ένα κομμάτι που αφορά την επικοινωνία μεταξύ View-Presenter είναι η σύνδεση (attach) και η αποσύνδεση (detach) του View στον Presenter. Κατά την δημιουργία ενός View πριν σταλεί το ScreenViewEvent, καλείται η μέθοδος attachView του Presenter από το View και αντίστοιχα κατά το τέλος ζωής του (καταστροφή) καλείται η detachView. Ο Presenter έχει τις μεθόδους attachView και detachView έτσι ώστε να γνωρίζει σε ποιο View στέλνει τα δεδομένα και επίσης με ποιο View είναι συνδεδεμένος.

Στην Εικόνα 7.4 βλέπουμε το interface που περιέχει την υπογραφή των Presenters της εφαρμογής. Το View πέρα από τις attachView και detachView χρησιμοποιεί επίσης την μέθοδο publishEvent με την οποία δημοσιεύει (publish) τα ViewEvents και την μέθοδο states για την παρατήρηση των νέων καταστάσεων που δημιουργεί ο Presenter.

```
export interface Presenter<S extends State> {

    attachView(view: any): void;

    publishEvent(event: Event): void;

    states(): Observable<S>;

    detachView(): void;
}
```

Εικόνα 7.4 Presenter

```typescript
export abstract class ViewPresenter<S extends State> implements Presenter<S> {

    protected _view: any;
    private _events: Subject<Event> | undefined;
    protected _states: Subject<S>;
    protected _lastState: S;
    private _eventsSubscription: Subscription;

    protected constructor() {...}

    attachView(view: any): void {
        this._view = view;
        this._events = new Subject<Event>();
        this._states = new Subject<S>();
    }

    publishEvent(event: Event): void {
        if (!this._events) {
            return;
        }

        this._events.next(event);
    }

    states(): Observable<S> {
        if (!this._events) {
            return EMPTY;
        }

        this._eventsSubscription = this._events.pipe(
            mergeMap( project: event => this.onEvent(event)),
            tap( next: state => this._states.next(state))
        ).subscribe();

        return this._states.pipe(
            tap( next: state => this._lastState = state)
        );
    }

    protected abstract onEvent(event: Event): Observable<S>;

    detachView(): void {
        this._view = null;
        if (this._events) {
            this._events.complete();
            this._events = undefined;
        }
        if (this._eventsSubscription) {
            this._eventsSubscription.unsubscribe();
        }

        if (this._states) {
            this._states.complete();
            this._states = undefined;
        }

        this._lastState = undefined;
    }
}
```

Εικόνα 7.5 ViewPresenter

Για λόγους αρχιτεκτονικής, επαναχρησιμοποίησης και ευκολίας το Presenter interface δεν υλοποιείται άμεσα από κάθε Presenter αλλά αρχικά υλοποιείται από την αφηρημένη κλάση (abstract) ViewPresenter όπως φαίνεται στην Εικόνα 7.5 και έπειτα ο ViewPresenter επεκτείνεται από τους Presenters. Αυτό συμβαίνει γιατί οι λειτουργίες των μεθόδων που περιγράφηκαν πρέπει να είναι ίδιες παντού και με την κλάση ViewPresenter μπορούν να υλοποιηθούν μία φορά και να επαναχρησιμοποιηθούν από τους υπόλοιπους. Έτσι πλέον κάθε καινούρια Presenter κλάση πρέπει να υλοποιεί μόνο την onEvent μέθοδο κατά την οποία όπως βλέπουμε στην Εικόνα 7.5 όταν γίνεται publish ένα Event, αυτό δίνεται στην onEvent και ανάλογα με τον τύπο και την πληροφορία που περιέχει γίνονται οι κατάλληλες αλλαγές στο μοντέλο με απώτερο σκοπό την αλλαγή της κατάστασης άρα και την δημιουργία νέας κατάστασης.

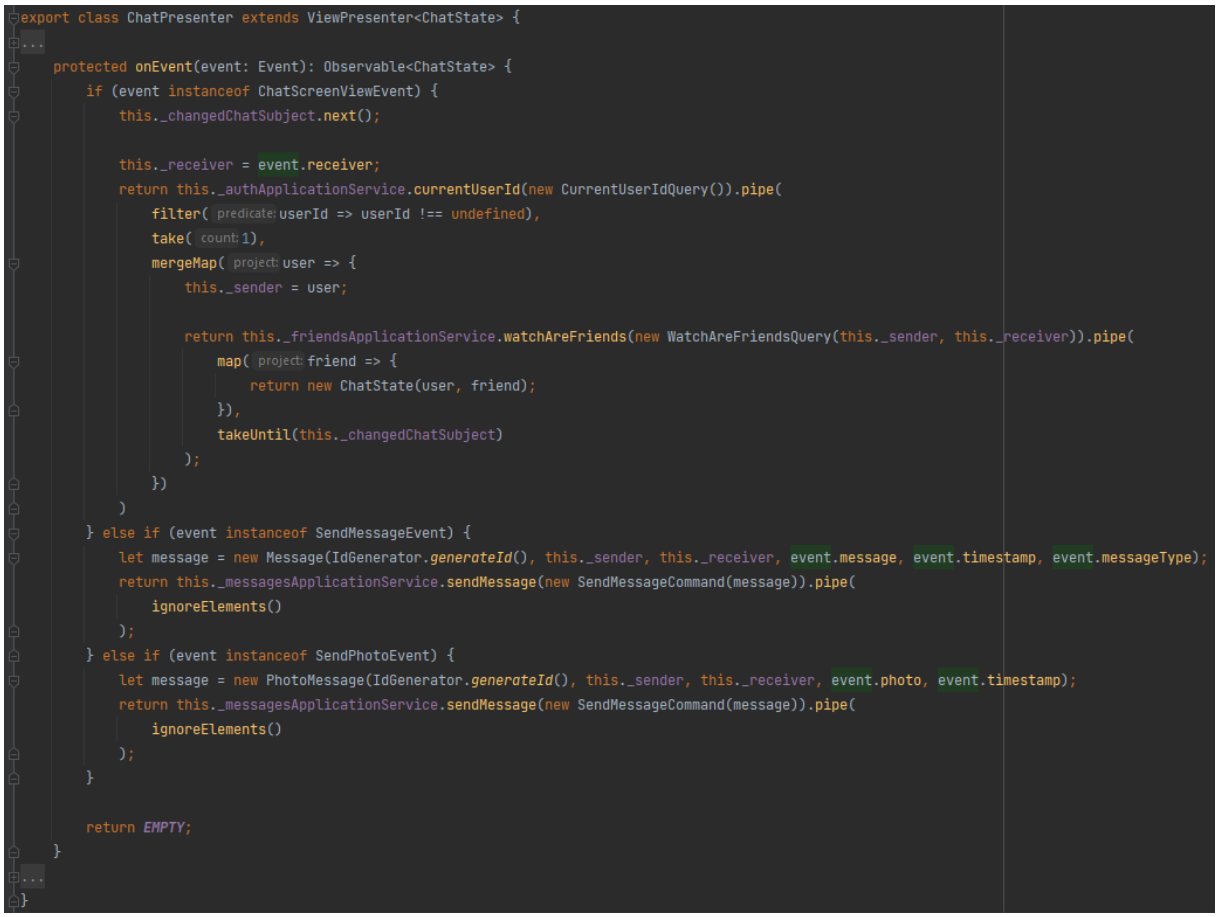
Στην Εικόνα 7.6 παρουσιάζεται ο ChatPresenter ο οποίος επεκτείνει την ViewPresenter κλάση και υλοποιεί την λειτουργία της onEvent. Χρησιμοποιεί τα AuthApplicationService, MessagesApplicationService και FriendsApplicationService τα οποία είναι κομμάτια από το business επίπεδο της εφαρμογής για την εκτέλεση των κατάλληλων αλλαγών στα μοντέλα της εφαρμογής. Υποστηρίζει 3 διαφορετικά Events, τα ChatScreenViewEvent, SendMessageEvent, SendPhotoEvent (ο λόγος που ο ChatPresenter είναι υπεύθυνος για την αποστολή μηνυμάτων και φωτογραφιών είναι λόγω της αρχιτεκτονικής του View επιπέδου).

Όταν δέχεται ένα ChatScreenViewEvent, ο Presenter αρχικά χρησιμοποιεί το AuthApplicationService για την ανάκτηση του χρήστη που είναι εξουσιοδοτημένος εκείνη την στιγμή, έπειτα το FriendsApplicationService έτσι ώστε να παρακολουθεί αν καθόλη την διάρκεια που χρησιμοποιείται ο συγκεκριμένος Presenter από το συγκεκριμένο View οι δύο χρήστες είναι συνδεδεμένοι ως φίλοι στην εφαρμογή και εν τέλει την δημιουργία του ChatState. Όταν λαμβάνει SendMessageEvent ή SendPhotoEvent, χρησιμοποιεί το MessagesApplicationService για την αποστολή των μηνυμάτων.

7.4 use-view και views

Για την επίτευξη της επικοινωνίας του View με τον Presenter όπως περιγράφηκε στις προηγούμενες παραγράφους κάθε View χρησιμοποιεί τις δημόσιες μεθόδους του Presenter. Όπως και στην περίπτωση του ViewPresenter για λόγους επαναχρησιμοποίησης και ευκολίας κάθε View δηλαδή κάθε React Component χρησιμοποιεί το useViewComponent το οποίο είναι ένα custom hook. Το useViewComponent παίρνει ως παραμέτρους ένα function που επιστρέφει ένα νέο αντικείμενο του Presenter που θα χρησιμοποιήσει ο Component και μία μεταβλητή τύπου string το οποίο περιλαμβάνει το όνομα του View. Είναι υπεύθυνο για να κάνει την σύνδεση του View με τον Presenter καλώντας την attachView του Presenter κατά την δημιουργία του Component, την παρατήρηση των αλλαγών στην κατάσταση κάνοντας εγγραφή στην μέθοδο states και κατά την καταστροφή, το detach και απεγγραφή από την παρατήρηση των states(). Τέλος, το useViewComponent επιστρέφει την κατάσταση που παρατηρεί και το αντικείμενο του Presenter ώστε ο Component να μεταφράσει την κατάσταση στην διεπαφή και να μπορεί να στέλνει Events στον Presenter.

Η Εικόνα 7.7 δείχνει τον τρόπο που χρησιμοποιείται το useViewComponent hook από το App Component και την δημοσίευση του AppEvent κατά την δημιουργία του Component.



```
export class ChatPresenter extends ViewPresenter<ChatState> {
...
    protected onEvent(event: Event): Observable<ChatState> {
        if (event instanceof ChatScreenViewEvent) {
            this._changedChatSubject.next();

            this._receiver = event.receiver;
            return this._authApplicationService.currentUserId(new CurrentUserIdQuery()).pipe(
                filter( predicate: userId => userId !== undefined),
                take( count: 1),
                mergeMap( project: user => {
                    this._sender = user;

                    return this._friendsApplicationService.watchAreFriends(new WatchAreFriendsQuery(this._sender, this._receiver)).pipe(
                        map( project: friend => {
                            return new ChatState(user, friend);
                        }),
                        takeUntil(this._changedChatSubject)
                    );
                })
            )
        } else if (event instanceof SendMessageEvent) {
            let message = new Message(IdGenerator.generateId(), this._sender, this._receiver, event.message, event.timestamp, event.messageType);
            return this._messagesApplicationService.sendMessage(new SendMessageCommand(message)).pipe(
                ignoreElements()
            );
        } else if (event instanceof SendPhotoEvent) {
            let message = new PhotoMessage(IdGenerator.generateId(), this._sender, this._receiver, event.photo, event.timestamp);
            return this._messagesApplicationService.sendMessage(new SendMessageCommand(message)).pipe(
                ignoreElements()
            );
        }

        return EMPTY;
    }
...
}
```

Εικόνα 7.6 Chat Presenter

```
export default function App() {
    let [documentTitleService] = useState<DocumentTitleService>( initialState: () => injector.get(DocumentTitleService));

    let {
        state,
        presenter
    } = useViewComponent<AppState, AppPresenter>( fn: () => new AppPresenter(injector.get(AuthApplicationService)), VIEW);

    useEffect( effect: () => {
        presenter.publishEvent(new AppEvent());
        documentTitleService.start();

        return () => {
            documentTitleService.stop();
        }
    }, deps: []);


    if (!state) {
        return <></>;
    }

    return (
        <> {
            state.isAuthed ? <DashboardComponent userId={state.userId}/> : <LoginComponent/>
        }
        </>
    );
}
```

Εικόνα 7.7 App Component

7.5 Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκε το πρότυπο Model-View-Intent, ο τρόπος με τον οποίο κυκλοφορούν και αλλάζουν τα δεδομένα στο MVI και πως αυτό υλοποιείται στην εφαρμογή. Η μονόδρομη ροή και τα αμετάβλητα μοντέλα κάνουν την εφαρμογή να είναι συμπαγής και η παρουσία του State για την αναπαράσταση της κατάστασης διεπαφής βοηθούν στην καλύτερη διαχείριση των διάφορων καταστάσεων που μπορεί να προκύψουν κατά την εκτέλεση της εφαρμογής.

