ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ДИГИТАЛНИ НАУКИ „СОФТУНИ СВЕТЛИНА“

# ДИПЛОМНА РАБОТА

ЗА ПРИДОБИВАНЕ НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ

НА ТЕМА:

**Проектиране и реализиране на чат приложение в уеб среда**

Изготвил: **Мартин Радославов Иванов**

Научен ръководител: **Светлана Александрова Виктомирова**

03.05.2023г.

Гр. София

# Съдържание

[**Дипломна работа 1**](#_3tbugp1)

[**Съдържание**](#_30j0zll) **2**

[**Увод**](#_1fob9te) **4**

[Проблем](#_3znysh7) 4

[Цели на дипломния проект](#_2et92p0) 4

[Задачи, произтичащи от целите](#_28h4qwu) 4

[**Глава 1. Комуникацията и важността на онлайн комуникацията**](#_tyjcwt) **6**

1.1. Комуникацията и нуждата за свързаност 6

1.2. Какво представлява “чат приложение”? 7

1.3. Предимствата на чат приложенията 7

1.4. Проучване на различни чат приложения 8

1.4.1. Messenger 8

1.4.2. Discord 10

1.4.3. Viber 11

[**Глава 2. Същност на приложението**](#_lnxbz9) **13**

2.1. ChatApp 13

2.2. Използвани технологии 13

2.2.1. React JS 13

2.2.2. Firebase 14

2.3. Използвани допълнителни библиотеки 15

2.3.1. React Router 15

2.3.2. React Firebase Hooks 16

2.4. Компоненти 16

2.4.1. Главни компоненти 17

2.4.2. Auth компоненти 17

2.4.3. Chat компоненти 18

2.4.4. Map компоненти 18

[**Глава 3. Проектиране на приложението**](#_qsh70q) **20**

3.1. Firebase 20

3.1.1. Автентикация на потребители 20

3.1.2. Хранилище 21

3.1.3. База данни 21

3.1.3.1. users 22

3.1.3.2. groups 22

3.1.3.3. private-rooms 22

3.1.3.4. messages 23

3.2. Кодова имплементация 23

3.2.1. Настройване на връзката между сървърната и клиентската част 23

3.2.2. Потребителска автентикация и оторизация 24

3.2.3. Показване на чат стаи 26

3.2.4. Показване и изпращане на съобщения 30

[**Глава 4. Ръководство за употреба на приложението**](#_3fwokq0) **32**

[**Заключение**](#_1v1yuxt) **36**

[**Информационни източници**](#_4f1mdlm) **37**

[**Приложение**](#_2u6wntf) **38**

[**Рецензия на дипломен проект**](#_19c6y18) **39**

# Увод

Настоящата разработка има за цел да запознае и да информира читателя с проблема и необходимостта за комуникация в уеб среда като за тази цел се прави анализ на всички досега съществуващи решения и се имплементира приложение за изпращане на съобщения в уеб среда с ReactJS и Firebase. За изпълнението на тези цели ще бъдат разгледани различните задачи, свързани с тяхното изпълнение.

## Проблем

Ние, хората, сме социални същества и множеството от глобални събития през последните няколко години (COVID-19 пандемията, противоепидемичните мерки, социалната дистанция) успяха да ни докажат само едно нещо: комуникацията и нуждата за свързаност са повече от необходими в настоящето. С развитието на Интернет и различните технологии, комуникацията между хората никога досега през историята на човечеството не е била толкова развита, като тя успява да свърже различни хора от всички краища на света и благодарение на това разпространяването на различна информация, за добро или лошо, вече става доста по-лесно. Благодарение на многото вече съществуващи приложения за изпращане на съобщения (или още познати като чат приложения) много от проблемите с традиционната форма на комуникация или другите форми за комуникация през Интернет като имейлите биват разрешени (времеви и пространствени ограничения, асинхронна комуникация, претоварване и сигурност).

## Цели на дипломния проект

Целта на настоящата дипломна работа е да се разработи уеб приложение за изпращане на съобщения като тя ще има клиентска и сървърна част. Преди изработката ще бъде направено проучване на вече съществуващи приложения и на различните технологии, които ще бъдат използване в етапа на имплементация.

## Задачи, произтичащи от целите

Във връзка с поставената цел за разработване на приложение за комуникация в уеб среда са поставени следните задачи:

* проучване и подготовка на обзор на проблемната област
* проучване на различни работещи решения на дадения проблем
* проучване на различните технологии за проектиране на сървърната и клиентската част
* проектиране на модел на база данни
* реализация на уеб приложение за изпращане на съобщения: клиентска част (уеб приложение за комуникация в скрити или публични чат стаи) и сървърна част (вдигане на cloud база данни, както и настройване на автентикацията на потребителите и съхранението на снимки в базата данни)
* извеждане на изводи и заключенията на базата на разработения проект

# Глава 1. Комуникацията и важността на онлайн комуникацията

С напредването на технологиите успяваме да преодолеем различните трудности от всекидневния живот като биват предоставени по-бързи и по-ефективни решения. Пример за това биват приложенията за изпращане на съобщения, които решават множеството пречки и недостатъци на традиционната комуникация и скорошните глобални събития ни показаха, че такива приложения са повече от необходими на обществото.

## 1.1. Комуникацията и нуждата за свързаност

Общуването и свързаността между хората са от съществено значение за физическото и психическото им благополучие. Чрез общуването изграждаме и поддържаме взаимоотношения, обменяме идеи и знания, изразяваме емоциите си и си сътрудничим за постигане на общи цели. Пандемията на COVID-19 обаче постави значителни предизвикателства пред начина, по който си комуникираме и се свързваме помежду си.

С необходимостта от социална дистанция и ограничаване на физическите взаимодействия хората се обърнаха към технологиите, за да поддържат връзка помежду си. Видеоконференциите, приложенията за изпращане на съобщения и платформите на социалните медии се превърнаха във важни инструменти за свързване на хората по време на пандемията. Благодарение на тях, те позволиха на хората да продължат да работят, да учат и да общуват въпреки глобалните предизвикателства.

Отвъд пандемията комуникацията и връзката между хората играят решаваща роля в живота ни. Проучванията показват, че социалната изолация и самотата могат да доведат до негативни последици за здравето, включително депресия, тревожност и сърдечно-съдови заболявания. От друга страна е доказано, че положителните социални взаимоотношения и социалната подкрепа имат положително въздействие върху психическото и физическото здраве. Общуването и връзката с другите хора са от решаващо значение за осигуряване на социална подкрепа и намаляване на чувството за изолация и самота.

Разпространението на информация е също толкова важно, особено по време на криза. По време на пандемията на COVID-19 точната и навременна информация бе от решаващо значение за подпомагане на хората да вземат информирани решения за своето здраве и безопасност. Дезинформацията и фалшивите новини могат да доведат до объркване, паника и дори до увреждане на общественото здраве. Ролята на комуникацията за разпространение на точна информация бе от голяма значимост по време на пандемията и докато продължаваме да се справяме с предизвикателствата на пандемията и след това, трябва да дадем приоритет на комуникацията, свързаността и разпространението на точна информация.

## 1.2. Какво представлява “чат приложение”?

Приложенията за изпращане на съобщения (или още познати като чат приложения) са приложения, които позволяват на потребителите да изпращат и получават текстови съобщения, гласови съобщения и мултимедийно съдържание по интернет. Пазарът на тези приложения нарасна значително през последните няколко години, като милиарди потребители по целия свят разчитат на тях. Тези приложения се предлагат с широк спектър от функции, като групов чат, гласови и видео разговори, споделяне на файлове, споделяне на местоположение и други.

Чат приложенията се превърнаха в съществена част от ежедневната комуникация както за физически лица, така и за бизнеса. Те революционизираха начина, по който хората си комуникират помежду си, предлагайки по-удобна и рентабилна алтернатива на традиционните методи за комуникация като телефонни обаждания и текстови съобщения. С чат приложенията потребителите могат да общуват с хора по целия свят и по всяко време, стига да имат интернет връзка и отвсякъде.

## 1.3. Предимствата на чат приложенията

Едно от предимствата на чат приложенията е способността им да осигуряват незабавна комуникация, което позволява на потребителите бързо да изпращат и получават съобщения без забавянето на традиционните методи за комуникация като електронна поща или обикновена поща. Тази бързина е била от особена важност по време на пандемията, тъй като позволява на хората бързо да проверяват състоянието на близките си, да споделят актуализации и да получават важна информация. Освен това тези приложения често включват функции като разписки за прочетени съобщения и индикатори за набиране на текст, които помагат за намаляване на безпокойството и несигурността, свързани с комуникацията.

Приложенията за изпращане на съобщения също така предоставят на потребителите по-голяма гъвкавост и контрол върху комуникацията. Например те позволяват на потребителите да избират вида комуникация, в която искат да участват, като например текстови, аудио или видео разговори. Това позволява по-персонализирано и адаптивно общуване, което е особено важно по време на стрес и несигурност. Приложенията за изпращане на съобщения също така позволяват асинхронна комуникация, което означава, че потребителите могат да четат и да отговарят на съобщенията със свое собствено темпо, а не да изпитват натиск от незабавен отговор.