Κεφάλαιο 8ο: Dependency Injection

8.1 Εισαγωγή

Το Dependency Injection ή εν συντομία DI είναι ένα αρχιτεκτονικό πρότυπο το οποίο έχει να κάνει με τον διαχωρισμό της δημιουργίας ενός αντικειμένου και της χρήσης του σε ένα πρόγραμμα. Στο DI ένα αντικείμενο δέχεται τις εξαρτήσεις του ως είσοδο (injection) και δεν τις δημιουργεί το ίδιο. Για παράδειγμα μια κλάση Α μπορεί να έχει ένα πεδίο τύπου μιας άλλης κλάσης Β, αυτό το πεδίο είναι μια εξάρτηση για την κλάση Α και για να επιτευχθεί το DI πρέπει αυτή η εξάρτηση να πάρει τιμή (δηλαδή η δημιουργία του αντικειμένου κλάσης Β) από το εξωτερικό της κλάσης, δηλαδή η τιμή να δοθεί στον δομητή της κλάσης ή σε κάποια setter και όχι να δημιουργηθεί μέσα στην ίδια την κλάση Α. Με αυτό τον τρόπο μία κλάση που έχει διάφορες εξαρτήσεις δεν απασχολείται με την δημιουργία τους καθώς τις γίνονται inject. Αυτό έχει επίσης ως αποτέλεσμα, αν για κάποιο λόγο μία από τις εξαρτήσεις της κλάσης Β αποκτήσει δικές της ή αλλάξει τις ήδη υπάρχων εξαρτήσεις της τότε η κλάση Α δεν θα επηρεαστεί καθόλου.

Μία γνωστή πρακτική του DI η οποία χρησιμοποιείται στα περισσότερα σημεία της εφαρμογής, είναι κατά τον σχεδιασμό τον κλάσεων που πρόκειται να είναι εξαρτήσεις άλλων κομματιών κώδικα, οι λειτουργίες αυτών των κλάσεων να ορίζονται από ένα Interface που υλοποιούν αυτές οι κλάσεις. Το Interface αυτό μπορεί να χρησιμοποιηθεί σαν εξάρτηση μίας κλάσης και η κλάση να μην γνωρίζει τον τρόπο που είναι υλοποιημένο, αν επικοινωνεί με κάποια εξωτερική βιβλιοθήκη ή αν αυτό έχει άλλες εξαρτήσεις.

8.2 Πως χρησιμοποιείται στην εφαρμογή

Στην εφαρμογή χρησιμοποιείται η βιβλιοθήκη @mindspace-io που προσφέρει τον μηχανισμό του Dependency Injection για όλες τις typescript εφαρμογές και πιο συγκεκριμένα την @mindspace-io/react που είναι για React. Στο αρχείο di.ts παρέχονται όλες οι κλάσεις που πρόκειται να γίνουν inject σε άλλα μέρη της εφαρμογής στην μέθοδο makeInjector που παρέχει η βιβλιοθήκη για την δημιουργία του injector. Για να γίνει inject ένα αντικείμενο σε ένα άλλο σημείο της εφαρμογής μπορεί να γίνει είτε μέσω του injector που δημιουργείται από το makeInjector function, είτε μέσω της deps ιδιότητας (dependencies) που υπάρχει όταν παρέχετε ένα object στον injector. Παρακάτω φαίνονται δύο παραδείγματα, το πρώτο με το AuthApplicationService που έχει ως εξαρτήσεις ένα αντικείμενο τύπου AuthService και ένα ακόμη τύπου UserRepository που δίνεται στο makeInjector function και το δεύτερο στο AppComponent με τον AppPresenter να έχει εξάρτηση στο AuthApplicationService που με χρήση του injector.get γίνεται inject στον AppPresenter.

1. {
   provide: AuthApplicationService,
   useFactory: authApplicationServiceFactory,
   deps: [FirebaseAuthService, FirebaseUserRepository]
   }
2. new AppPresenter(injector.get(AuthApplicationService))

Να σημειωθεί ότι οι κλάσεις FirebaseAuthService, FirebaseUserRepository υλοποιούν τα AuthService και UserRepository αντίστοιχα και επιτυγχάνεται με την βοήθεια του πολυμορφισμού το Dependency Injection χωρίς προβλήματα.

8.3 Επίλογος

Σε αυτό το κεφάλαιο εν συντομία παρουσιάστηκε τι είναι το Dependency Injection και πως βοηθά κομμάτια κώδικα να μην απασχολούνται με διαδικασίες που δεν θα έπρεπε να γνωρίζουν και τέλος ενδεικτικά παρουσιάστηκε η βιβλιοθήκη που χρησιμοποιήθηκε και πως γίνεται χρήση του Dependency Injection στην εφαρμογή.

Κεφάλαιο 9ο: Domain Driven Design

9.1 Εισαγωγή

Το Domain Driven Design είναι μια προσέγγιση για την σχεδίαση λογισμικού όπου η σχεδίαση επικεντρώνεται στις λειτουργίες που αναφέρεται το λογισμικό. Το DDD (εν συντομία το Domain Driven Design) παρουσιάστηκε πρώτη φορά το 2003 απο τον Eric Evans στο βιβλίο του ”Domain-Driven Design: Tackling Complexity in the Heart of Software”. Παρουσιάζει ανάμεσα σε άλλα πως μπορούν να αναπτυχθούν με την χρήση του μεγάλες εφαρμογές βασισμένες στο Domain και πως αυτή η προσέγγιση βελτιώνει την επικοινωνία των εμπλεκομένων.

Το Domain Driven Design έχει να κάνει περισσότερο με την επιχειρηματική λογική παρότι με τον κώδικα, τις γλώσσες προγραμματισμού και τις βιβλιοθήκες που θα χρησιμοποιεί η εφαρμογή. Σκοπεύει στην δημιουργία προϊόντων με τρόπο τέτοιο ώστε να είναι αντιληπτά από άτομα που δεν είναι στον τομέα του προγραμματισμού, όπως οι ιδιοκτήτες εταιρειών που δεν είναι απαραίτητο ότι θα έχουν άμεση σχέση με το αντικείμενο της εφαρμογής και τον τομέα του προγραμματισμού.

Δημιουργώντας μια τέτοια εφαρμογή αυξάνεται η αποτελεσματικότητα των εμπλεκομένων και γίνεται η επικοινωνία καλύτερη καθώς διάφοροι εμπλεκόμενοι είναι σε θέση να καταλάβουν τι αφορά και τι πρέπει να κάνει αυτή η εφαρμογή χωρίς να χρειάζονται εξειδικευμένες γνώσεις και μπορούν να επικοινωνήσουν μεταξύ τους πιο εύκολα. Για την επίτευξη αυτού υπάρχει ανάγκη μιας κοινής γλώσσας - ορολογίας την οποία πρέπει να καταλαβαίνουν όλοι οι εμπλεκόμενοι, έχει να κάνει με τις λειτουργίες της εφαρμογής και χρησιμοποιείται καθ’ όλη την διάρκεια τόσο της σχεδίασης όσο και της ανάπτυξης της εφαρμογής.

Η διαδικασία της σχεδίασης της εφαρμογής είναι ιδιαίτερα δύσκολη καθώς χρειάζονται άτομα με εμπειρία και εξειδίκευση ώστε να μπορούν να αναγνωρίσουν και να ορίσουν τα σημαντικά σημεία της εφαρμογής και ποιες λειτουργίες πρέπει να κάνει ώστε να είναι κατανοητά σε όλους τους εμπλεκόμενους και να περιγράφουν σωστά τις ανάγκες της εφαρμογής.

9.2 Τι είναι το Domain

Το Domain για μία εφαρμογή είναι το κομμάτι το οποίο χαρακτηρίζει την εφαρμογή και που παρουσιάζει με τι έχει να κάνει μία εφαρμογή. Για παράδειγμα το Domain στην συγκεκριμένη εφαρμογή είναι η ανταλλαγή μηνυμάτων, οι κλήσεις και οι τυχαίες συνομιλίες δηλαδή όλα όσα περιγράφουν την εφαρμογή αυτή.

9.3 Τι είναι η επιχειρηματική λογική (Business Logic)

Η επιχειρηματική λογική μιας εφαρμογής είναι οι λειτουργίες που πρέπει να έχει ένα προϊόν και τους κανόνες που πρέπει να εφαρμόζει. Το Business Logic στο DDD ονομάζεται Domain Logic και κάθε Domain έχει τους δικούς του επιχειρηματικούς κανόνες. Για παράδειγμα στην συγκεκριμένη εφαρμογή, μέρος του Domain Logic είναι η λειτουργία της εφαρμογής να στέλνει μηνύματα και να κάνει κλήσεις. Το Business/Domain Logic πρέπει να παραμένει άθικτο και να μην επηρεάζεται από τεχνολογίες που μπορεί να αλλάζουν συνεχώς κατά την υλοποίηση του.

9.4 Τι είναι Aggregates & Aggregate Root

Τα Aggregates είναι συλλογές από αντικείμενα που έχουν να κάνουν με το business επίπεδο και πρέπει από τις διάφορες αλλαγές που δέχονται να παραμένουν ακέραια και να μην παραβιάζουν τους επιχειρηματικούς κανόνες. Επειδή για να επιτευχθεί η ακεραιότητα πολλών διαφορετικών business objects μετά από μία αλλαγή είναι δύσκολο καθώς το καθένα μεταφέρει σημαντική πληροφορία και μπορεί να αλληλεξαρτώνται, χρησιμοποιούνται τα Aggregate Roots. Τα Aggregate Roots λειτουργούν

ως σημεία εισόδου για τις αλλαγές στα Aggregates και όλες οι αλλαγές πρέπει να γίνονται μέσω αυτών. Φροντίζουν ώστε να εφαρμόζονται οι επιχειρηματικοί κανόνες και τα objects να είναι συνεπείς στις αλλαγές που δέχονται. Τα Aggregate Roots της εφαρμογής είναι τα User, UserInfo, Random, Messages, Friend, Call.

9.5 Τι είναι τα Commands

Τα Commands χρησιμοποιούνται για να αλλάξουν την κατάσταση της εφαρμογής και συγκεκριμένα για κάνουν μία αλλαγή στο μοντέλο της εφαρμογής. Όταν πρέπει να γίνει μία αλλαγή δημιουργείται ένα αντίστοιχο Command που μεταφέρει πληροφορία η οποία σηματοδοτεί την αλλαγή που πρέπει να γίνει. Παράδειγμα ενός Command είναι το EnqueueCommand το οποίο χρησιμοποιείται για να μπει ο χρήστης στην ουρά για αναζήτηση ατόμου. Τα Commands δίνονται από το Presentation επίπεδο στα αντίστοιχα Application Services τα οποία γνωρίζουν πως να τα διαχειριστούν.

9.6 Τι είναι τα Queries

Τα Queries χρησιμοποιούνται για την άντληση δεδομένων που αφορούν τα μοντέλα. Όπως και τα Commands, τα Queries τα διαχειρίζονται τα Application Services τα οποία γνωρίζουν πως να επικοινωνήσουν για να επιστρέψουν τα δεδομένα. Ένα παράδειγμα Query είναι το WatchRandomQuery μέσω του οποίου ο χρήστης μαθαίνει αν βρέθηκε τυχαίος χρήστης για συνομιλία.

9.7 Τι είναι τα Application Services

Τα Application Services υπάρχουν για την διαχείριση και οργάνωση των Domain Objects. Η ουσιαστική δουλειά του Application Service είναι να δέχεται εντολές/Commands και ερωτήματα/Queries και να τα χρησιμοποιήσει ώστε να κάνει κάποια λειτουργία όπως είναι μία αλλαγή (Command) στα Domain Object ή να επιστρέψει δεδομένα (Query). Παρόλο που βρίσκεται στο κομμάτι του Domain δεν γνωρίζει για επιχειρηματικούς κανόνες καθώς αυτούς τους εφαρμόζει το Aggregate Root. Τέλος είναι το μοναδικό σημείο μέσω του οποίου μπορούν να επικοινωνήσουν άλλα επίπεδα της εφαρμογής -για παράδειγμα το Presentation- με το Domain επίπεδο. Τα Application Services της εφαρμογής είναι τα AuthApplicationService, MessagesApplicationService, UserInfoApplicationService, RandomApplicationService, FriendsApplicationService και CallApplicationService.

9.8 Τι είναι τα Microservices

Τα Microservices/μικρό-υπηρεσίες είναι αυτόνομα κομμάτια λογισμικού που αναλαμβάνουν μία λειτουργία της εφαρμογής. Σε σύγκριση με την μονολιθική αρχιτεκτονική που ολόκληρη η εφαρμογή χτίζεται εξαρχής και αναπτύσσεται ως μία υπηρεσία, τα Microservices χωρίζουν κάθε λειτουργία της εφαρμογής. Αυτό φέρνει αρκετά πλεονεκτήματα στην Microservices αρχιτεκτονική σε σύγκριση με την Μονολιθική.

Όπως όταν η εφαρμογή θα χρειάζεται να γίνει scale οριζόντια δηλαδή να αυξηθούν οι κόμβοι που τρέχουν την εφαρμογή ώστε να μπορεί για παράδειγμα να εξυπηρετήσει περισσότερους χρήστες ταυτόχρονα στο κομμάτι των μηνυμάτων, τότε στην μονολιθική θα πρέπει να ξαναγίνει deploy ολόκληρη η εφαρμογή. Από την άλλη μεριά στην Microservices αρχιτεκτονική θα χρειαστεί να γίνει ξανά deploy μόνο το microservice το οποίο έχει να κάνει με την ανταλλαγή μηνυμάτων.

Επίσης κατά την μονολιθική αρχιτεκτονική η εφαρμογή πρέπει να αναπτυχθεί εξ ολοκλήρου με την ίδια γλώσσα προγραμματισμού και συνήθως θα υπάρχει και μία κοινή βάση δεδομένων. Σε σύγκριση με τις μικρό-υπηρεσίες όπου η κάθε μία μπορεί να είναι γραμμένη σε ξεχωριστή γλώσσα προγραμματισμού και με διαφορετικές τεχνολογίες όπως επίσης και με διαφορετική βάση δεδομένων όπου ένα Microservice μπορεί να έχει σχεσιακή βάση δεδομένων και ένα άλλο αν έχει NoSQL βάση δεδομένων. Αυτά έχουν σαν αποτέλεσμα την εύκολη εναλλαγή γλώσσας ή βάσης δεδομένων σε σύγκριση με τον μονολιθικό τρόπο καθώς σε αυτόν θα έπρεπε να ξαναγραφτεί ολόκληρη η εφαρμογή από την αρχή αν

έπρεπε να αλλαχτεί η γλώσσα. Άλλο ένα παράδειγμα είναι όταν πρέπει να προστεθεί κάποια επιπλέον λειτουργία τότε στην μία περίπτωση απλά θα προστεθεί ένα καινούριο Microservice το οποίο θα λειτουργεί αυτόνομα ενώ στην άλλη περίπτωση θα πρέπει να προστεθούν κομμάτια κώδικα που πιθανόν να επιφέρουν προβλήματα στις υπόλοιπες λειτουργίες της εφαρμογής. Τέλος ένα ακόμη βασικό μειονέκτημα της μονολιθικής αρχιτεκτονικής είναι ότι συνήθως επειδή η εφαρμογή λειτουργεί ως μία υπηρεσία τότε αν προκύψει κάποιο πρόβλημα σε κάποια λειτουργία της εφαρμογής ή για κάποιο λόγο η βάση δεδομένων δεν είναι διαθέσιμη τότε ολόκληρη η υπηρεσία θα καταρρεύσει και δεν θα είναι διαθέσιμη μέχρι να λυθούν τα προβλήματα, όμως στην αρχιτεκτονική μικρό-υπηρεσιών αν καταρρεύσει μία υπηρεσία δεν θα επηρεάσει τις υπόλοιπες υπηρεσίες του συστήματος.

Να σημειωθεί ότι Microservices χρησιμοποιούνται και αναφέρονται συνήθως στο back-end και για να ανταλλάξουν δεδομένα χρησιμοποιούν κάποιο δικτυακό πρωτόκολλο επικοινωνίας.

Η παραπάνω αρχιτεκτονική ενσωματώθηκε στην ανάπτυξη της εφαρμογής και παρόλο που είναι front-end η Microservice αρχιτεκτονική φάνηκε αρκετά χρήσιμη ώστε να απομονωθούν διάφορες λειτουργίες της εφαρμογής και σε συνδυασμό με την domain-driven προσέγγιση που αναπτύχθηκε το καθένα, τα Microservices έγιναν πιο συμπαγής και η εφαρμογή επιτυγχάνει τους επιχειρηματικούς κανόνες της. Επίσης σε αυτή την περίπτωση η επικοινωνία των υπηρεσιών επιτυγχάνεται με εσωτερικά services όπως γίνεται σε αυτή την εφαρμογή ή με κάποιο άλλο πρότυπο όπως είναι το event-sourcing.

9.8.1 Ποια είναι τα Microservices της εφαρμογής

Τα Microservices της εφαρμογής είναι τα auth, calling, friends, messages, random, user-info και θα αναλυθούν παρακάτω.

9.8.1.1 Auth Microservice

Το Auth Microservice αναλαμβάνει τις λειτουργίες που αφορούν την αυθεντικοποίηση του χρήστη στην εφαρμογή. Αναλαμβάνει την αρχική εγγραφή και σύνδεση του χρήστη στην εφαρμογή, επίσης αναλαμβάνει την αποσύνδεση του χρήστη από την εφαρμογή και τέλος προσφέρει σε όποιο σημείο το χρειαστεί ενημέρωση για την κατάσταση του client δηλαδή ποιος χρήστης είναι συνδεδεμένος και αν είναι συνδεδεμένος. Το auth συνδυάζεται - χρησιμοποιεί το User Aggregate ώστε να αποθηκεύσει και να ενημερώσει το Aggregate για τον χρήστη. Το Auth Microservice είναι πολύ σημαντικό για την λειτουργία της εφαρμογής γιατί χωρίς τις λειτουργίες που προσφέρει δεν θα μπορούσε ο χρήστης να συνδεθεί και να χρησιμοποιήσει την εφαρμογή.

9.8.1.2 User-info Microservice

Το User-info Microservice αναλαμβάνει λειτουργίες που έχουν να κάνουν με την άντληση δεδομένων που η εφαρμογή παρουσιάζει στον χρήστη. Το SideBar της εφαρμογής παρουσιάζει πληροφορίες όπως ο χρήστης που είναι συνδεδεμένος, οι φίλοι του χρήστη, τα τελευταία μηνύματα ανά φίλο και την ενεργή κατάσταση αυτών. Έτσι η μικρό-υπηρεσία έχει ως βασική λειτουργία την άντληση αυτών των πληροφοριών για την χρήση τους από άλλα επίπεδα. Το user-info είναι ένα παράδειγμα που προσφέρει συγκεκριμένες πληροφορίες όπως είναι οι αυτές του SideBar που μέσω αυτού ο χρήστης μπορεί να μεταφερθεί και να ενημερωθεί. Γίνεται ξεκάθαρο ότι το user-info έχει αναπτυχθεί για να καλύψει μία λειτουργία που προκύπτει στην εφαρμογή και αν για κάποιο λόγο η εφαρμογή δεν χρειάζεται πλέον το SideBar τότε το Auth Microservice μπορεί να αφαιρεθεί ή να αντικατασταθεί εύκολα.

9.8.1.3 Messages Microservice

Το Messages Microservice έχει να κάνει με την ανταλλαγή μηνυμάτων ανάμεσα σε δύο χρήστες, προσφέρει δυνατότητα άντλησης των μηνυμάτων για μία συγκεκριμένη συνομιλία και επίσης αποστολή μηνυμάτων σε από έναν χρήστη σε έναν άλλο. Το Messages είναι από τα πιο βασικά κομμάτια της εφαρμογής καθώς χρησιμοποιείται για την ίσως πιο βασική λειτουργία της εφαρμογής που είναι η

επικοινωνία μέσω μηνυμάτων και πέρα από τα γραπτά μηνύματα υποστηρίζει και την αποστολή εικόνων.

9.8.1.4 Random Microservice

Μία λειτουργία της εφαρμογής είναι η επιλογή για συνομιλία με τυχαία άτομα τα οποία δεν είναι ήδη στην λίστα φίλων ενός χρήστη και μέσω αυτής της συνομιλίας αν συμφωνούν και οι δύο να γίνουν φίλοι. Για την υποστήριξη αυτής της ανάγκης υπάρχει το Random Microservice το οποίο εκτελεί λειτουργίες όπως ο χρήστης να προστεθεί ή να αφαιρεθεί από την ουρά αναζήτησης τυχαίου ατόμου. Επίσης υποστηρίζει την αποστολή, ακύρωση και απάντηση αιτήματος φιλίας και την αποχώρηση από την συνομιλία σε περίπτωση που δεν επιθυμεί να επικοινωνήσει άλλο με αυτό το άτομο. Τέλος υποστηρίζει άντληση πληροφοριών για την λίστα, δηλαδή αν βρίσκεται στην λίστα και πληροφορίες για το Random δηλαδή αν βρέθηκε τυχαίος να μιλήσει.

9.8.1.5 Friends Microservice

Το Friends Microservice υλοποιεί λειτουργίες που έχουν να κάνουν με τους φίλους των χρηστών, λειτουργίες όπως η διαγραφή φίλων και η άντληση των φίλων ώστε να ξέρει ο χρήστης ποιοι βρίσκονται στην λίστα φίλων του. Το Friends χρησιμοποιείται επίσης και για πληροφορίες που έχουν να κάνουν με την κατάσταση του φίλου δηλαδή πότε συνδέθηκε τελευταία φορά ή αν είναι ακόμα φίλοι με τον συνδεδεμένο χρήστη.

9.8.1.6 Calling Microservice

Οι κλήσεις στην εφαρμογή είναι μία βασική λειτουργία που μπορούν να εκτελέσουν δύο φίλοι και ίσως η δυσκολότερη λειτουργία της εφαρμογής καθώς χρειάζεται διάφορες εσωτερικές αρχιτεκτονικές και τεχνολογίες για να υλοποιηθεί συγκριτικά με τα υπόλοιπα Microservices. Έτσι το Calling Microservice είναι υπεύθυνο για την υλοποίηση κλήσεων και βίντεο-κλήσεων στην εφαρμογή, όπως επίσης είναι υπεύθυνο για την παροχή λειτουργιών όπως είναι η πληροφορίες της συγκεκριμένης κλήσης, αν ο χρήστης βρίσκεται σε κλήση και τα χαρακτηριστικά των κλήσεων. Χωρίς το Calling Microservice η εφαρμογή θα περιοριζόταν μόνο στην γραπτή επικοινωνία, τέλος είναι πολύ σημαντικό καθώς παρουσιάζει διαφορετικές λειτουργίες που δεν έχουν τα υπόλοιπα microservices και αντιπροσωπεύει ακριβώς τον στόχο της Microservices αρχιτεκτονικής.

9.8.2 Πως χρησιμοποιείται το DDD στα Microservices της εφαρμογής

9.8.2.1 Auth

Το Auth Microservice χρησιμοποιείται συνδυαστικά με το User Aggregate. Το User Aggregate Root περιέχει το Aggregate Id, το πεδίο email, το πεδίο displayName που αναπαριστά το όνομα του χρήστη και το πεδίο photoUrl για την διεύθυνση της φωτογραφίας του χρήστη. Υπάρχει και το UserRepository το οποίο βοηθά στην εγγραφή και την άντληση πληροφοριών του User.

Με το AuthApplicationService η εφαρμογή μπορεί να ενημερωθεί για την κατάσταση του User, δηλαδή αν είναι συνδεδεμένος ή όχι με το AuthWatchQuery, επίσης μπορεί να ενημερωθεί για την ταυτότητα (id) του συνδεδεμένου χρήστη με το CurrentUserIdQuery και τέλος να κάνει αποσυνδεθεί από την εφαρμογή με το LogoutCommand. Μία επιπλέον λειτουργία του ApplicationService είναι ότι χρησιμοποιεί το AuthWatchQuery ώστε όταν ο χρήστης συνδεθεί στην εφαρμογή να γίνει εγγραφή του στην βάση σε περίπτωση που είναι πρώτη φορά που συνδέεται.

Τέλος ένα σημαντικό στοιχείο που υπάρχει μέσα στο Auth Microservice είναι το AuthService, όπου χρησιμοποιείται από το AuthApplicationService για να κάνει όλες του τις ενέργειες μέσω του οποίου γίνεται και η σύνδεση του χρήστη στην εφαρμογή.

9.8.2.2 Messages

Στο Messages Microservice υπάρχει το Message το οποίο είναι το Aggregate Root πάνω στο οποίο γίνονται οι αλλαγές. Το Message περιέχει τα πεδία που δείχνουν τον αποστολέα του μηνύματος - πεδίο from, τον παραλήπτη - πεδίο to, το μήνυμα - πεδίο message, την ώρα της αποστολής - πεδίο timestamp, τον τύπο του μηνύματος δηλαδή αν είναι μήνυμα κειμένου, φωτογραφίας ή πληροφορίας - πεδίο type τύπου MessageType και το πεδίο id που είναι το αναγνωριστικό του. Επίσης έχει το MessagesRepository interface που υλοποιείται για να γίνουν οι λειτουργίες αποστολής και άντλησης των μηνυμάτων.

Για την διαχείριση των αλλαγών χρησιμοποιείται το MessagesApplicationService που δέχεται το GetMessagesQuery για την άντληση των μηνυμάτων ανάμεσα σε δύο χρήστες και το SendMessageCommand για την αποστολή μηνύματος από έναν χρήστη σε έναν άλλο. Τέλος υπάρχει η κλάση PhotoMessage για την αποστολή φωτογραφιών που είναι και αυτό ένα Aggregate Root επεκτείνοντας την κλάση Message.

9.8.2.3 Friends

Το Friends Microservice έχει το Friend Aggregate Root το οποίο έχει πληροφορίες όπως το id του Friend που είναι το αναγνωριστικό του και δείχνει την μοναδικότητα του, το πεδίο displayName που περιέχει το όνομα του χρήστη, την τελευταία φορά που ήταν ενεργός - πεδίο lastOnline και το πεδίο photoUrl που δείχνει την διεύθυνση URL της φωτογραφίας προφίλ του χρήστη. Επιπλέον υπάρχει το FriendsRepository interface το οποίο βοηθά στην ανάγνωση και στην διαγραφή κάποιου φίλου και επεκτείνει το InMemoryRepository ώστε να αποθηκεύονται οι φίλοι στην μνήμη και να μην γίνεται κάθε φορά κλήση στην βάση για να αντληθούν οι φίλοι.

Υπάρχει το FriendsApplicationService το οποίο δέχεται το RemoveFriendCommand και χρησιμοποιώντας το FriendsRepository θα διαγράψει τον φίλο από την λίστα φίλων του χρήστη και το query GetFriendQuery μέσω του οποίου ο χρήστης θα πάρει πληροφορίες για έναν φίλο δηλαδή θα λάβει το Friend Aggregate Root για τον συγκεκριμένο φίλο. Τέλος δέχεται το WatchAreFriendsQuery μέσω του οποίου και του FriendsRepository θα επιστραφεί στον χρήστη ο συγκεκριμένος Friend αν όντως υπάρχει και αν προστεθεί ως φίλος στο μέλλον τότε ο χρήστης που παρατηρεί αυτό το Query θα ενημερωθεί.

9.8.2.4 Random

Το Random Microservice έχει το Random Aggregate Root το οποίο περιέχει το randomId που είναι το id του τυχαίου χρήστη που συνομιλεί και ταυτόχρονα το Aggregate Id που δείχνει την μοναδικότητα του. Περιέχει το friendRequest πεδίο τύπου FriendRequest και δείχνει πότε υπάρχει αίτημα φιλίας και απο ποιον χρήστη. Επίσης το RandomRepository interface παρέχει μεθόδους που αφορούν τα αιτήματα φιλίας, την λίστα αναμονής και την τυχαία συνομιλία.