Друго предимство на чат приложенията е способността им да улесняват груповата комуникация. По време на пандемията те изиграха решаваща роля, като помогнаха на хората да поддържат връзка със семейството, приятелите и колегите си въпреки предизвикателствата на социалната дистанция. Функциите за групово изпращане на съобщения позволяват лесно координиране на виртуални събития, споделяне на информация и предоставяне на подкрепа на нуждаещите се. Установено е, че този вид групова комуникация намалява чувството за самота и социална изолация, които доказано оказват отрицателно въздействие върху психичното здраве.

В допълнение към тези предимства, приложенията за изпращане на съобщения са също така изключително достъпни и рентабилни. Повечето такива приложения са безплатни за използване и изискват само интернет връзка, което ги прави достъпни за хората във всички части на света, независимо от социално-икономическия им статус. Тази достъпност е особено важна по време на пандемията, тъй като позволява на хората да поддържат връзка с близките си дори когато са изправени пред финансови предизвикателства и загуба на работа.

Важността на тези приложения е особено видима по време на пандемията COVID-19, тъй като мерките за социална дистанция принудиха хората да разчитат на технологиите за комуникация повече от всякога. Изследванията показват, че тези мерки са оказали значително въздействие върху психичното здраве, като са отчетени повишени нива на тревожност, стрес и депресия (Holmes et al., 2020). Чат приложенията са осигурили на много хора връзка с близките си и да имат достъп до важна информация във време на голяма несигурност.

## 1.4. Проучване на различни чат приложения

С цел проучване бяха разгледани три от най-известните чат приложения: Messenger, Discord и Viber. Макар и на основно ниво да си приличат по възможността за комуникация, всяко едно от тези приложения съдържа различни функционалности и начини, по които може да се общува.

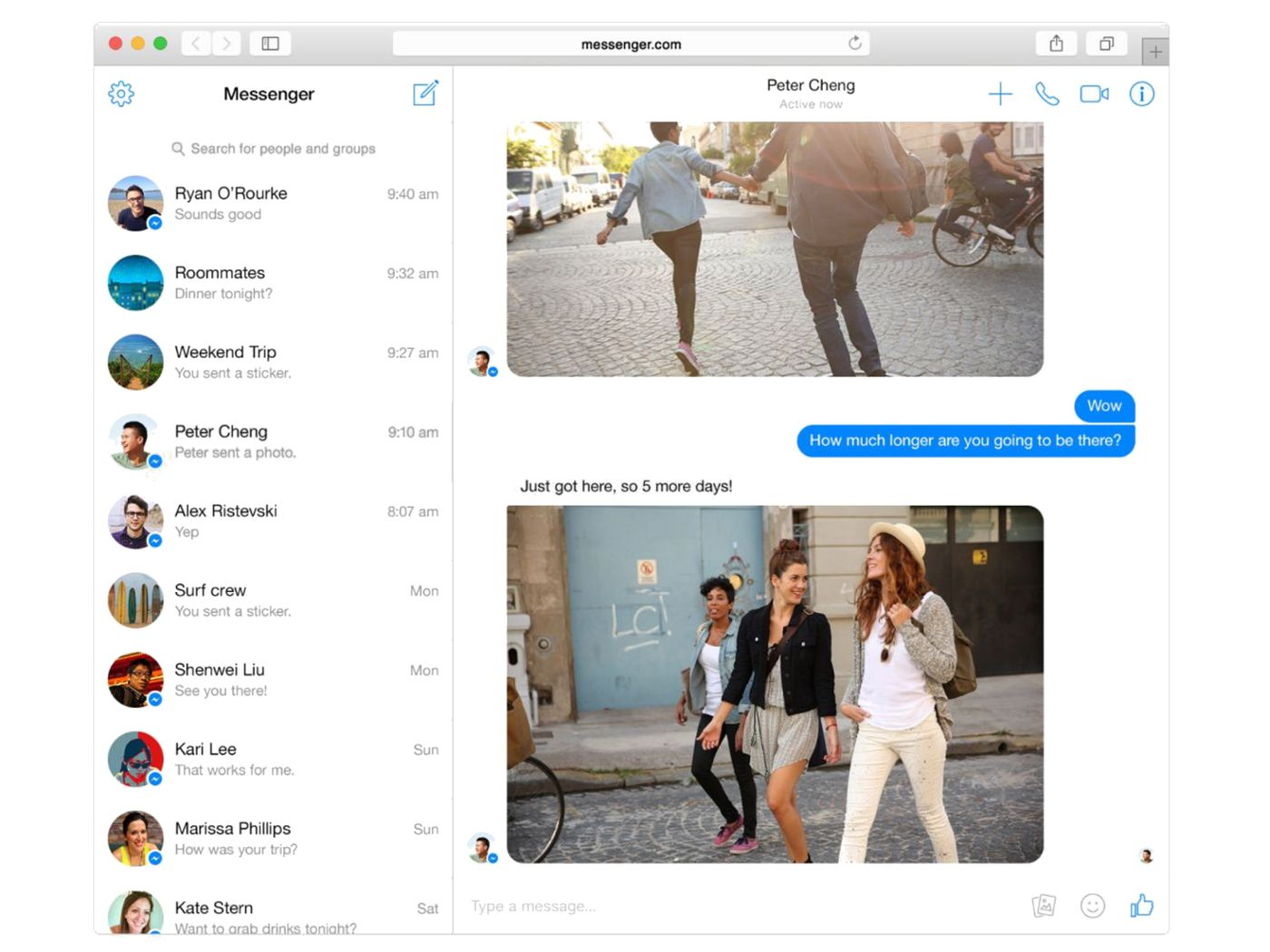
### 1.4.1. Messenger

Messenger е чат приложение, разработено от Facebook, Inc. Първоначално бе стартирано през 2011 г. като Facebook Messenger, но по-късно бе ребрандирано като Messenger. Приложението позволява на потребителите да изпращат съобщения, да правят гласови и видео разговори, да споделят снимки и видеоклипове и да играят игри с приятели и семейство, които също са в приложението.

Една от функциите, която отличава Messenger от другите чат приложения, е неговата безпроблемна интеграция с Facebook. Тъй като Facebook е най-голямата в света платформа за социални медии, улеснява потребителите да се свързват с приятели и семейство, които вече са във Facebook. Потребителите могат да изпращат съобщения директно до своите приятели във Facebook, без да се нуждаят от телефонни номера. Messenger също така позволява на потребителите да създават групови чатове с до 250 души, което го прави чудесен инструмент за организиране на събития или поддържане на връзка с големи групи хора.

Messenger също има широк спектър от функции, които го правят да се откроява от другите подобни на него приложения. Една от тези характеристики е Messenger Rooms, която стартира през 2020 г. в отговор на пандемията COVID-19. Messenger Rooms позволява на потребителите да създават виртуални зали за срещи, които могат да държат до 50 души наведнъж, като общуват чрез видео разговори. Тази функция се използва широко от бизнеса и организациите с цел провеждане на виртуални срещи и събития, както и от обикновени хора, за да останат свързани с приятели и семейство по време на социалното дистанциране.

Можем да видим как изглежда потребителският интерфейс на Messenger на фигура 1-1:



Фигура 1-1: Потребителски интерфейс на приложението Messenger

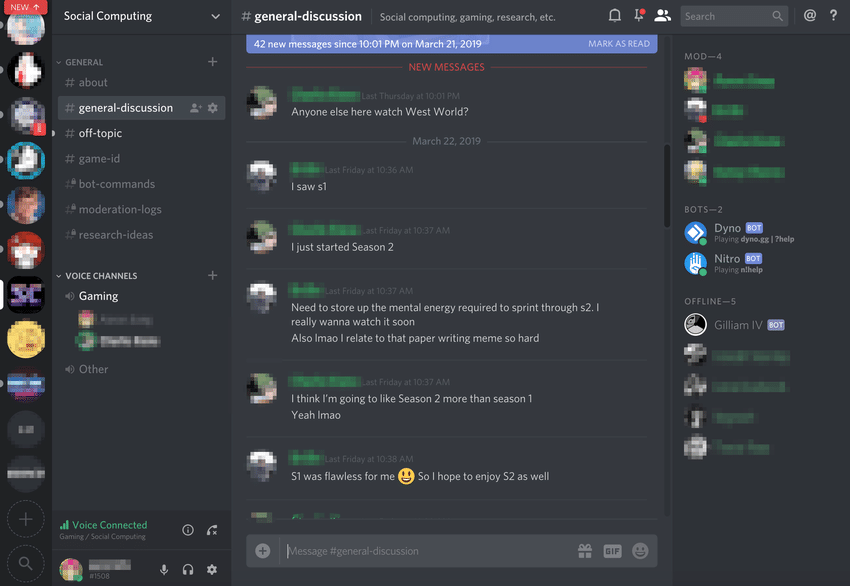
Друга функция, която откроява Messenger, са неговите възможности за чатбот. Чатботовете са компютърни програми, които използват изкуствен интелект, за да взаимодействат с потребителите по разговорен начин. Те могат да се използват за широк спектър от задачи, като поръчка на храна, пътуване с резервации или дори получаване на подкрепа за психично здраве. Това улеснява предприятията и организациите да предоставят обслужване и поддръжка на клиентите, както и за хората да имат достъп до информация и услуги, без да е необходимо човешко взаимодействие.

Messenger също има редица опции за персонализиране, които позволяват на потребителите да персонализират своите чатове. Потребителите могат да променят цвета на чата, да добавят емоджи и GIF и дори да играят игри с приятели директно в приложението. Тези функции правят Messenger забавна и ангажираща платформа за комуникация и връзка.