Το RandomApplicationService δέχεται τα WatchRandomQuery και WatchQueueQuery τα οποία επιστρέφουν πληροφορίες για την τυχαία συνομιλία και για την λίστα αντίστοιχα. Δέχεται τα EnqueueCommand και DequeueCommand που έχουν να κάνουν με την εγγραφή και απεγγραφή από την λίστα αναμονής. Επιπρόσθετα δέχεται τα SendFriendRequestCommand, CancelFriendRequestCommand και RespondFriendRequestCommand που η λειτουργίας τους είναι να δημιουργήσουν, να ακυρώσουν και απαντήσουν σε αιτήματα φιλίας. Τέλος υπάρχει το NextCommand που σηματοδοτεί την αποχώρηση του χρήστη από την τυχαία συνομιλία.

9.8.2.5 UserInfo

Στο UserInfo Microservice έχει αναπτυχθεί το UserInfo Aggregate Root το οποίο έχει το Aggregate Id, το όνομα και το επίθετο του χρήστη (πεδία firstName και lastName), το πεδίο username, την διεύθυνση της φωτογραφίας προφίλ (πεδίο photo) και μία λίστα αντικειμένων FriendInfo που περιέχουν

πληροφορίες για τον φίλο με σημαντικότερες το id του, το όνομα του, το τελευταίο μήνυμα και η ώρα του μηνύματος αυτού.

Το UserInfoApplicationService δέχεται τα UserInfoWatchQuery, GetUserInfoFriendQuery και σε συνδυασμό με το UserInfoRepository προσφέρει πληροφορίες για το UserInfo Aggregate όπως και για τους φίλους που υπάρχουν στην λίστα του UserInfo. Μόλις ο χρήστης συνδεθεί στην εφαρμογή τότε η εφαρμογή χρησιμοποιεί το UserInfo Aggregate για να αντλήσει τις πληροφορίες που χρειάζεται για το SideBar της εφαρμογής.

9.8.2.6 Calling

Το Calling Microservice χρησιμοποιείται για την επίτευξη κλήσεων ανάμεσα σε δύο χρήστες. Το Call Aggregate Root περιέχει αρκετές σημαντικές πληροφορίες πέρα από το Aggregate Id που έχουν να κάνουν με μία κλήση όπως είναι το id του caller, η ώρα της προσφοράς κλήσης - πεδίο offerTime, αν η κλήση απαντήθηκε και η ώρα της απάντησης - πεδία answered και answeredTime. Επίσης περιέχει δύο αντικείμενα που έχουν πληροφορίες για την κατάσταση του μικροφώνου και της κάμερας κάθε χρήστη - πεδία caller και callee τύπου { mic: boolean; video: boolean; } και τέλος το πεδίο info τύπου CallFinishInfo που περιέχει πληροφορίες για το τέλος της κλήσης (αν έχει τελειώσει) όπως ο τρόπος που τερματίστηκε η κλήση και ποιος την τερμάτισε.

Το CallApplicationService αναλαμβάνει αρκετές διαδικασίες που έχουν να κάνουν με το Call Aggregate. Αρχικά με τα StartCallCommand, AnswerCallCommand και FinishCallCommand το CallApplicationService σηματοδοτεί την αρχή, την απάντηση και το τέλος μιας κλήσης. Επιπλέον υπάρχουν τα ToggleMicCommand και ToggleVideoCommand μέσω τον οποίον γίνονται αλλαγές στις καταστάσεις του μικροφώνου και της κάμερας, δηλαδή αν οι συσκευές είναι ενεργές ή όχι. Τα σημαντικότερα Queries που διαχειρίζεται είναι το WatchForCallsQuery μέσω του οποίου ενημερώνεται ο συνδεδεμένος χρήστης για εισερχόμενες κλήσεις και το WatchCurrentCallQuery το οποίο χρησιμοποιείται για πληροφορίες της κλήσης την στιγμή που ο χρήστης είναι σε κλήση.

9.9 Επίλογος

Με το Domain Driven Design (DDD) γίνεται εύκολη η μετάφραση των επιχειρηματικών εννοιών σε κώδικα και έτσι υπάρχει καλύτερη επικοινωνία ανάμεσα στους εμπλεκόμενους μιας εφαρμογής. Με την περιγραφή του DDD παραπάνω έγινε κατανοητό γιατί είναι σημαντική η χρήση του και τα προβλήματα που λύνει με τα αρχικά πολύπλοκα αλλά εν τέλει ισχυρά εργαλεία που προσφέρει. Τα Microservices λύνουν αρκετά προβλήματα που προκύπτουν από την αρχιτεκτονική και σε συνδυασμό με το DDD το αποτέλεσμα είναι ποιοτικό και συντηρήσιμο και αυτό φαίνεται και στα Microservices που μόλις περιγράφηκαν οι λειτουργίες τους.

Κεφάλαιο 10ο: Διεπαφές χρήστη (UI)

10.1 Εισαγωγή

Το User Interface (UI) της εφαρμογής φτιάχτηκε με απλά στοιχεία ώστε ο χρήστης να έχει μία καλή εμπειρία και να μην νιώθει άβολα χρησιμοποιώντας το. Για αυτό το βασικό UI απαρτίζεται από ένα AppBar, ένα SideBar και έναν container που εκεί παρουσιάζεται το κύριο περιεχόμενο της εφαρμογής. Η εφαρμογή είναι web οπότε στο UI εφαρμόζονται Styles με την γλώσσα CSS (Cascading Style Sheets) και πιο συγκεκριμένα με την βιβλιοθήκη Sass(Syntactically Awesome Style Sheets) η οποία είναι μία επέκταση του CSS. Επίσης χρησιμοποιούνται αρκετά Material Components και Icons ώστε να δώσουν στην εφαρμογή μία επιπλέον αισθητική που σε συνδυασμό με τα Styles δίνουν στον χρήστη μία καλαίσθητη εφαρμογή τόσο εμφανισιακά όσο και ως εμπειρία χρήσης.

10.2 Sass

Το Sass είναι μία γλώσσα για styles που μεταφράζεται σε CSS και είναι ένα υπέρ-σύνολο του CSS (όπως είναι και η TypeScript για την JavaScript), αυτό σημαίνει ότι όλο το συντακτικό του CSS υποστηρίζεται από το Sass. Προσφέρει διάφορες λειτουργίες όπως μεταβλητές (διαφορετικές από τις CSS μεταβλητές), mixins, εμφωλεύσεις και functions.

10.2.1 Συντακτικό

Υποστηρίζει δύο διαφορετικά συντακτικά, το scss που χρησιμοποιείται στην εφαρμογή και είναι πιο κοντά συντακτικά στο CSS και το sass που βασίζεται στις εσοχές. Τα αρχεία που περιέχουν scss συντακτικό έχουν κατάληξη .scss ενώ αυτά που περιέχουν sass συντακτικό έχουν κατάληξη .sass.

10.2.2 Μεταβλητές

Οι μεταβλητές Sass ορίζονται με ονόματα που αρχίζουν με το σύμβολο του δολαρίου ($), για παράδειγμα η μεταβλητή με όνομα background και τιμή #ffffff ορίζεται: $background: #ffffff. Έπειτα η μεταβλητή αυτή μπορεί να χρησιμοποιηθεί αντί για την ίδια την τιμή, για παράδειγμα το border: 1px solid $background; είναι ισοδύναμο με το 1px solid #ffffff. Επίσης υποστηρίζει και μεταβλητές map που περιέχει τιμές τύπου key - value όπως είναι το $colors map Εικόνα 10.1 που χρησιμοποιείται στο App.scss της εφαρμογής.

```scss
$primary-color: #ea5247;
$drawer-width: 280;
$secondary-color: #1976d2;
$colors: (
        'primary': $primary-color,
        'lighter-primary': lighten($primary-color, 30%),
        'secondary': $secondary-color,
        'lighter-secondary': lighten($secondary-color, 50%),
        'hover': lighten($primary-color, 35%),
);
```

Εικόνα 10.1 Colors Map

10.2.3 Εμφωλευμένοι κανόνες (nested rules)

Οι εμφωλευμένοι κανόνες βοηθούν ώστε να μην επαναλαμβάνονται τα ίδια ονόματα κλάσεων και στοιχειών ώστε να στοχευθούν συγκεκριμένα στοιχεία. Στην Εικόνα 10.2 φαίνεται η διαφορά και η ευκολία που προσφέρει το Scss (αριστερά) τόσο στην γραφή όσο και στην κατανόηση του κώδικα σε σύγκριση με το CSS (δεξιά).

```scss
div {
  h1 {
    padding: 0;
    background: red;
  }

  video {
    background: white;
  }

  a {
    font-size: 14px;
  }
}
```

```css
div h1 {
  padding: 0;
  background: red;
}

div video {
  background: white;
}

div a {
  font-size: 14px;
}
```

Εικόνα 10.2 Scss vs CSS

10.2.4 Mixins

Τα mixins χρησιμοποιούνται για να δημιουργήσουν επαναχρησιμοποιούμενα styles. Ορίζονται με την λέξη-κλειδί @mixin και έπειτα ένα όνομα. Επίσης υποστηρίζονται οι παράμετροι βάζοντας το όνομα από κάθε παράμετρο χωρισμένες με κόμμα, μέσα σε παρενθέσεις (). Για παράδειγμα παρακάτω στην Εικόνα 10.3 το mixin dashboard-component δέχεται δύο παραμέτρους το $primary-color και το $drawer-width και ορίζει κάποιους κανόνες. Τα mixins χρησιμοποιούνται σε άλλα αρχεία με την λέξη κλειδί @include, έτσι το @include dashboard-component($primary-color, $drawer-width); συμπεριλαμβάνει το dashboard-component mixin στο αρχείο που το χρησιμοποιεί.

```scss
@mixin dashboard-component($primary-color, $drawer-width) {
  .content-placeholder {
    display: flex;
    width: calc(100% - #{$drawer-width}px);
    margin-left: #{$drawer-width}px;

    @media (max-width: 600px) {
      width: 100%;
      margin-left: 0;
    }
  }
}
```

Εικόνα 10.3 Dashboard mixin

10.2.5 Functions

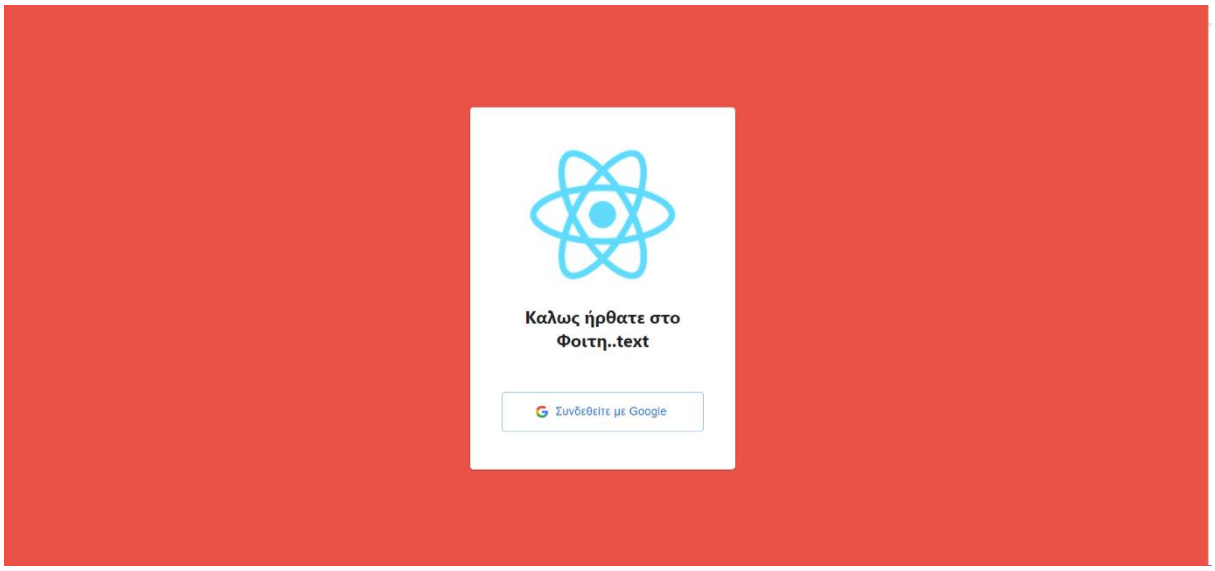
Το Sass προσφέρει έτοιμα functions που εκτελούν διάφορες λειτουργίες. Μερικά από αυτά που χρησιμοποιούνται στην εφαρμογή είναι το map function που βοηθά στην πρόσβαση σε κάποιο στοιχείο μιας μεταβλητής και το lighten function που χρησιμοποιείται για να αλλάξει την φωτεινότητα μιας μεταβλητής.

10.3 Material

Αρχικά το Material είναι μία γενική κατευθυντήρια γραμμή της Google με συμβουλές για design, εργαλεία και components για την δημιουργία διεπαφών χρήστη. Στην εφαρμογή χρησιμοποιούνται οι components που προσφέρει και τα Material Icons για τις εικόνες της εφαρμογής. Το Material έχει υποστήριξη για όλες τις πλατφόρμες, web - Android - iOS και για την καλύτερη προσβασιμότητα πολλές βιβλιοθήκες έχουν αναλάβει να υλοποιήσουν τους αντίστοιχους Components, έτσι το MUI (από το Material UI) έχει αναλάβει την υλοποίηση του Material για React και είναι η βιβλιοθήκη μέσω τις οποίας χρησιμοποιούνται στην εφαρμογή τα Material εργαλεία.

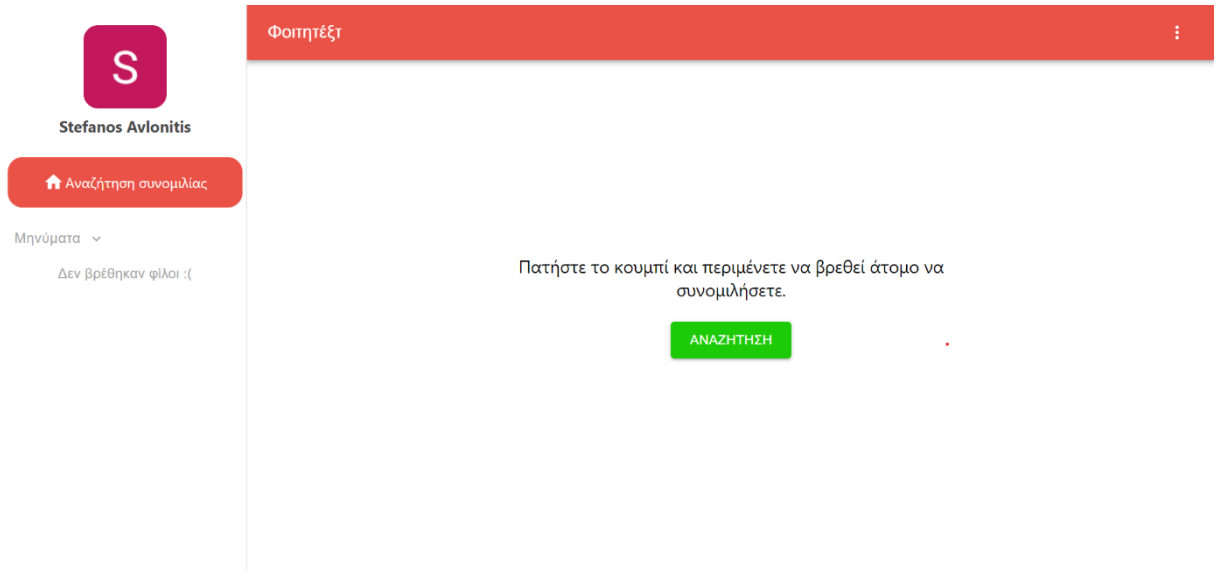
10.4 Εμφάνιση της εφαρμογής

Στην Εικόνα 10.4 φαίνεται πως είναι η σελίδα σύνδεσης της εφαρμογής. Όταν ο χρήστης ανοίγει την εφαρμογή εμφανίζεται η παρακάτω οθόνη και πατώντας το κουμπί “Συνδεθείτε με Google”, ο χρήστης μπορεί να κάνει εγγραφή και να συνδεθεί με έναν λογαριασμό Google στην εφαρμογή.



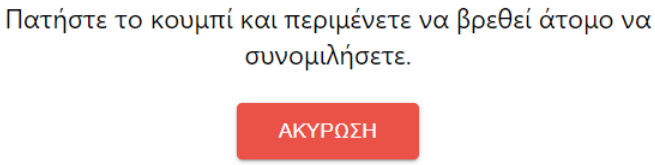
Εικόνα 10.4 Login Screen

Αφού γίνει η σύνδεση στην εφαρμογή μέσω Google ο χρήστης μεταφέρετε στην αρχική σελίδα της εφαρμογής η οποία φαίνεται στην Εικόνα 10.5. Πάνω φαίνεται το βασικό AppBar της εφαρμογής το οποίο έχει το όνομα της εφαρμογής το οποίο είναι «Φοιτητέξτ» και ένα μενού στο δεξί μέρος το οποίο αν επιλεχθεί ο χρήστης μπορεί να κάνει αποσύνδεση από την εφαρμογή. Αριστερά υπάρχει το SideBar στο οποίο αρχικά υπάρχει η εικόνα προφίλ του χρήστη και το όνομα του που αντλήθηκαν και τα δύο από τον Google λογαριασμό του. Έπειτα υπάρχει ένα κουμπί το οποίο σε μεταφέρει στην αρχική σελίδα (στην σελίδα δηλαδή που ήδη είναι ο χρήστης) και πιο κάτω υπάρχει η λίστα των μηνυμάτων μαζί με ένα μήνυμα που ενημερώνει τον χρήστη ότι δεν βρέθηκαν φίλοι άρα δεν έχει και μηνύματα. Στο κέντρο της εικόνας υπάρχει ο βασικός container της εφαρμογής που θα χρησιμοποιηθεί για να παρουσιάσει την βασική πληροφορία ανά οθόνη - προορισμό εντός της εφαρμογής. Έτσι η αρχική σελίδα περιέχει ένα μήνυμα και ένα κουμπί με το οποίο ο χρήστης μπορεί να αναζητήσει άτομο για συνομιλία.



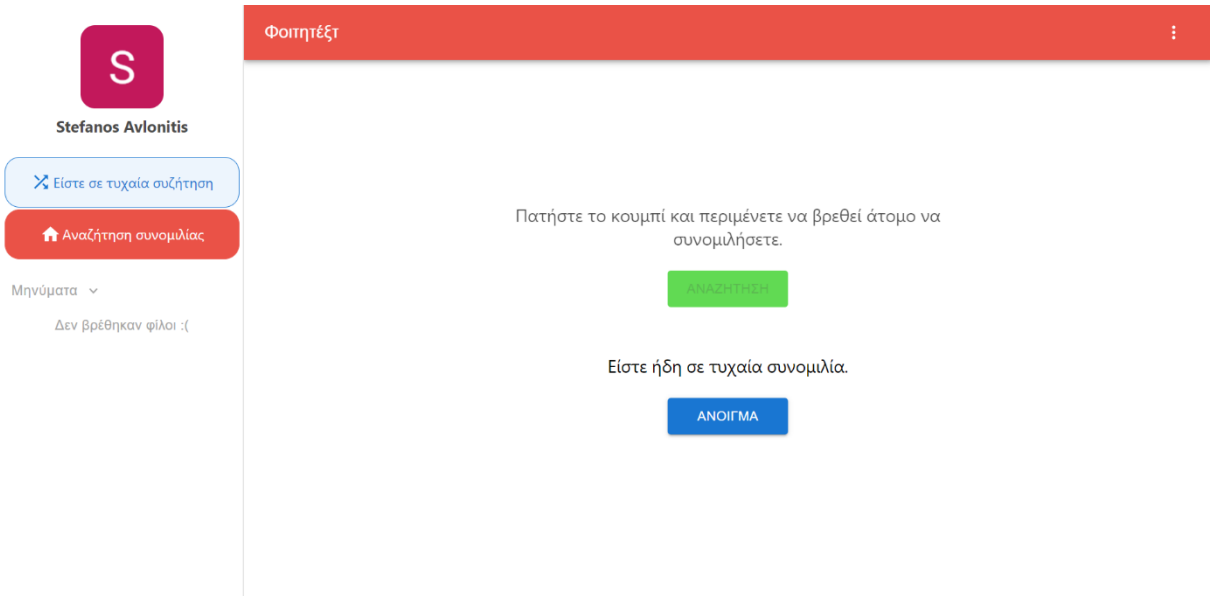
Εικόνα 10.5 Αρχική Σελίδα

Πατώντας το κουμπί αναζήτησης αλλάζει χρώμα και περιεχόμενο το οποίο δείχνει στον χρήστη ότι πλέον βρίσκεται σε αναζήτηση συνομιλίας Εικόνα 10.6.



Εικόνα 10.6 Αναζήτηση συνομιλίας

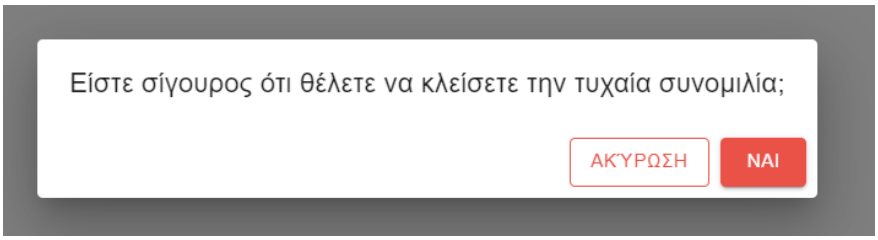
Μόλις βρεθεί τυχαίο άτομο για συνομιλία όπως φαίνεται στην εικόνα Εικόνα 10.7 ενημερώνει τον χρήστη με έναν Text ότι βρέθηκε συνομιλία και εμφανίζεται το κουμπί «ΑΝΟΙΓΜΑ». Πατώντας το ο χρήστης μεταφέρετε σε τυχαία συνομιλία, επίσης στο SideBar εμφανίζεται ένα επιπλέον κουμπί το οποίο και αυτό με το πάτημα του μεταφέρει τον χρήστη στην τυχαία συνομιλία.



Εικόνα 10.7 Τυχαία συνομιλία βρέθηκε

Παρακάτω στην Εικόνα 10.8, ο χρήστης πλέον έχει μεταφερθεί στην τυχαία συνομιλία και έχουν αρχικά αλλάξει ο τίτλος στο AppBar σε “Άγνωστος / Άγνωστη” και προστέθηκε το κουμπί προσθήκης φίλου. Έπειτα στον βασικό container πλέον υπάρχουν τα στοιχεία που απαρτίζουν την συνομιλία όπως είναι τα μηνύματα που μέχρι τώρα υπάρχει μόνο το αυτοματοποιημένο μήνυμα ενημέρωσης ότι οι δύο χρήστες συνδέθηκαν. Επίσης το κάτω μέρος φαίνεται το input μέσω του οποίου γράφονται και στέλνονται τα μηνύματα και αριστερά υπάρχει ένα κουμπί το οποίο αν πατηθεί εμφανίζει τον διάλογο που φαίνεται στην Εικόνα 10.9 για την αποχώρηση του χρήστη από την τυχαία συνομιλία.