### 1.4.2. Discord

Discord е популярно чат приложение, което придоби огромна популярност сред геймърите, артистите и различните общности от началото си през 2015 г. Това е универсална платформа, която предлага широка гама от функции за комуникация, включително гласов и видео чат, споделяне на екрани и директни съобщения. Това, с което Discord се откроява от други подобни приложения като Viber и Messenger, е фокусът му върху създаването на платформа, която се грижи за нуждите на своите потребители.

Можем да видим на фигура 1-2 как изглежда потребителският интерфейс на Discord:



Фигура 1-2: Потребителски интерфейс на приложението Discord

Един от важните аспекти на Discord е способността му да създава сървъри, които са като виртуални пространства, където потребителите могат да комуникират помежду си въз основа на споделени интереси. Потребителите могат да създават и персонализират сървъри, да изберат тема на сървъра и да настроят различни канали за комуникация, които могат да бъдат или текстови, или аудио канали. Всеки сървър може да побере до 250 000 членове, което го прави идеален за общности от всякакъв размер. Потребителите могат също така да създават частни сървъри, които могат да бъдат достъпни само с покана, осигурявайки сигурно пространство за комуникация.

Друга функция, която прави Discord да изпъква сред останалите приложение, е интеграцията му с други приложения и услуги, включително Spotify, YouTube и Twitch. Потребителите могат да свържат своите акаунти и да споделят музика, видеоклипове и друго съдържание директно на Discord. Тази интеграция прави приложението идеална платформа за геймърите, които искат да споделят своите игри с други потребители в реално време.

Както Messenger, Discord също предлага и широк спектър от опции за персонализиране. Потребителите могат да създават персонализирани емоджита, аватари и фонове и дори да добавят персонализирани ботове за автоматизиране на определени задачи. Това ниво на персонализиране го прави по-забавно и ангажиращо за потребителите и им позволява да изразят себе си.

Едно от най-важните предимства на Discord пред другите чат приложения е фокусът му върху поверителността и сигурността. Discord използва end-to-end криптиране, като гарантира, че съобщенията и медиите са защитени от хакери. Платформата също така позволява на потребителите да контролират настройките си за поверителност, включително кой може да види техния онлайн статус и кой може да им изпраща директни съобщения.

### 1.4.3. Viber

Viber е популярно чат приложение, което се появява за първи път през 2010 г. от израелската компания Viber Media. През годините тя придобива значителна потребителска база и се превърна в едно от най-използваните приложения за изпращане на съобщения в света. Viber позволява на потребителите да правят гласови и видео обаждания, да изпращат текстови съобщения, да споделят файлове и да участват в групови чатове. Предлага се на всички основни платформи, включително iOS, Android и Windows.

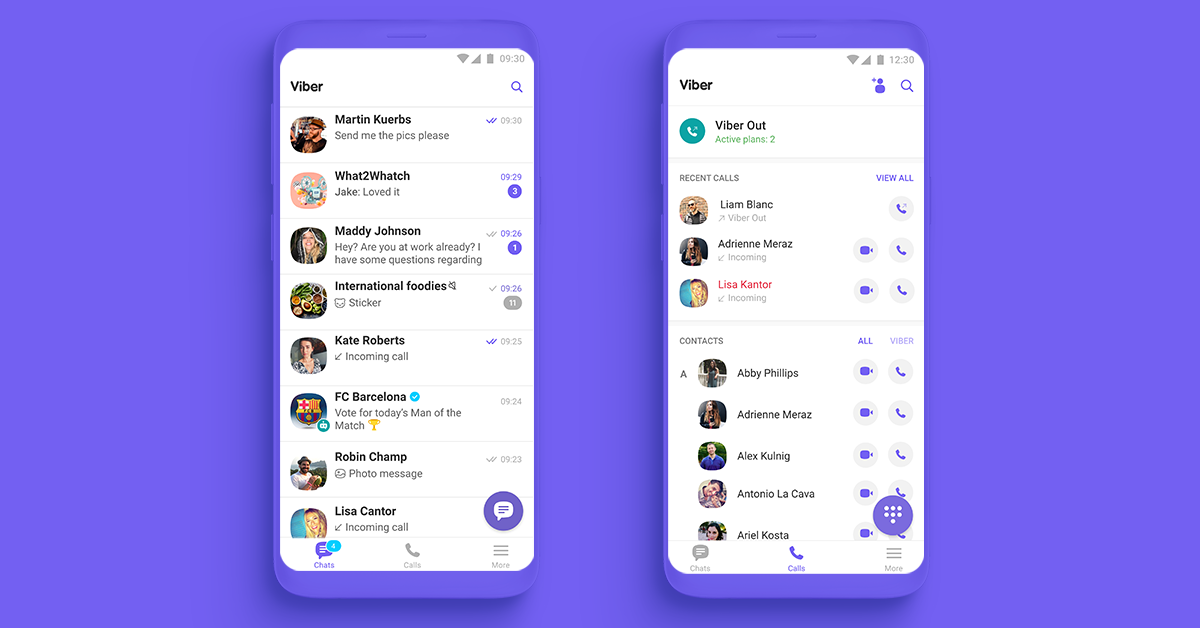
Една от отличителните характеристики на Viber е неговият силен фокус върху поверителността и сигурността. Всички съобщения и обаждания на платформата използват end-to-end криптиране, като се гарантира, че само предвиденият получател може да получи достъп до тях. Освен това потребителите имат възможност да задават таймери за самоунищожаване на съобщения, което гарантира, че те се изтриват автоматично след определен период. Тази функция осигурява на потребителите по-високо ниво на контрол върху техните разговори и помага да се защити тяхната поверителност.

Друга ключова характеристика на Viber е способността да се правят безплатни гласови и видео разговори към други потребители на Viber, независимо къде се намират в света. Тази функция е особено полезна за тези, които имат приятели и членове на тяхното семейство, живеещи в чужбина, тъй като им позволява да поддържат връзка, без да се налага да плащат скъпите международни такси за обаждане. В допълнение към това, Viber предлага и нискотарифни повиквания към хора, които не използват приложението, и това го прави привлекателна опция за тези, които искат да останат свързани без да плащат допълнително.

Както и при гореспоменатите приложения, потребителите на Viber могат да изпращат стикери, GIF и друго забавно дигитално съдържание на своите приятели и членове на семейството. Също така Viber има функция, наречена Viber Communities, която позволява на потребителите да се присъединяват към групи въз основа на общи интереси.

И накрая, Viber е невероятно лесен за използване. Потребителският интерфейс е интуитивен и удобен за потребителя, което улеснява дори и начинаещите в технологиите да започнат да го използват. Независимо дали искате да направите гласово повикване, да изпратите текстово съобщение или да споделите файл, всички функции са лесни за намиране и използване.

Във фигура 1-3 може да се види как изглежда потребителският интерфейс на Viber:



Фигура 1-3: Потребителски интерфейс на Viber

# Глава 2. Същност на приложението

След като бе разгледан в детайл проблемът както и неговите работещите решения, можем да преминем към заданието на софтуера, както и разглеждане на използваните технологии и библиотеки и компонентите в приложението.

## 2.1. ChatApp

ChatApp е уеб приложение (или по-точно single-page application) за изпращане на съобщения. Неговата главна цел както при всички чат приложения е да се изпращат съобщения, като потребителите могат да общуват помежду си по два начина: в частни (private) чат стаи, до които само двама потребители имат достъп до прочитането и изпращането на съобщения, или публични чат стаи, до които всички имат достъп.

## 2.2. Използвани технологии

За разработката на ChatApp трябваше да бъдат избрани най-подходящите технологии и езици, с които да постигнем максимално добър резултат. Поради факта, че ChatApp е уеб приложение, става ясно, че то ще бъде разработено с класическите езици за създаване на уеб сайтове или уеб приложения, а именно HTML, CSS и JavaScript. За улеснение, по-добро организиране и по-добра поддръжка на приложението са използвани front-end библиотеката React JS за изготвянето на потребителския интерфейс и back-end service Firebase за направата на сървърната част.

### 2.2.1. React JS

React JS е една от най-популярните JavaScript библиотеки, която се използва широко за изграждане на потребителски интерфейси. Създаден е от Facebook и сега се поддържа от общност от разработчици. React JS предоставя няколко предимства за разработване на уеб приложения.

Едно от основните предимства на React JS е неговата компонентно-базирана архитектура, която позволява на разработчиците да създават UI компоненти за многократна употреба. Това означава, че разработчиците могат да създадат библиотека от компоненти, които могат да се използват в цялото приложение, спестявайки време и усилия в разработката. Компонентите също са по-лесни за тестване и поддръжка, тъй като са изолирани от други части на приложението и могат да се актуализират независимо.

React JS също използва виртуален DOM, който е леко представяне на действителния DOM. Виртуалният DOM позволява ефективни актуализации на потребителския интерфейс, без да е необходимо да се манипулира действителният DOM, което може да бъде бавно и изискващо много ресурси. React JS актуализира само частите от потребителския интерфейс, които трябва да бъдат променени, което може значително да подобри производителността на приложението.

Друго предимство на React JS е способността му да работи с други библиотеки и софтуерни рамки (frameworks). React JS може да се използва с други библиотеки като Redux за управление на състоянието, а също така може да се интегрира с други софтуерни рамки като Angular или Vue.js. Това прави React JS гъвкава опция, която може да се адаптира към различни нужди на разработката.

React JS също има голяма и активна общност от разработчици, което означава, че има много налични ресурси и инструменти за подпомагане на разработката. Има множество библиотеки и пакети, налични за React JS, които могат да разширят функционалността му и да улеснят разработката.

Освен това React JS има добра поддръжка за SEO (оптимизация за търсачки), тъй като може да визуализира страници от страна на сървъра. Това означава, че търсачките могат лесно да обхождат и индексират съдържание, което може да подобри видимостта на приложението в резултатите от търсенето.

Също така се наблюдава ускорение на търсенето на уеб приложения заради пандемията от COVID-19 поради необходимостта от социално дистанциране и дистанционна работа. React JS е много подходящ за изграждане на уеб приложения, които могат да бъдат достъпни отвсякъде, което го прави ценен инструмент за разработчици, които трябва да създават приложения, които могат да се използват от разстояние.

### 2.2.2. Firebase

Firebase е мобилна и уеб платформа за разработка на сървърна част, която предоставя набор от инструменти и услуги, които разработчиците могат да използват за бързо и лесно създаване на висококачествени приложения. Firebase е разработен от Google и е популярен избор за разработчиците на приложения поради многото си предимства.

Едно от основните предимства на използването на Firebase е, че предоставя набор от инструменти и услуги, които улесняват създаването на приложения. Firebase предоставя платформа backend-as-a-service (BaaS), която разработчиците могат да използват за бързо изграждане и внедряване на своите приложения, без да е необходимо да се притесняват за изграждането и управлението на собствена сървърна инфраструктура.

Firebase също така предоставя набор от функции, които улесняват разработването на приложения, които са бързи и отзивчиви. Те включват база данни в реално време, която позволява на разработчиците да създават приложения, които се актуализират в реално време, и съхранение в облак, което улеснява съхраняването и достъпа до данни в облака.

Друго предимство на използването на Firebase е, че предоставя набор от инструменти за анализ, които разработчиците могат да използват, за да разберат как се използват техните приложения. Firebase предоставя анализи в реално време, които могат да помогнат на разработчиците да разберат как се използват техните приложения, както и потребителски анализи, които могат да помогнат на разработчиците да разберат поведението на своите потребители.

Firebase също така предоставя набор от функции за удостоверяване и сигурност, които улесняват изграждането на сигурни приложения. Firebase предоставя услуги за удостоверяване, които позволяват на разработчиците лесно да удостоверяват потребителите и да управляват потребителски акаунти, както и правила за сигурност, които позволяват на разработчиците да определят контроли за достъп за своите приложения.

Firebase предоставя набор от интеграции, които улесняват интегрирането с други услуги и платформи. Firebase се интегрира с набор от услуги на трети страни, включително Google Cloud Platform, Google Analytics и AdMob.

В допълнение към тези предимства, Firebase също е популярен избор за разработчиците на приложения поради своята лекота на използване и гъвкавост. Firebase е проектиран да бъде лесен за използване дори за разработчици, които са начинаещи в изграждането на приложения. Firebase също е доста гъвкав, което означава, че може да се използва за създаване на приложения от всякакъв размер, от малки мобилни приложения до големи корпоративни приложения.

## 2.3. Използвани допълнителни библиотеки

### 2.3.1. React Router

React Router е популярна библиотека за маршрутизиране за React приложения. Той позволява на разработчиците лесно да дефинират и управляват маршрути от страна на клиента в техните приложения, осигурявайки безпроблемно изживяване за потребителите, докато навигират между различни изгледи и компоненти.

Една от ключовите характеристики на React Router е способността му да поддържа nested routes, което може да помогне за организирането и структурирането на по-големи приложения. Разработчиците могат да дефинират nested routes в рамките на родителски маршрут, позволявайки по-модулна и лесно поддържана кодова база.

Друго предимство на React Router е способността му да обработва различни видове преходи на маршрути, като анимации и пренасочвания. Това може да помогне за създаването на по-изчистено и интуитивно потребителско изживяване, докато потребителите се движат между различни части на приложението.

React Router също така предоставя гъвкави и мощни опции за конфигуриране на маршрутизацията. Разработчиците могат да използват различни техники за маршрутизиране, като динамично маршрутизиране, което позволява предаване на параметри към компонентите на маршрута, и декларативно маршрутизиране, което позволява дефиниране на маршрути по по-сбит и изразителен начин.

В допълнение към тези функции, React Router също се интегрира добре с други React библиотеки и инструменти. Например, може лесно да се използва с Redux за управление на състоянието или с React Native за изграждане на мобилни приложения.

### 2.3.2. React Firebase Hooks

React Firebase Hooks е библиотека, която позволява на разработчиците на React лесно да интегрират Firebase в своите приложения с помощта на React Hooks. React Firebase Hooks опростява процеса на свързване към Firebase, като предоставя предварително изградени кукички, които разработчиците могат да използват за достъп до услугите на Firebase в техните React компоненти. React Firebase Hooks предоставя редица hooks, които могат да се използват за взаимодействие с различни услуги на Firebase.

React Firebase Hooks е мощен инструмент за разработчиците на React, защото опростява процеса на интегриране на Firebase в React приложения. Той позволява на разработчиците лесен достъп до услугите на Firebase, използвайки познатия синтаксис на React Hooks, което улеснява интегрирането на Firebase в съществуващи кодови бази на React. Освен това, тъй като е базиран на React Hooks, той позволява лесно управление на състоянието и елиминира необходимостта от сложни методи на жизнения цикъл.

Популярността на React Firebase Hooks нарасна значително през последните няколко години, отчасти поради лекотата на използване и популярността на React и Firebase като инструменти за разработка. Библиотеката се поддържа активно и се актуализира и има голяма общност от разработчици, които допринасят за библиотеката и осигуряват поддръжка за други разработчици.

## 2.4. Компоненти

Поради компонентно-базираната архитектура на React JS, кодът е разделен на отделни компоненти. За по-добра организация и по-лесна поддръжка компонентите са разделени в няколко папки като тези папки имат различно предназначение.

Във фигура 2-1 е показана файловата структура на всички компоненти в приложението:



Фигура 2-1: Файлова структура на компонентите

### 2.4.1. Главни компоненти

Това са компонентите, които не принадлежат на нито една от трите горепосочени папки (auth / chat / map-components). На тях се изобразяват останалите компоненти и те контролират какво ще бъде показано на екрана в зависимост от това дали потребителят е оторизиран.

Тези компоненти са:

* **Home.js** - Това е главният компонент. На него се изобразяват останалите компоненти.
* **UnauthenticatedApp.js** - На този компонент се показва съдържанието, за което не е нужно удостоверяване на потребителя.
* **AuthenticatedApp.js** - На този компонент се показва съдържанието, което може да бъде достъпено само от удостоверени потребители.

### 2.4.2. Auth компоненти

В папка **auth** се съдържат всички компоненти, които отговарят за автентикацията на потребителите. Благодарение на тях те лесно могат както да създават нов профил, така и да влизат в него и съответно да излизат от него.

Тези компоненти са:

* **Login.js -** Този компонент служи за влизането във вече съществуващ профил. Представлява диалогов прозорец, в която се съдържа форма, чрез която потребителите могат да влязат по един от два начина: с имейл и парола или с Google профил.
* **SignUp.js -** Този компонент служи за създаването на нов профил. Както *Login.js*, този компонент представлява диалогов прозорец, в който се съдържа форма, чрез която потребителите могат да създадат нов профил по един от два начина: с имейл и парола (като тук също се изискват и първо и последно име, както и профилна снимка) или с Google профил.
* **SignOut.js -** Този компонент служи за излизането на вече автентикиран потребител от своя профил. За разлика от останалите два компонента, той е един бутон, който при кликване се излиза от профила.

### 2.4.3. Chat компоненти

В папка **chat** се съдържат всички компоненти, които отговарят за показването на всички чат стаи, показването на всички съобщения в съответните чат стаи и изпращането на съобщения.

Тези компоненти са:

* **Chatroom.js -** Този компонент служи за показването на дадена чат стая. В нея може както да се показват всички изпратени съобщения от тази стая, така и могат да се изпращат съобщения и тези съобщения да бъдат виждани от определен потребител (или в зависимост от типа стая те могат да бъдат виждани и от всички).
* **ChatroomCreateDialog.js -** Този компонент служи за създаване на скрита чат стая. Представлява диалогов прозорец, в който потребителят може да създаде нова стая като изпрати съобщение към даден потребител като кликне върху него.
* **PrivateChatRoomPage.js -** Този компонент служи за показване на всички скрити чат стаи, достъпни до потребителя. Ако такива не съществуват, на потребителят му се показват всички останали потребители и когато кликне върху тях, може да им изпрати съобщение чрез диалоговия прозорец от *ChatroomCreateDialog.js.*
* **PublicGroupPage.js -** Този компонент служи за показване на всички съществуващи публични чат стаи (още познати като *групи*). Чрез кликване на една от тези групи, потребителят може да влезе в тях. В този компонент също може да се създават нови групи.

### 2.4.4. Map компоненти

В папка **map-components** се съдържат компонентите, в които ще се показва нужната информация. Тези компоненти служат за шаблони, чрез които ще бъдат показвани данните.

Тези компоненти са:

* **User.js -** Този компонент служи за показване на информацията на останалите потребители.
* **Message.js -** Този компонент служи за показване на съобщенията.
* **PrivateRoom.js -** Този компонент служи за показването на скритите чат стаи.
* **Group.js -** Този компонент служи за показването на различните групи.