Εικόνα 10.8 Τυχαία συνομιλία

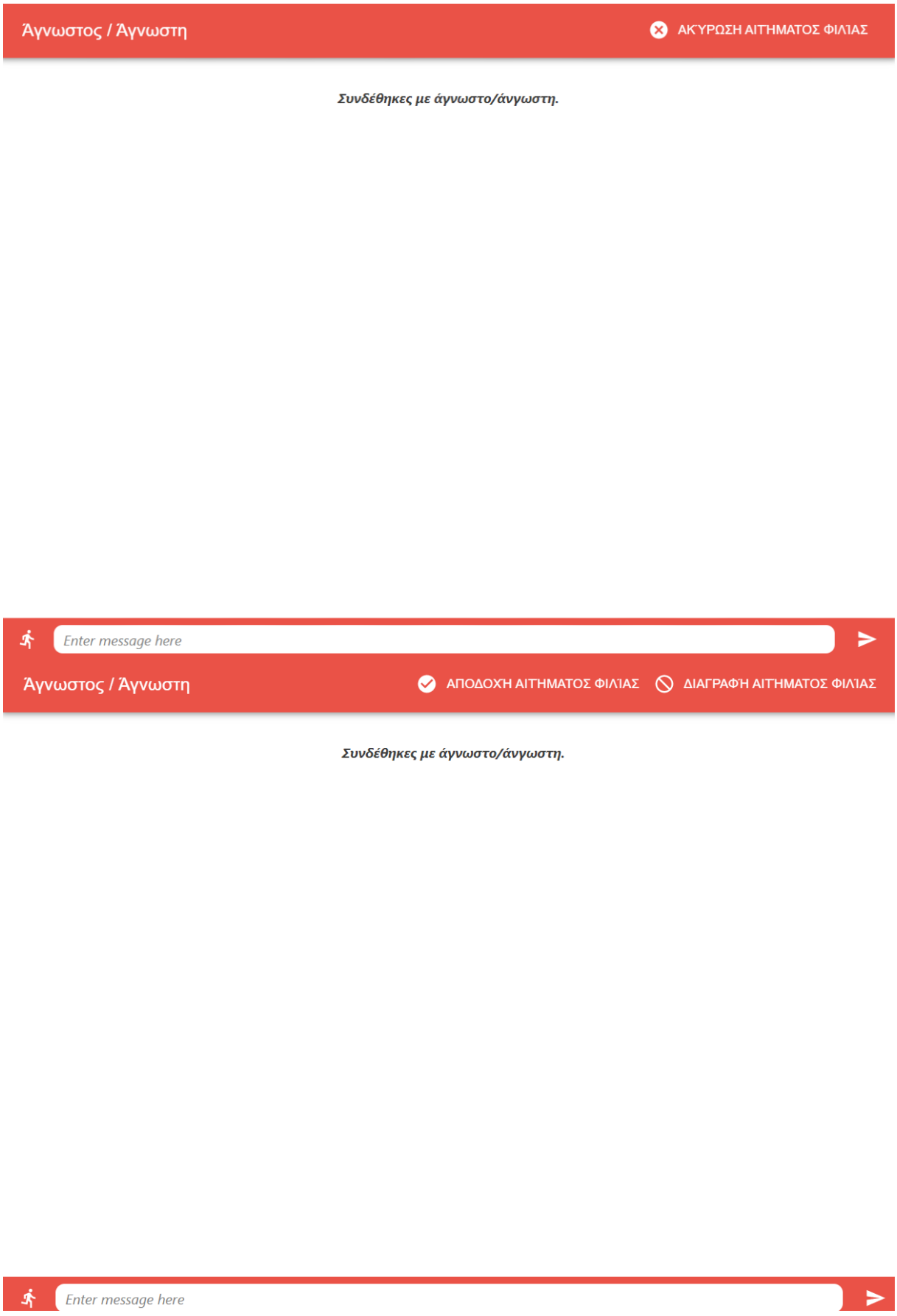


Εικόνα 10.9 Αποχώρηση από τυχαία συνομιλία

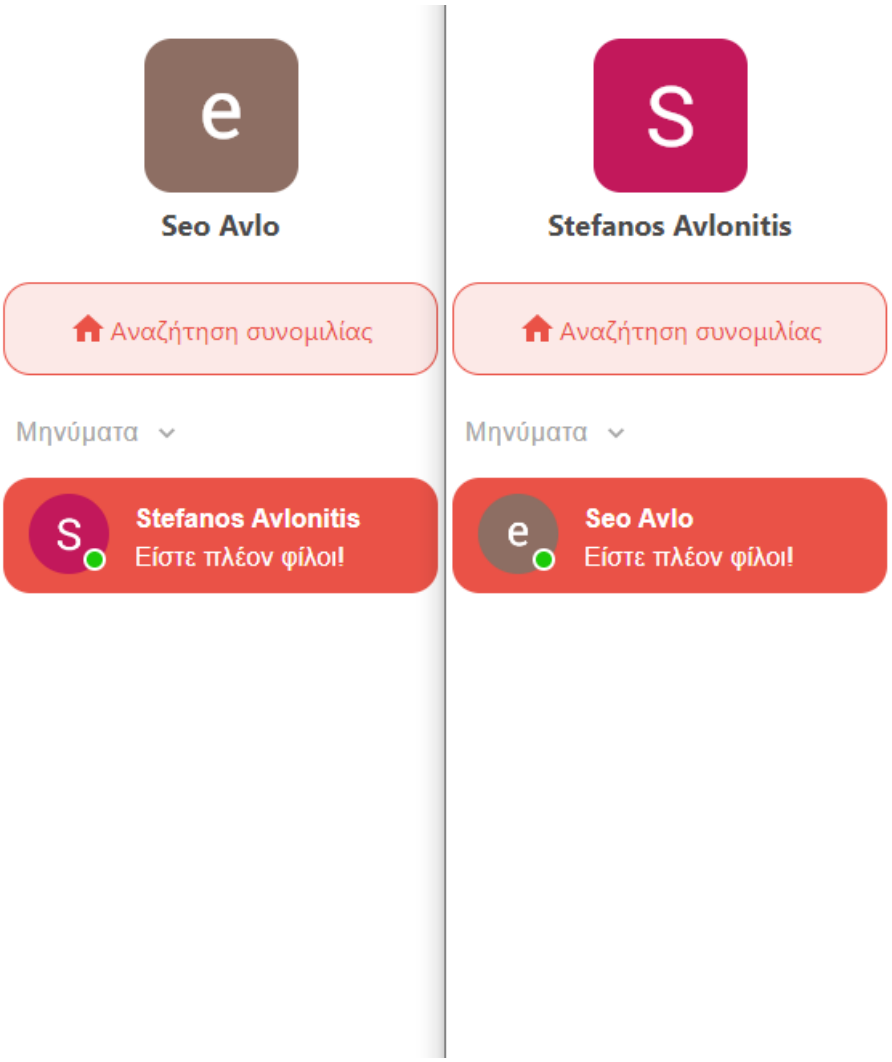
Η Εικόνα 10.10 δείχνει αρχικά τα μηνύματα ανάμεσα στους δύο χρήστες και τι συμβαίνει στην τυχαία συνομιλία όταν ένας από τους δύο πατήσει το κουμπί προσθήκης φίλου. Οι κυριότερες αλλαγές που συμβαίνουν είναι ότι άλλαξαν τα κουμπιά του AppBar δίνοντας τις λειτουργίες απάντησης ή διαγραφής του αιτήματος από τον παραλήπτη του και ακύρωση του μηνύματος από τον αποστολέα.

Καθώς ο παραλήπτης αποδέχεται το αίτημα φιλίας και οι δύο χρήστες πλέον είναι φίλοι, τα δύο chat ενημερώνονται με τα στοιχεία των χρηστών (αλλαγές του τίτλου και των επιλογών του AppBar) και στο SideBar η λίστα μηνυμάτων έχει τον καινούριο φίλο που μόλις έκανε ο χρήστης Εικόνα 10.11. Τέλος παρατηρείται ότι οι χρήστες επειδή έγιναν φίλοι σταμάτησαν να είναι σε τυχαία συνομιλία οπότε το κουμπί τυχαίου chat εξαφανίζεται από την οθόνη.

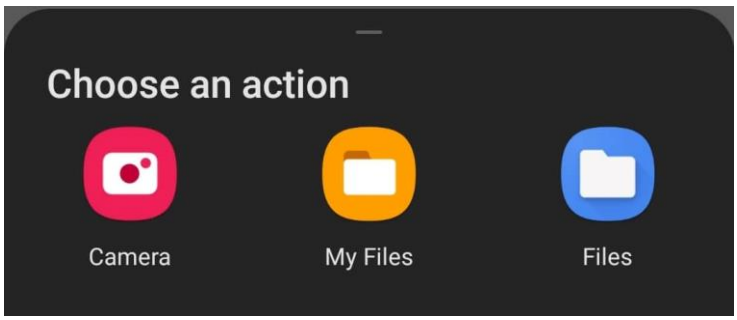
Επιπλέον έχει προστεθεί δίπλα στην εισαγωγή μηνύματος ένα κουμπί το οποίο επιτρέπει την αποστολή εικόνων. πατώντας το ανοίγει ο File Explorer της συσκευής που χρησιμοποιεί ο χρήστης και φιλτράρει μόνο τις εικόνες όπως φαίνεται στην Εικόνα 10.12 και στην Εικόνα 10.13.



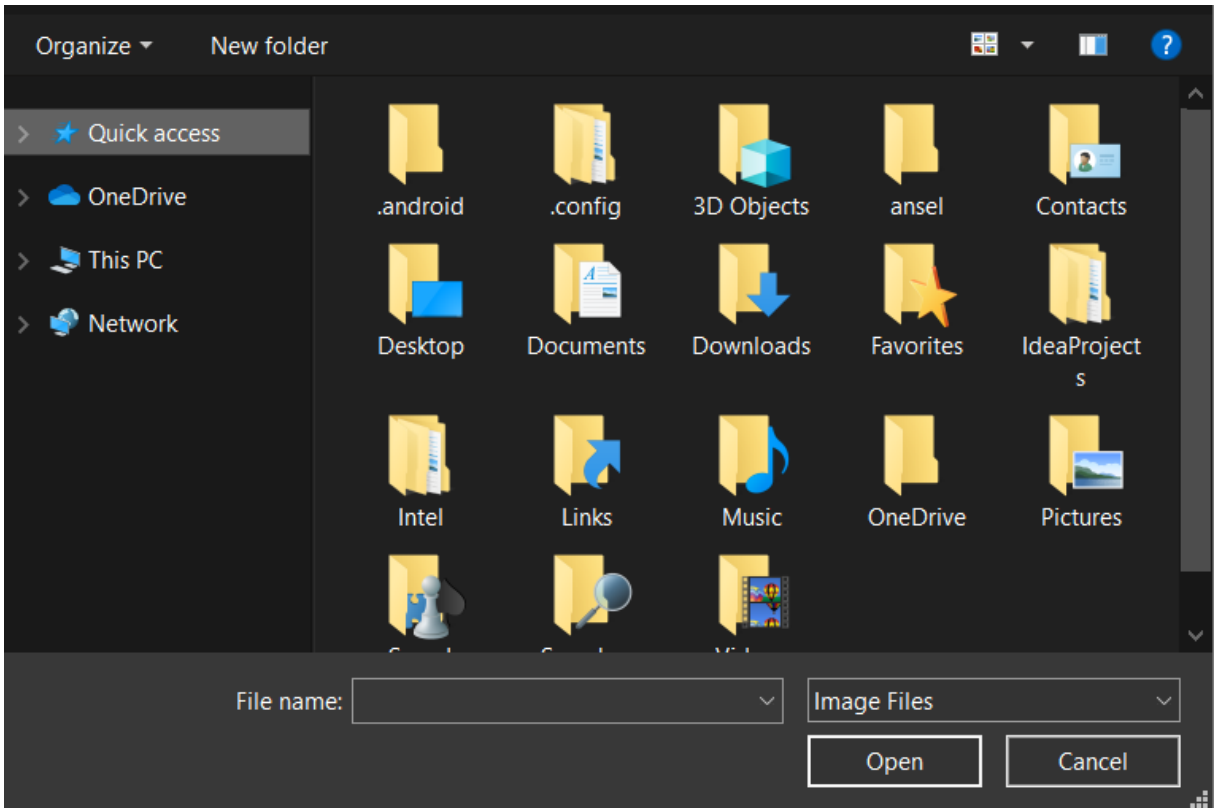
Εικόνα 10.10 Friend Request



Εικόνα 10.11 SideBar με φίλους

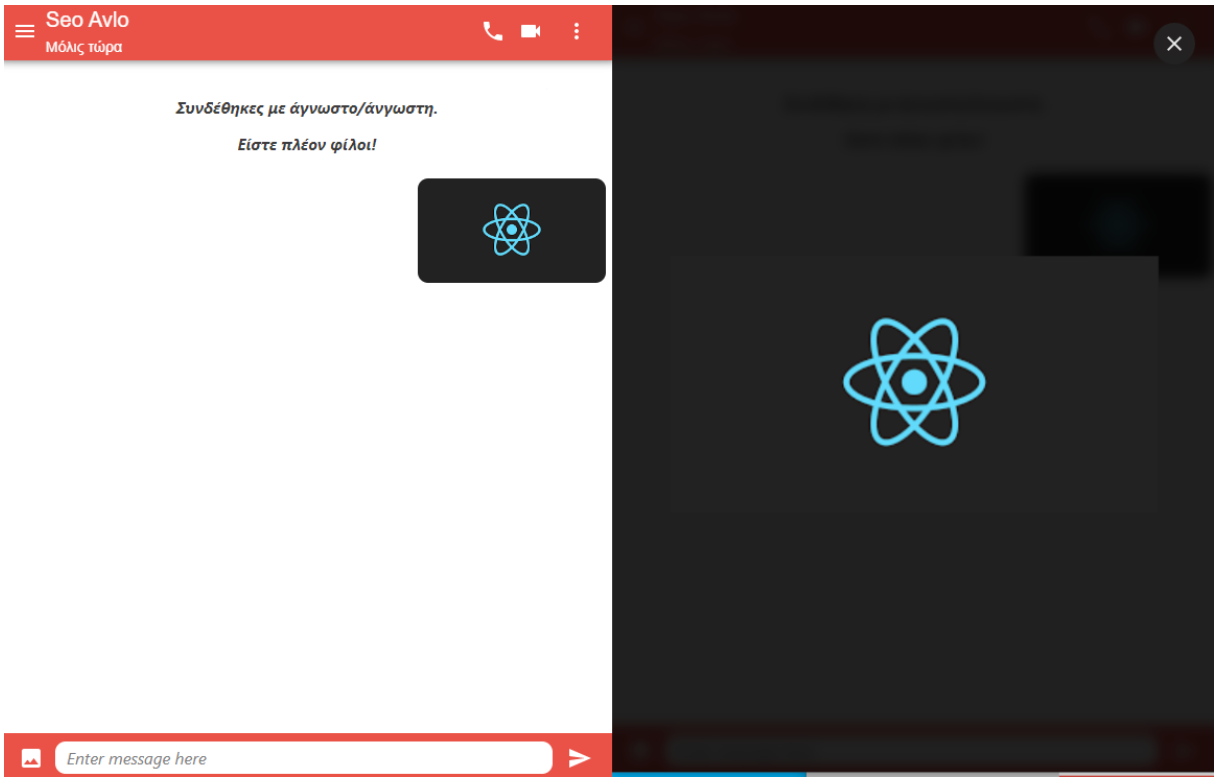


Εικόνα 10.12 File Explorer Android Prompt



Εικόνα 10.13 File Explorer Windows

Επιλέγοντας μία εικόνα θα εμφανιστεί στο μέρος εισαγωγής μηνύματος σαν ένα αντικείμενο chip με το όνομα της εικόνας και πατώντας το κουμπί της αποστολής η επιλεγμένη εικόνα θα αποσταλεί στον χρήστη εικόνα όπως φαίνεται παρακάτω στο αριστερό μέρος της Εικόνα 10.14 . Ο χρήστης επιπλέον έχει την επιλογή να πατήσει (κλικ) πάνω στην εικόνα ώστε να ανοίξει και να εμφανιστεί μεγαλύτερη όπως φαίνεται στο δεξί μέρος της Εικόνα 10.14



Εικόνα 10.14 Photo send

Μπαίνοντας στο τελευταίο κομμάτι των λειτουργιών της εφαρμογής, όταν δύο χρήστες γίνονται φίλοι τους παρέχεται η δυνατότητα να κάνουν κλήσεις και βίντεο-κλήσεις.

Στο AppBar υπάρχουν δύο κουμπιά που δείχνουν τις λειτουργίες που υποστηρίζονται, το κουμπί με το σήμα του τηλεφώνου και το κουμπί με το σήμα της βίντεο-κάμερας αντίστοιχα (Εικόνα 10.15).

Εικόνα 10.15 Κουμπιά κλήσης

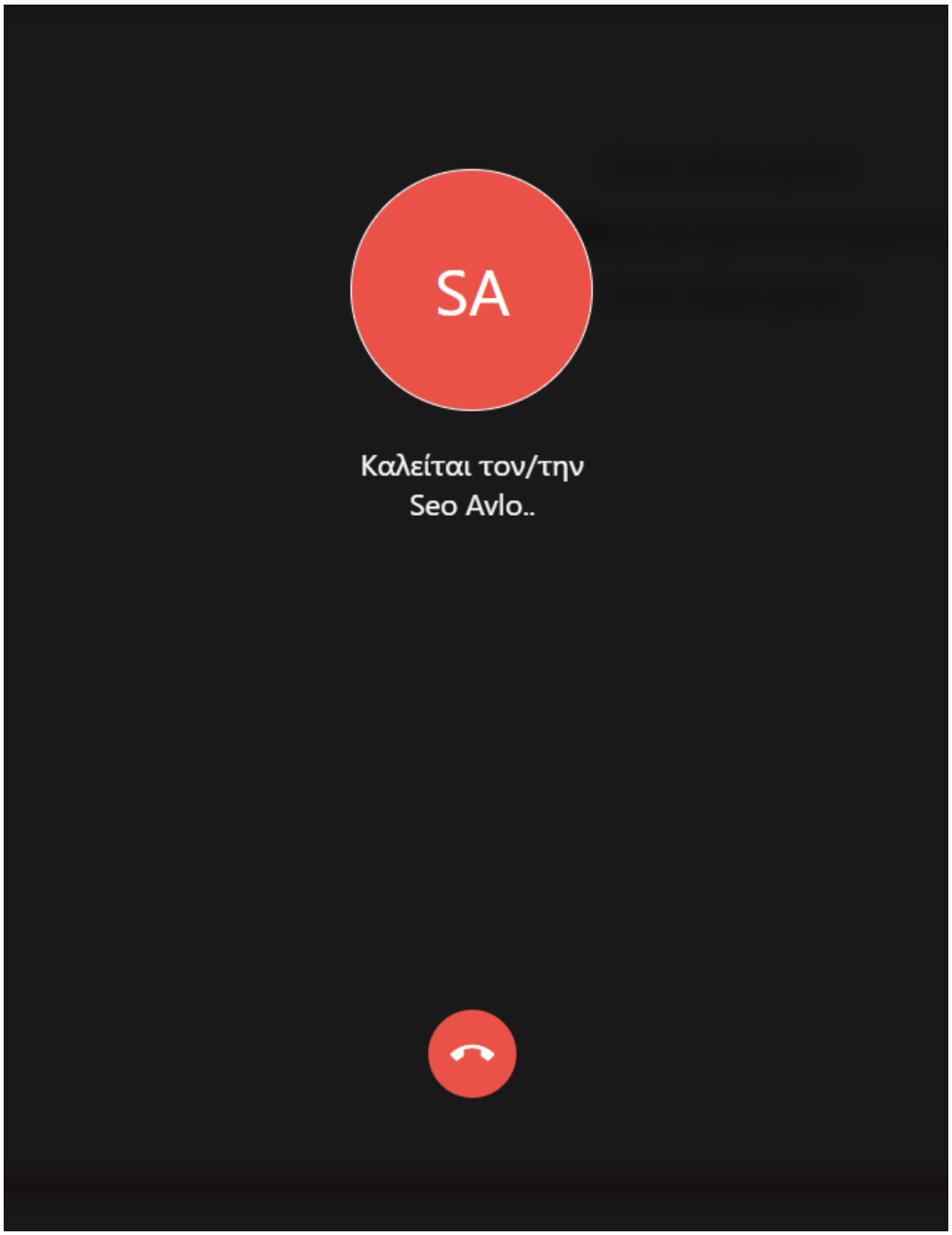
Με το πάτημα ενός εκ των δύο κουμπιών θα εμφανιστεί ένα παράθυρο στον browser που χρησιμοποιεί ο χρήστης που ζητάει από τον χρήστη να δώσει τις άδειες για χρήσης του μικροφώνου της συσκευής και της κάμερας. Είναι σημαντικό ο χρήστης αν θέλει να χρησιμοποιήσει τις λειτουργίες της κλήσης να αποδεχτεί τις άδειες διαφορετικά δεν θα μπορέσει να υλοποιηθεί κάποια κλήση.

Αφού ο χρήστης κάνει αποδοχή των αδειών τότε μπαίνει σε κατάσταση αναμονής - dialing(Εικόνα 10.16) για την απάντηση της κλήσης από τον άλλο χρήστη ενώ παράλληλα και ο άλλος χρήστης μπαίνει σε κατάσταση απάντηση - ringing της κλήσης (Εικόνα 10.17).

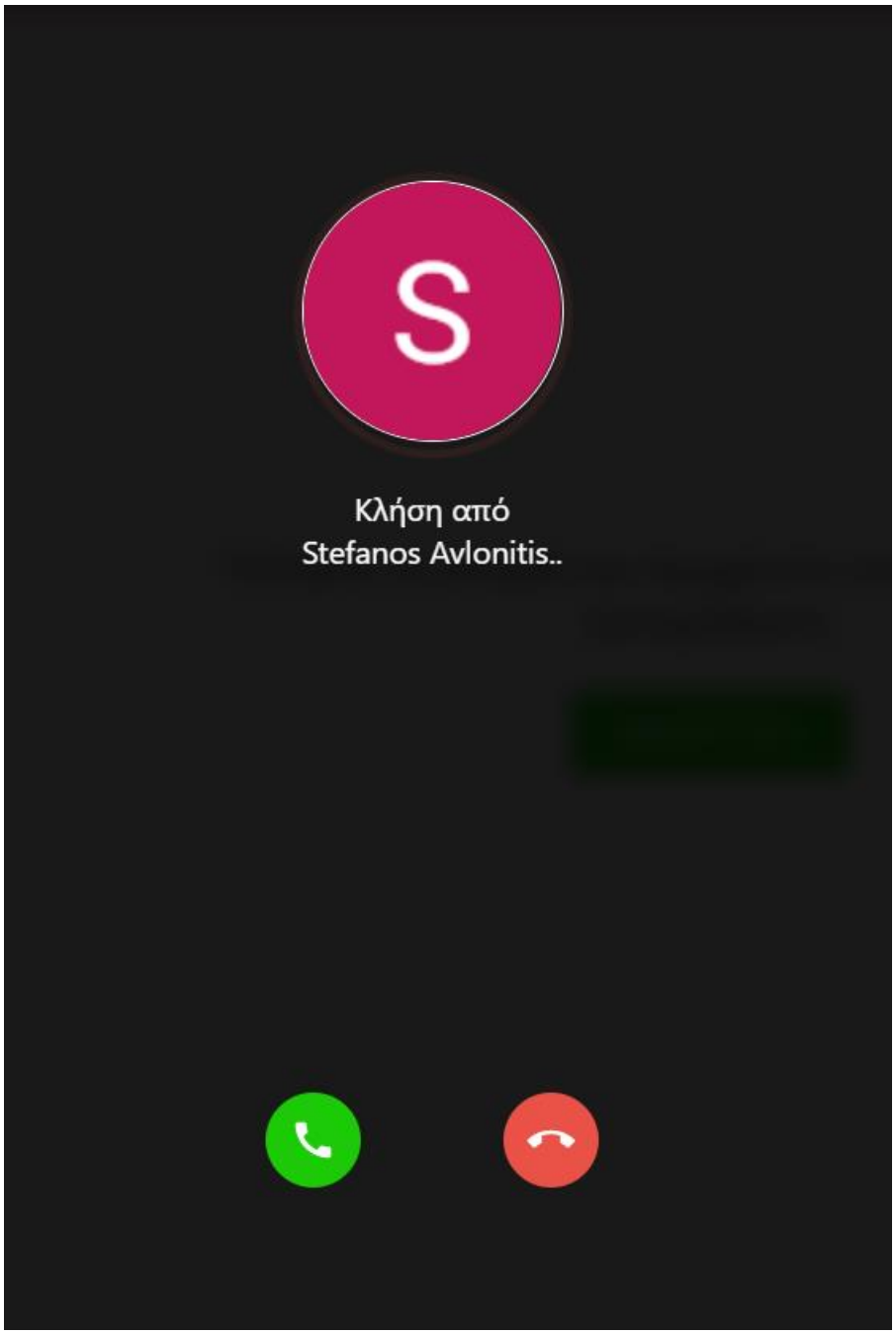
Στην εικόνα του dialing φαίνεται η εικόνα προφίλ του άλλου χρήστη, το όνομα του και ένα κουμπί για την ακύρωση της κλήσης.

Στην εικόνα του ringing φαίνεται η εικόνα προφίλ του χρήστη που καλεί, το όνομα του και σε αυτή την περίπτωση ένα κουμπί για την απάντηση της κλήσης και ένα κουμπί για την απόρριψη της κλήσης.

Να σημειωθεί ότι κατά την διάρκεια του dialing και του ringing και στους δύο χρήστες ακούγεται ο αντίστοιχος ήχος κλήσης στον καθένα.



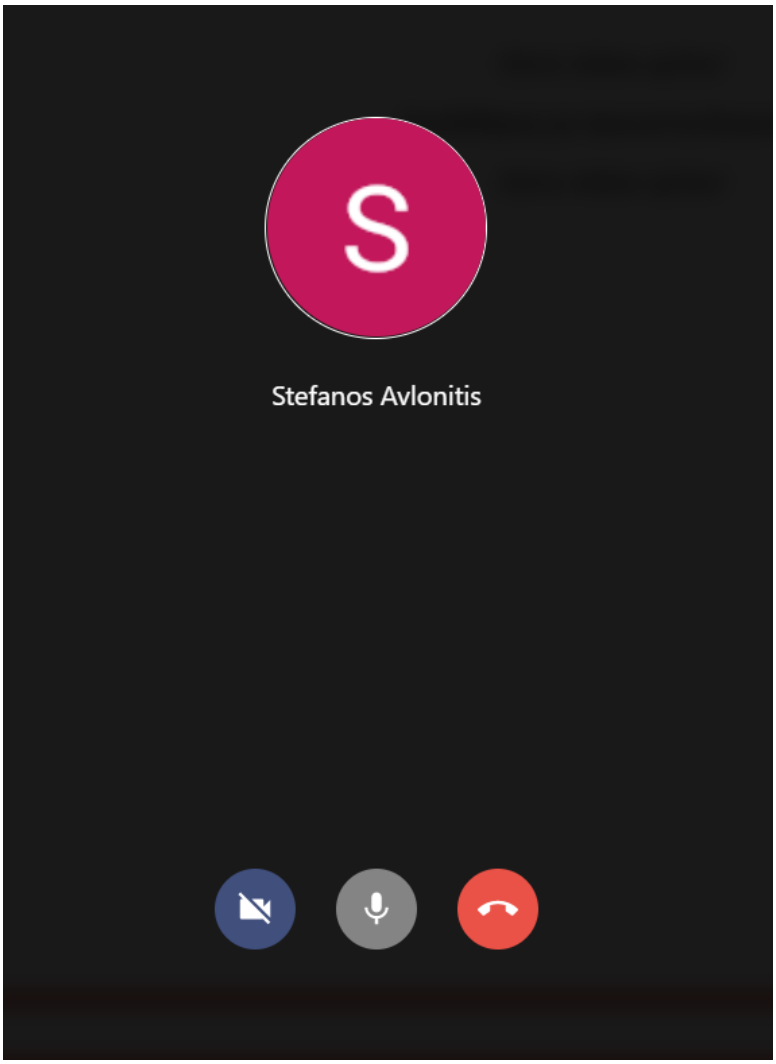
Εικόνα 10.16 Διεπαφή αναμονής κλήσης



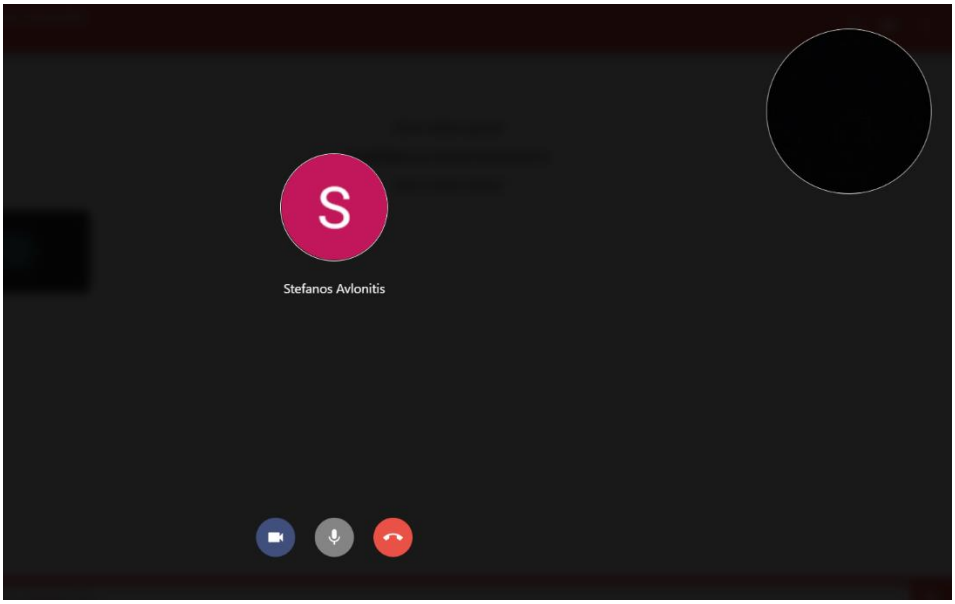
Εικόνα 10.17 Διεπαφή εισερχόμενης κλήση

Αν η κλήση απαντηθεί τότε εμφανίζεται η εικόνα προφίλ του χρήστη που συνομιλούν και το όνομα του και στις δύο πλευρές αντίστοιχα. Επίσης εμφανίζονται 3 κουμπιά (Εικόνα 10.18) που χρησιμεύουν στην ενεργοποίηση/απενεργοποίηση της κάμερας, στην ενεργοποίηση/απενεργοποίηση του μικροφώνου και στον τερματισμό της κλήσης αντίστοιχα.