# Глава 3. Проектиране на приложението

В тази глава ще бъде разгледана имплементацията на проекта, неговата сървърна и клиентска част в детайл.

## 3.1. Firebase

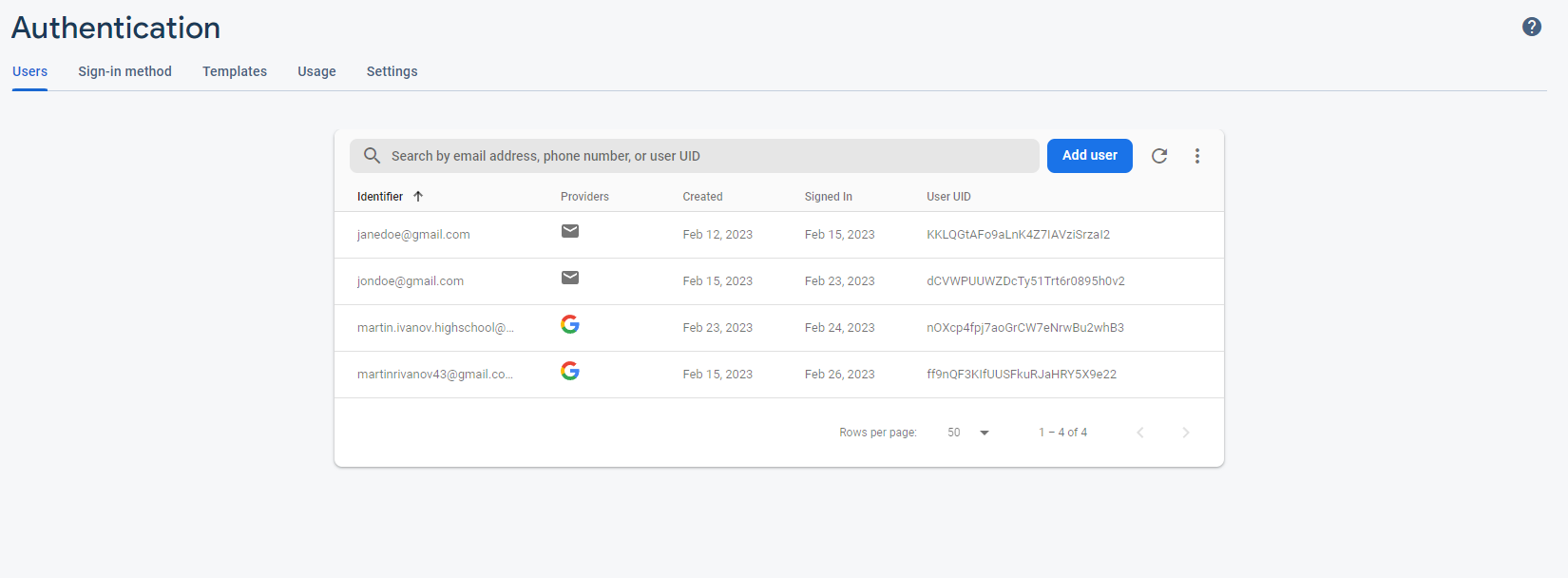
За сървърната част на приложението се работи с back-end service Firebase. Чрез него бе настроена автентикацията на потребителите, хранилището, в което ще се пазят профилните снимки на потребителите, и базата данни.

### 3.1.1. Автентикация на потребители

Автентикацията на потребители (или още познато като удостоверяване на потребители) представлява процесът, чрез който се потвърждава идентичността на потребителя. Благодарение на това приложението бива по-защитено, като само оторизирани потребители могат да имат достъп до определено съдържание.

Автентикацията в приложението става чрез един от два начина: имейл и парола или Google акаунт. Всеки създаден профил си има уникален потребителски идентификационен номер (или накратко записано uid) и чрез проверка на този номер се извършва оторизацията.

Във фигура 3-1 са показани потребителите, записани в базата данни:



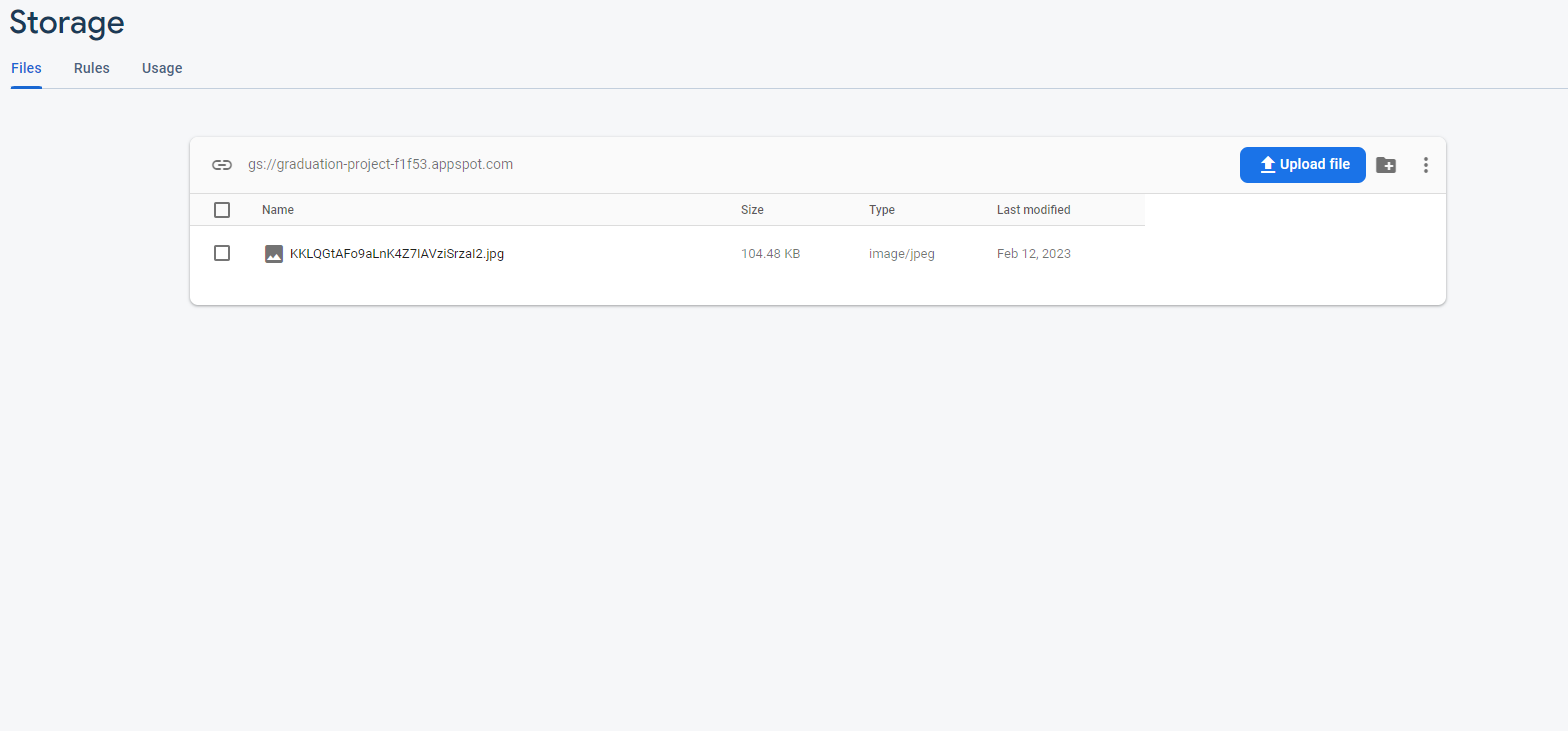
Фигура 3-1: Потребителите на нашето приложение

### 3.1.2. Хранилище

В хранилището на Firebase могат да се пазят множество файлове, които не биха могли да бъдат запазени в обикновена база данни. Предимствата на хранилището са във бързото четене и записване на файлове, което за разлика от много други хранилища бива безплатно.

В хранилището на проекта се пазят профилните снимки на потребителите, чийто метод за автентикация е чрез имейл и парола, тъй като при потребителите, регистрирали се чрез Google акаунт, се използва профилната снимка от този акаунт. Когато ни трябва достъп до една от тези снимки, тя може да бъде достъпена през генериран от Firebase DownloadURL.