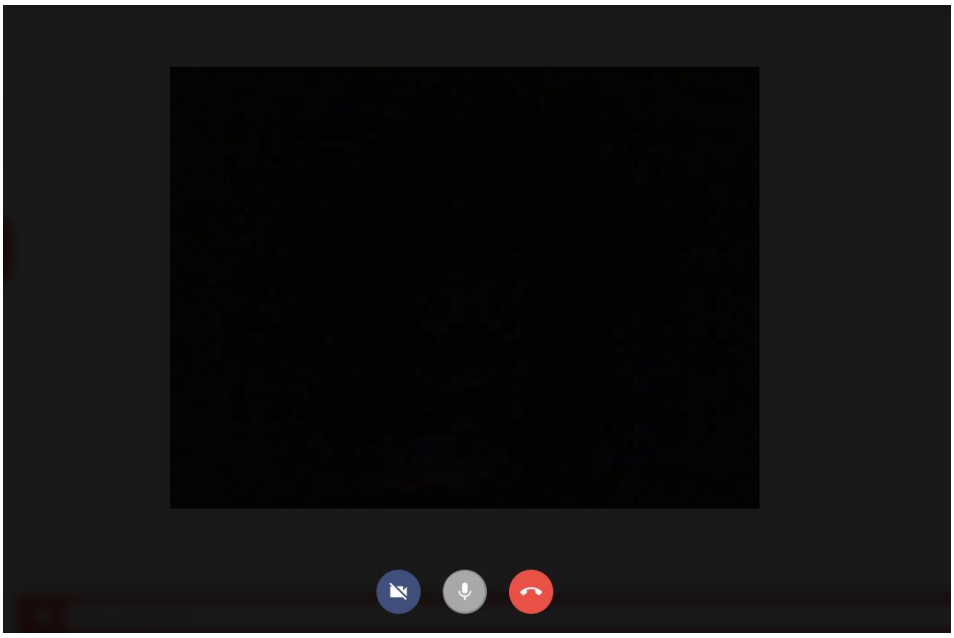
Αν κάποιος εκ των δύο ενεργοποιήσει την κάμερα του (ή αν η κλήση είναι εξαρχής βίντεο-κλήση είναι ήδη ενεργοποιημένη) τότε θα εμφανιστεί αρχικά στο δεξί μέρος της οθόνη του χρήστη που την ενεργοποίησε ένας κύκλος στον οποίο υπάρχει το περιεχόμενο που καταγράφει η κάμερα του (Εικόνα 10.19). Στον άλλον χρήστη θα εμφανιστεί το περιεχόμενο της κάμερας που εκπέμπει ο απέναντι χρήστης (Εικόνα 10.20).



Εικόνα 10.18 Διεπαφή κλήσης ήχου

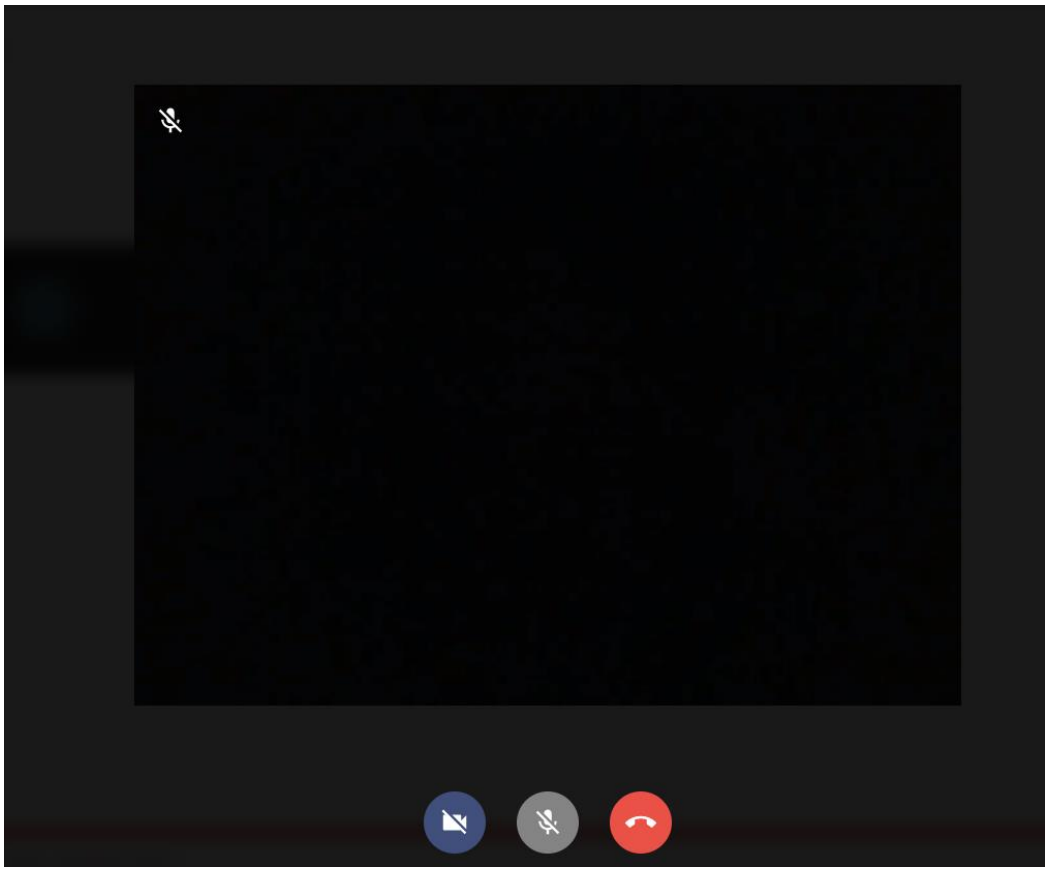


Εικόνα 10.19 Διεπαφή με ανοιχτή κάμερα



Εικόνα 10.20 Διεπαφή με ανοιχτής κάμερα άλλου χρήστη

Μερικές σημειώσεις για τις κλήσεις είναι ότι αρχικά αν η κλήση είναι βίντεο κλήση τότε τα μικρόφωνα και των δύο χρηστών θα παραμείνουν κλειστά (όπως φαίνεται στο κάτω μέρος στην Εικόνα 10.21) και θα πρέπει να τα ενεργοποιήσουν οι ίδιοι οι χρήστες. Το παραπάνω συμβαίνει για την επίτευξη καλύτερης εμπειρίας χρήσης ώστε οι χρήστες να μην έρθουν σε άβολη θέση την ώρα της βίντεο-κλήσης με την μετάδοση ανεπιθύμητων ήχων. Επίσης όταν το μικρόφωνο κάποιου εκ των δύο είναι κλειστό τότε εμφανίζεται ένα εικονίδιο κλειστού μικροφώνου στην απέναντι πλευρά όπως φαίνεται στο αριστερό πάνω άκρου του βίντεο στην Εικόνα 10.21.

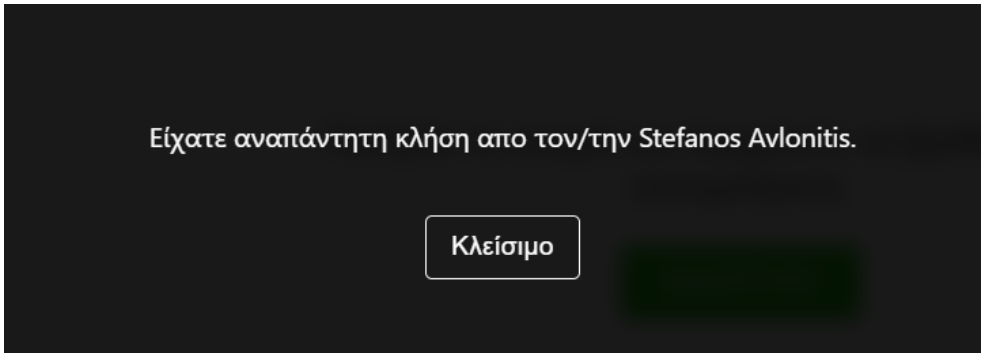


Εικόνα 10.21 Διεπαφή βίντεο-κλήσης με κλειστό μικρόφωνο

Όταν η κλήση για κάποιο λόγο τελειώσει ή απορριφθεί εμφανίζεται στους χρήστες κατάλληλο μήνυμα όπως φαίνεται στην Εικόνα 10.22. Τα πιθανά μηνύματα που μπορεί να εμφανιστούν στους χρήστες είναι

1. Η κλήση με τον/την {όνομα χρήστη} ακυρώθηκε.
2. Η κλήση με τον/την {όνομα χρήστη} ολοκληρώθηκε.
3. Είχατε αναπάντητη κλήση από τον/την {όνομα χρήστη}.
4. Ο/Η {όνομα χρήστη} δεν απάντησε.
5. Η κλήση τέλειωσε

Με το πάτημα του κουμπιού “Κλείσιμο” ο χρήστης επαναφέρεται στην συνομιλία.



Εικόνα 10.22 Παράδειγμα μηνύματος τέλους κλήσης

10.5 Επίλογος

Στο παραπάνω κεφάλαιο αρχικά έγινε μία σύντομη εισαγωγή στην γλώσσα SASS που χρησιμοποιήθηκε για την εφαρμογή των styles στην εφαρμογή, παρουσιάστηκαν τα σημαντικότερα σημεία που προσφέρει και που χρησιμοποιήθηκαν κατά την ανάπτυξη και σχεδίαση των διεπαφών χρήστη. Επίσης αναφέρθηκε το Material, οι Components και τα Icons που προσφέρει και χρησιμοποιούνται στην εφαρμογή. Τέλος παρουσιάστηκε η διεπαφή χρήστη και οι λειτουργίες της εφαρμογής με χρήση εικόνων, και σειρά την οποία ακολουθεί ένας χρήστης την πρώτη φορά που χρησιμοποιεί την εφαρμογή.

Κεφάλαιο 11ο: Συμπεράσματα

Τελειώνοντας με την ανάπτυξη της εφαρμογής, στο πιο σημαντικό συμπέρασμα που καταλήγω είναι ότι για να κατασκευαστεί μία σωστά δομημένη εφαρμογή τόσο σε επίπεδο κώδικα όσο και σε επίπεδο διεπαφών πέρα από την γνώση των τεχνολογιών χρειάζεται και αρκετή εμπειρία. Η κάθε τεχνολογία – αρχιτεκτονική μπορεί να προσφέρει σημαντικά πλεονεκτήματα στον προγραμματιστή αλλά ο προγραμματιστής πρέπει να είναι σε θέση να χρησιμοποιήσει την κάθε μία από αυτές με τον βέλτιστο τρόπο.

Η επιλογή των τεχνολογιών που χρησιμοποιήθηκαν έκανε την διαδικασία ανάπτυξης πιο ευχάριστη καθώς οι τεχνολογίες παρουσιάζουν χαρακτηριστικά που δεν είχα χρησιμοποιήσει και έτσι ήρθα αντιμετώπως με καινούριες έννοιες και προκλήσεις. Η ReactJS έκανε την κατασκευή και διαχείριση των διεπαφών αρκετά εύκολη και σε συνδυασμό με το Model-View-Intent pattern η ανάπτυξη των διάφορων λειτουργιών έγινε με συμπαγή τρόπο και η διαχείριση των λαθών σε επίπεδο View βελτιστοποιήθηκε. Το Firebase με αρκετές από τις λειτουργίες του αποδείχθηκε από τα πιο σημαντικά εργαλεία και η επιλογή είχε σημαντικό αντίκτυπο. Το WebRTC πρόσθεσε με την μεθοδολογία του ένα σημαντικό feature της εφαρμογής που είναι οι κλήσεις και χωρίς αυτό η επίτευξη του θα ήταν πολύ πιο δύσκολη και οι κλήσεις σίγουρα πιο αργές. Τέλος το Domain Driven Design παρότι χρειάστηκε επιπλέον μελέτη για την υιοθέτηση του, έκανε την σχεδίαση και υλοποίηση της εφαρμογής ξεκάθαρη και με συγκεκριμένους στόχους με αποτέλεσμα νέα features και αλλαγές που προέκυψαν κατά την ανάπτυξη να γίνονται εύκολα.

Στο μέλλον κάποιες προσθήκες που θα μπορούσαν να γίνουν είναι αρχικά η υποστήριξη περισσότερων τρόπων εγγραφής και σύνδεσης στην εφαρμογή. Επίσης θα μπορούσαν να προστεθούν περισσότεροι τρόποι διασυνδέσεις με άλλους χρήστες όπως η αναζήτηση ή η αναζήτηση μέσω της τοποθεσίας. Μία ακόμα λειτουργία θα μπορούσε να είναι η υποστήριξη συζητήσεων με πολλά άτομα ταυτόχρονα. Τέλος μία χρήσιμη προσθήκη θα ήταν η εφαρμογή να είναι διαθέσιμη ως PWA (Progressive Web Apps) και ο χρήστης θα μπορούσε να την εγκαταστήσει σαν native εφαρμογή στο κινητό του, το οποίο μέσω της αρχιτεκτονικής της εφαρμογής και την κατάλληλη μελέτη θα ήταν εύκολα υλοποιήσιμο χωρίς πολλές αλλαγές.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] WebRTC, "Real-time communication for the web," [Online]. Available: https://webrtc.org/.

[2] TypeScript, "TypeScript: JavaScript With Syntax For Type," [Online]. Available: https://www.typescriptlang.org/.

[3] Sass, "Sass: Syntactically Awesome Style Sheets," [Online]. Available: https://sass-lang.com/.

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

[5] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, 2017.

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

[7] ReactJS, "A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org/.

[8] MUI, "Material UI for React," [Online]. Available: https://mui.com/.

[9] "Material Design," [Online]. Available: https://material.io/.

[10] "Cloud Functions for Firebase," [Online]. Available: https://firebase.google.com/docs/functions.

[11] "Firebase Realtime Database," [Online]. Available: https://firebase.google.com/docs/database.

[12] "Material Icons," [Online]. Available: https://fonts.google.com/icons.

[13] "Firebase," [Online]. Available: https://firebase.google.com/.

[14] "Firebase Security Rules," [Online]. Available: https://firebase.google.com/docs/rules.

[15] "Firebase Authentication," [Online]. Available: https://firebase.google.com/docs/auth.

[16] "Cloud Firestore," [Online]. Available: https://firebase.google.com/docs/firestore.

[17] FirebaseExtended, "RxFire," [Online]. Available: https://github.com/FirebaseExtended/rxfire.

[18] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, 2003.

[19] Cycle.js, "Model View Intent," [Online]. Available: https://cycle.js.org/model-view-intent.html.

[20] "Cloud Storage for Firebase," [Online]. Available: https://firebase.google.com/docs/storage.