Във фигура 3-2 се виждат записаните файлове в хранилището:

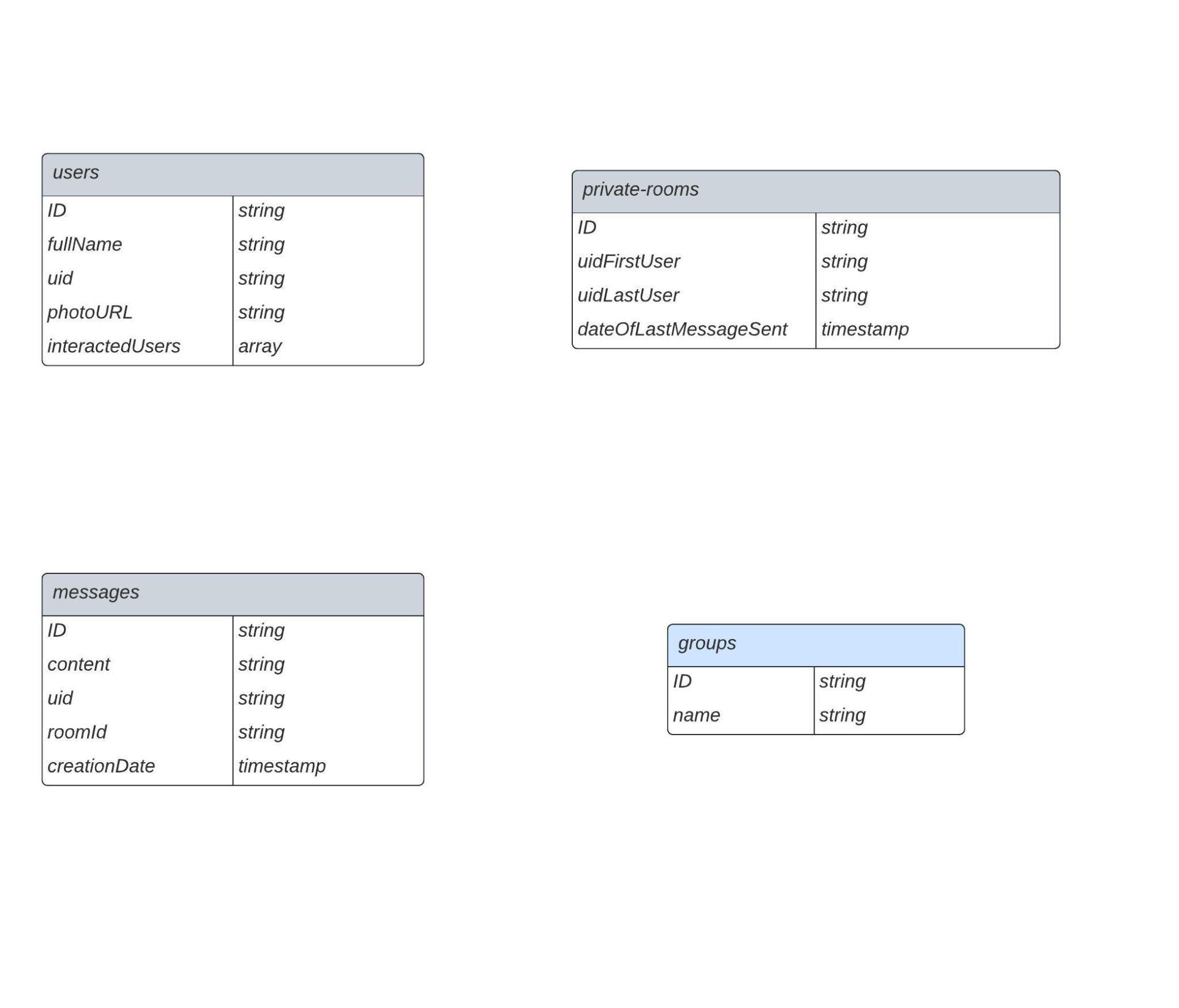


Фигура 3-2: Файловете в хранилището на проекта

### 3.1.3. База данни

За проекта е използвана Firestore Cloud база данни. Представлява NoSQL тип база данни, в която се работи с документи, които се намират в определени колекции, за разлика от традиционните SQL бази данни, където информацията бива записвана на отделни редове в таблица.

В базата данни на проекта са направени 4 колекции. Във фигура 3-3 е показан модел на всяка една от тези колекции и съответно техните полета.



Фигура 3-3: Модел на колекциите в базата данни

#### 3.1.3.1. users

В колекция users се пази необходимата информация за всички потребители. Целта на тази колекция е по-лесен достъп до потребителската информация, тъй като Firebase дава информация само за профила, в който сме влезли.

Полетата в тази колекция са:

* ID - string
* fullName (пълното име на потребителя) - string
* uid (потребителското ID) - string
* photoURL (адрес към профилна снимка) - string
* interactedUsers (потребители, с които съществува скрита чат стая) - array

#### 3.1.3.2. groups

В колекция groups се пазят всички съществуващи публични чат групи. Полетата в тази колекция са:

* ID - string
* name (име на групата) - string

#### 3.1.3.3. private-rooms

В тази колекция се пазят скритите чат стаи. Тези стаи биват достъпни само за двама потребители.

Полетата в тази колекция са:

* ID - string
* uidFirstUser (UID на първия потребител) - string
* uidSecondUser (UID на втория потребител) - string
* dateOfLastMessageSent (датата на последното изпратено съобщение в чат стаята) - timestamp

#### 3.1.3.4. messages

В тази колекция се пазят всички съобщения. Те биват достъпни само от една чат стая и от потребителите, които имат достъп до тази чат стая.

Полетата в тази колекция са:

* ID - string
* content (съдържание на съобщението) - string
* uid (UID на изпращача) - string
* roomId (ID на стаята, в която е изпратено съобщението) - string
* creationDate (датата, на която е изпратено съобщението) - timestamp

## 3.2. Кодова имплементация

### 3.2.1. Настройване на връзката между сървърната и клиентската част

Преди да се премине към самото настройване, първо трябва да се инициализира Firebase проект. Това става чрез командата **firebase init**. По този начин се улесняват много от последващите задачи като внедряване (deployment) или пускане на локален сървър (serving).

За да се настрои връзката между сървърната и клиентската част в приложението, бе направен отделен файл **setup.js**. В него се инициализира Firebase, за да може да работи в приложението, настройват се услугите auth, firestore и storage, чрез които ще се получава или модифицира информация относно потребители, хранилище и база данни, и се пазят референции към колекциите от базата данни.

Във фигура 3-4 може да се види настройването на базата данни в приложението:

|  |
| --- |
| const firebaseConfig = {  apiKey: "AIzaSyAIUqTResh55cqf-VLca3YVI\_f9RuhVHz4",  authDomain: "graduation-project-f1f53.firebaseapp.com",  projectId: "graduation-project-f1f53",  storageBucket: "graduation-project-f1f53.appspot.com",  messagingSenderId: "1052485363656",  appId: "1:1052485363656:web:0b8ab8eae92aba73e702fd" };  const firebaseApp = firebase.initializeApp(firebaseConfig);  const auth = firebase.auth(); const firestore = firebase.firestore(); const storage = getStorage(firebaseApp);  const usersRef = firestore.collection('users'); const messagesRef = firestore.collection('messages'); const privateRoomsRef = firestore.collection('private-rooms'); const groupsRef = firestore.collection('groups');  export {auth, firestore, storage, privateRoomsRef, usersRef, messagesRef, groupsRef}; |

Фигура 3-4: Сорс кодът на setup.js

### 3.2.2. Потребителска автентикация и оторизация

На първо време съдържанието на приложението е необходимо да бъде разделено в зависимост от това дали даден потребител е удостоверен. Точно затова и съществуват компонентите **AuthenticatedApp** и **UnauthenticatedApp**. За да можем обаче да контролираме кое от двете да се показва, проверяваме чрез **useAuthState()** дали има удостоверен потребител. Ако такъв има и ако този потребител има име (тази проверка се случва поради това, че при създаване на профил приложението трябва да изчака да се обновят данните за цяло име и профилна снимка), се показва съдържанието, предназначено за удостоверени потребители. В противен случай, неавтентикираните потребители могат да виждат само формите за регистрация и влизане в съществуващ профил.

Във фигура 3-5 е представен сорс кодът за проверка на потребителската оторизация и показване на съответното съдържание:

|  |
| --- |
| function Home(){  const [user] = useAuthState(auth);   return user && user.displayName ? <AuthenticatedApp currentUser={user} /> : <UnauthenticatedApp />  } |

Фигура 3-5: Сорс код от компонента Home

За създаването на нов профил е отговорен компонентът **SignUp**. Представлява диалогов прозорец, който съдържа регистрационна форма и бутон, с който може да се регистрира чрез Google акаунт. Ако потребителят реши да се регистрира чрез имейл и парола, при submit-ване на формата се изпълнява функцията **handleFormSubmit()**, чрез който се създава нов профил.

На фигура 3-6 може да се види сорс кода на тази функция:

|  |
| --- |
| const handleFormSubmit = (e) => {  e.preventDefault();  createUserWithEmailAndPassword(email, password)  .then(async (userCredential) => {  if(error){  alert(error.message);  }  else{  let imageFile = imageUpload ? userCredential.user.uid + '.jpg' : 'default-avatar.png';   const storageRef = ref(storage, imageFile);    if (imageUpload) {  await uploadFile(storageRef, imageUpload);  }   const photoURL = await getDownloadURL(storageRef);   await setDoc(doc(firestore, 'users', userCredential.user.uid), {  fullName: `${firstName} ${lastName}`,  photoURL,  uid: userCredential.user.uid,  interactedUsers: []  });   updateProfile({displayName: `${firstName} ${lastName}`, photoURL})  .then(() => {  auth.signOut()  })  .then(() => {  signInWithEmailAndPassword(email, password);  });  }  });  } |

Фигура 3-6: Сорс кодът на функцията handleFormSubmit()

Първо се създава нов профил чрез функцията **createUserWithEmailAndPassword()**, който бива инициализиран от вградения hook **useCreateUserWithEmailAndPassword()** от библиотеката React Firebase Hooks. След това се създава референция към профилната снимка на потребителя, качва се в хранилището и се взима нейното DownloadURL. Ако такава снимка не е зададена, тогава се използва default профилна снимка. След това се добавя се потребителя в базата данни, обновява се информацията за пълно име и профилна снимка и след това влизаме в профила наново с функцията **signInWithEmailAndPassword()**, за да може приложението да работи с обновената информация. В случай на грешка обаче нов потребител не се създава и клиентът бива предупреден за допуснатата от него грешка.

Ако обаче потребителят реши да се регистрира с Google акаунт, при кликване на бутона за регистрация с Google се изпълнява функцията **handleGoogleButtonClick()**.

На фигура 3-7 може да се види сорс кода на тази функция:

|  |
| --- |
| const handleGoogleButtonClick = () => {  signInWithGoogle()  .then(async (userCredential) => {  await setDoc(doc(firestore, 'users', userCredential.user.uid), {  fullName: userCredential.user.displayName,  photoURL: userCredential.user.photoURL,  uid: userCredential.user.uid,  interactedUsers: []  });  }) } |

Фигура 3-7: Сорс кодът на функцията handleGoogleButtonClick()

Първо се изпълнява функцията **signInWithGoogle()**, която бива инициализирана от вградения hook **useSignInWithGoogle()**. След това информацията за потребителя бива добавена в базата данни.

Влизането във вече съществуващ профил става през компонента **Login**. Той също показва диалогов прозорец със форма за влизане с имейл и бутон за влизане с Google акаунт. Влизането съответно става чрез вече споменатите функции **signInWithEmailAndPassword()** и **signInWithGoogle()**.

### 3.2.3. Показване на чат стаите

След автентикирането потребителят има достъп до това да вижда и да създава нови чат стаи, както и да изпраща и вижда съобщенията в тези стаи. Скритите чат стаи и груповите стаи са отделени в два компонента, а именно **PrivateChatRoomPage** и **PublicGroupPage**. Благодарение на функциите **showPrivateChatRoomPage()** и **showPublicGroupPage()** може да се показва само един от тези два компонента като показва единия и скрива другия. По този начин се осигурява по-лесен достъп и по-добра организация. По подразбиране се отваря първо страницата със скритите чат стаи.

Във фигура 3-8 е показан сорс кодът на тези две функции:

|  |
| --- |
| useEffect(() => {  showPrivateChatRoomPage(); }, [privateChatRoomPage]);  const showPrivateChatRoomPage = () => {  publicGroupPage.current.style.display = 'none';  if (privateChatRoomPage.current) {  privateChatRoomPage.current.style.display = 'block';  } }  const showPublicGroupPage = () => {  publicGroupPage.current.style.display = 'block';  privateChatRoomPage.current.style.display = 'none'; } |

Фигура 3-8: Сорс код на функциите showPrivateChatRoomPage() и showPublicGroupPage()

В **PrivateChatRoomPage** се виждат както всички скрити чат стаи, които притежаваме, така и всички потребители, с които няма съществуваща чат стая. Ако потребителят няма съществуващи чат стаи, се изписва че не си е общувал с никого.

Във фигура 3-9 е сорс кодът на **PrivateChatRoomPage**, в който е показана имплементацията на визуализирането на чат стаите и потребителите:

|  |
| --- |
| const interactions = users && users.filter(u => u.uid === uid)[0].interactedUsers;  const getPrivateRoomData = () => {  let data = [];   privateRooms.forEach(pr => {  let id = pr.id;  let privateRoomData = pr.data();   let entry = {  id,  ...privateRoomData  };   data.push(entry);  });   let filteredData = data.filter(pr => pr.uidFirstUser === uid || pr.uidSecondUser === uid);  return filteredData; }  return (  <div>  {!loading &&   <main ref={refference}>  {  (privateRooms.docs.length > 0 && privateRooms.docs.map(doc => doc.data()).filter(pr => pr.uidFirstUser === uid || pr.uidSecondUser === uid).length > 0) ?   <ul>  {getPrivateRoomData().map((pr, index) => <PrivateRoom key={index} room={pr} uid={uid} />)}  </ul> :  <h3>You haven't texted anyone yet</h3>  }  <p>Click on any user to send them a message</p>  <div className="grid-container">  {(users && interactions) && users.filter(u => u.uid !== uid && !interactions.includes(u.uid)).map((u, index) => <User key={index} user={u} setUserId={setUserId} dialogRef={dialogRef} />)}  </div>  <ChatroomCreateDialog currentUserId={uid} secondUserId={userId} setUserId={setUserId} reference={dialogRef} />  </main>}  </div> ); |

Фигура 3-9: Сорс код за визуализацията на всички чат стаи и потребители от PrivateChatRoomPage

Създаването на нова скрита чат стая става чрез компонента **ChatroomCreateDialog**, който представлява текстово поле за изпращане на съобщение. По този начин е гарантирано, че няма да се направи празна чат стая.

Във фигура 3-10 е показана функцията **handleDialogSubmit()**, която отговаря за създаването на нова скрита чат група:

|  |
| --- |
| const handleDialogSubmit = async (e) => {  e.preventDefault();   if (message && message.trim()) {  const lastMessageTimestamp = firebase.firestore.FieldValue.serverTimestamp();   const docRef = await privateRoomsRef.add({  uidFirstUser: currentUserId,  uidSecondUser: secondUserId,  dateOfLastMessageSent: lastMessageTimestamp  });   messagesRef.add({  content: message,  uid: currentUserId,  creationDate: lastMessageTimestamp,  roomId: docRef.id  }).then(() => {  usersRef.doc(currentUserId).update({  interactedUsers: firebase.firestore.FieldValue.arrayUnion(secondUserId)  })  }).then(() => {  usersRef.doc(secondUserId).update({  interactedUsers: firebase.firestore.FieldValue.arrayUnion(currentUserId)  })  }).then(() => {  navigate(`/room/${docRef.id}`)  })  } } |

Фигура 3-10: Сорс кодът на функцията handleDialogSubmit()

Както се вижда по-горе, чат стая не може да бъде създадена, ако съобщението е празно. Също така при създаването ѝ съобщението се записва в базата данни, като взема ID-то на новосъздадената стая. След това се обновява **interactedUsers** за двамата потребители, участващи в стаята, и след това потребителя бива навигиран към линка на стаята.

В **PublicGroupPage** се показват всички групови чат стаи, които са достъпни до всеки потребител. Всеки от тях може както да чете, така и да пише съобщения и всеки може да създава групова чат стая.

Във фигура 3-11 е показан сорс кодът на компонента **PublicGroupPage:**

|  |
| --- |
| return (  <main ref={refference}>  {!loading &&  <div>  <h3>Groups</h3>  <p>Click on one of the groups to start chatting</p>    <div className="group-list">  <ul>  {groups && groups.docs.map((docs, index) => <Group key={index} groupDocs={docs} />)}  <li className="group-card hover-form">  <p>Add new group</p>  <div className="group-form">  <input placeholder="Name of the group" value={groupName} onChange={(e) => handleGroupNameInput(e.currentTarget.value)} />  <button onClick={() => addNewGroup()}>Add new group</button>  </div>  </li>  </ul>  </div>  </div>  }  </main> ); |

Фигура 3-11: Сорс кодът на компонента PublicGroupPage

### 3.2.4. Показване и изпращане на съобщения

При влизане в дадена чат стая, потребителите имат правото да изпращат и виждат съобщенията вътре. Изпращането на съобщенията става чрез функцията **sendMessage()**.

Във фигура 3-12 е показан сорс кода на функцията:

|  |
| --- |
| const sendMessage = (e) => {  e.preventDefault();   if (message && message.trim()) {  messagesRef.add({  content: message,  uid: auth.currentUser.uid,  creationDate: firebase.firestore.FieldValue.serverTimestamp(),  roomId: params.id  }).then(() => {  setMessage('');    if (location.pathname.split('/')[1] === 'room') {  privateRoomsRef.doc(params.id).update({  dateOfLastMessageSent: firebase.firestore.FieldValue.serverTimestamp()  })  }  });  } } |

Фигура 3-12: Сорс код на функцията sendMessage()

Първо се прави проверка дали съобщението е празно. Ако не е, съобщението се добавя в базата данни, заедно с това кой го е пратил, на коя дата и от коя стая. След това се проверява дали съобщението е изпратено от групова стая или от скрита стая. В случай, че е изпратено от скрита стая, се обновява датата на последното изпратено съобщение в колекция **private-rooms**. По този начин най-отгоре ще се показва стаята, чието последно съобщение е изпратено най-скоро.

Съобщенията биват показани благодарение на компонента **Message**. В него се подават данните от колекция **messages** и то показва от кого и какво е изпратено.

Във фигура 3-13 е показан сорс кодът на компонента:

|  |
| --- |
| const Message = (props) => {  const {content, uid} = props.message;  const messageClass = uid === auth.currentUser.uid ? 'sent' : 'received';  const [userPhotoURL, setUserPhotoURL] = useState('');   useEffect(() => {  (async() => {  const userDoc = await usersRef.doc(uid).get();  setUserPhotoURL(userDoc.data().photoURL);  }) ();  }, [])   return (  <div className={`message ${messageClass}`}>  <img className="message-photo" src={userPhotoURL} referrerPolicy="no-referrer"/>  <p>{content}</p>  </div>  ); } |

Фигура 3-13: Сорс код на компонента Message

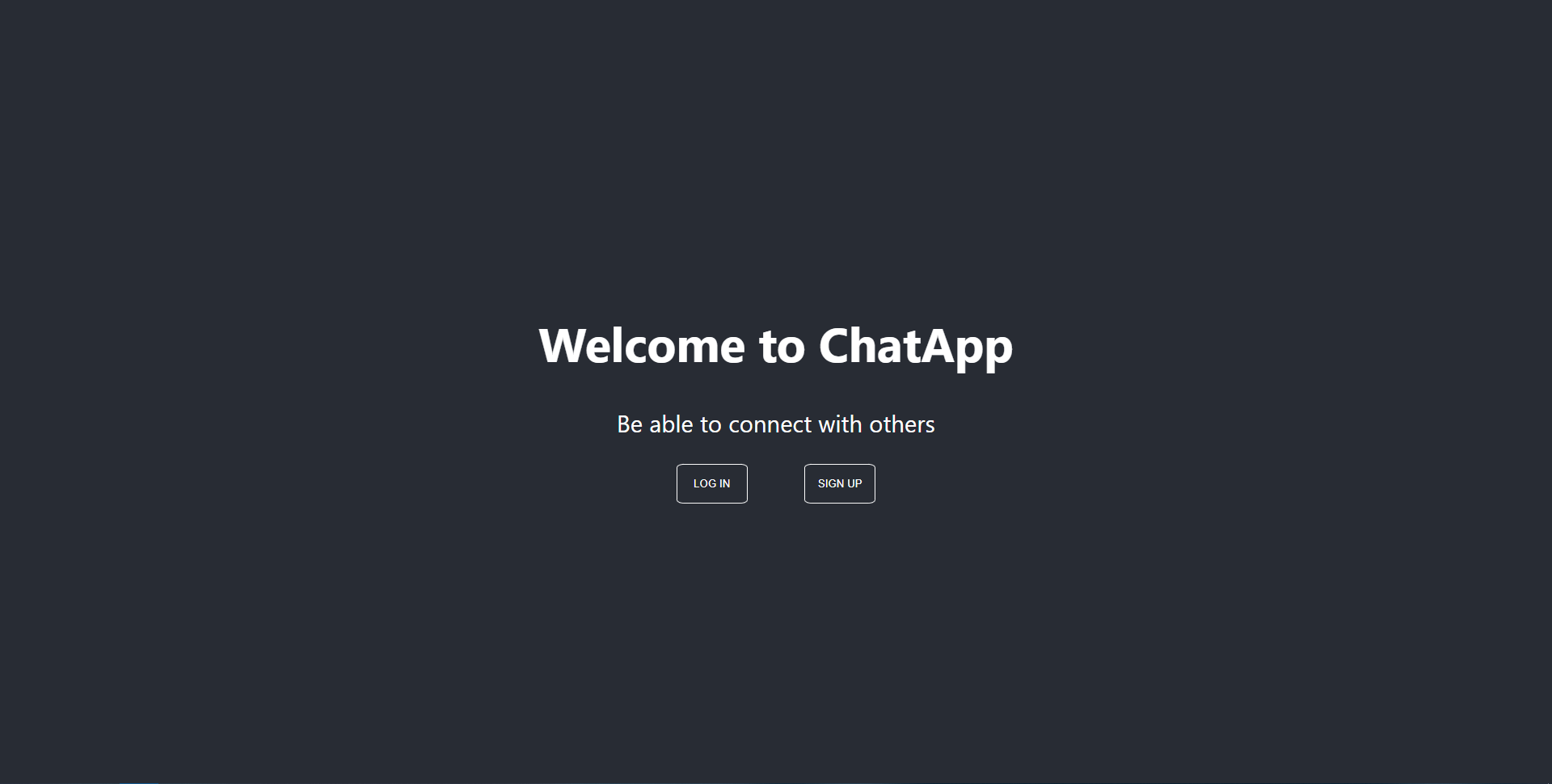
В компонента се прави проверка дали съобщението е изпратено от потребителя, в който сме влезли, или не. В зависимост от това му даваме клас **sent** или **received**. В useEffect() взимаме профилната снимка на потребителя, за да може да бъде показана заедно със съобщението.

# Глава 4. Ръководство за употреба на приложението

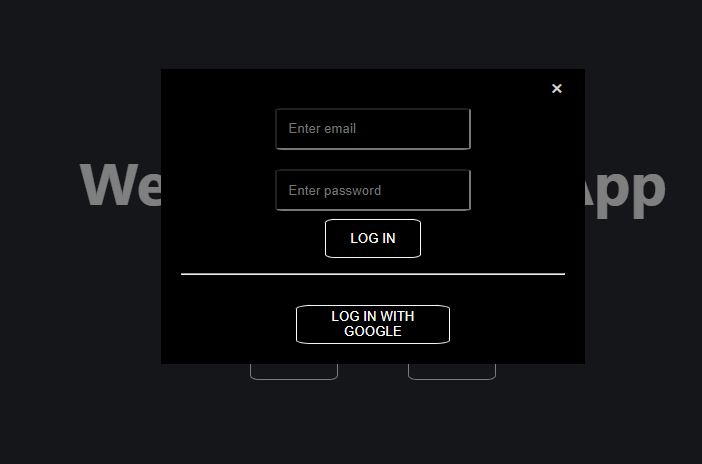
За стартиране на приложението е необходимо да се изпълнят следните стъпки:

1. Изтегляне на кода от GitHub (линк към repo-то може да бъде открит в Приложение) и NodeJS.
2. Инсталират се нужните библиотеки с командата **npm-install** в конзолата.
3. Стартира се приложението чрез командата **npm-start** в конзолата.

При влизане в приложението се отваря началната страница. В нея потребителят може да се регистрира или да влезе във вече съществуващ профил, както е показано на фигура 4-1 и фигура 4-2:



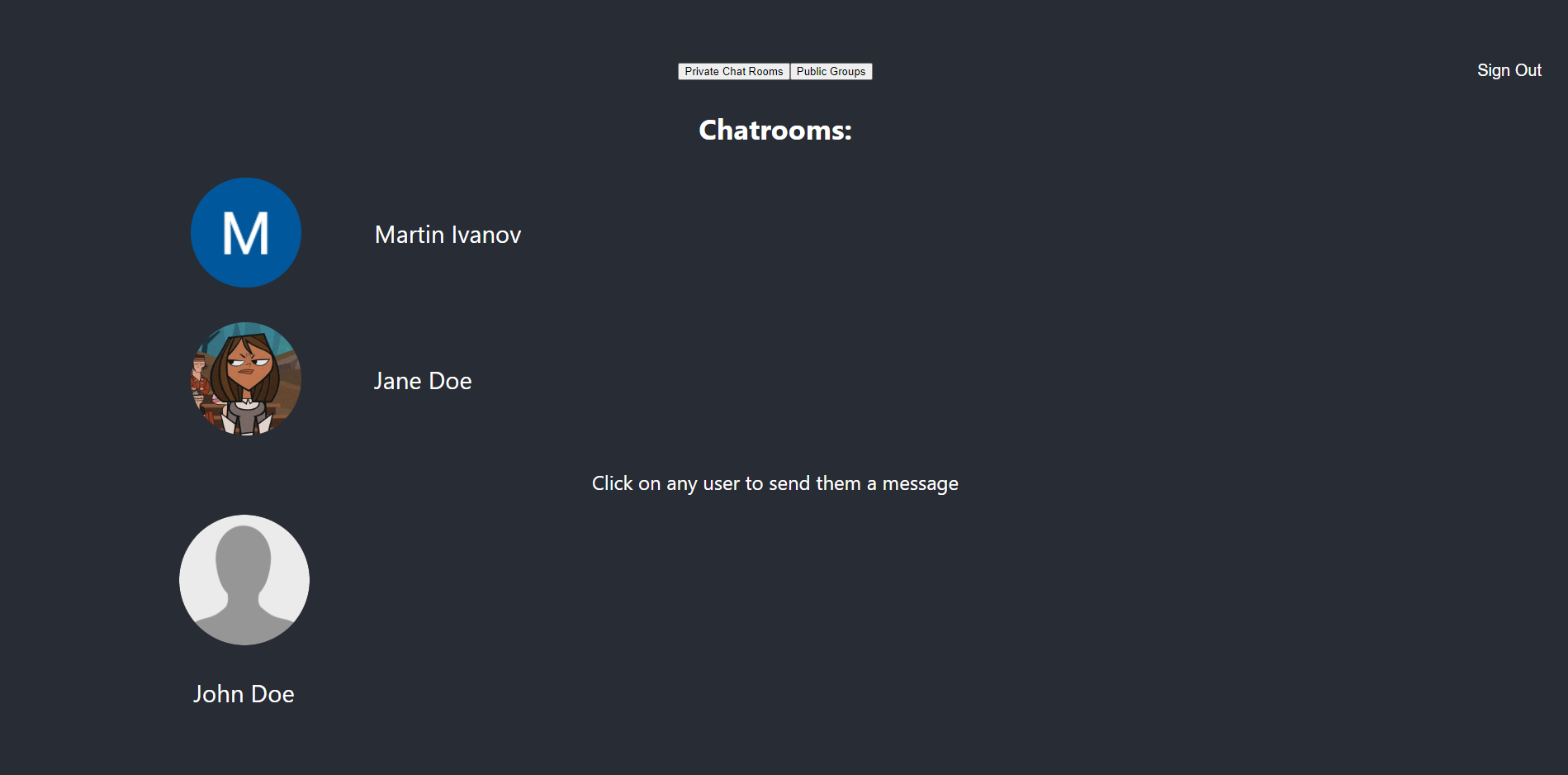
Фигура 4-1: Начална страница за неудостоверен потребител



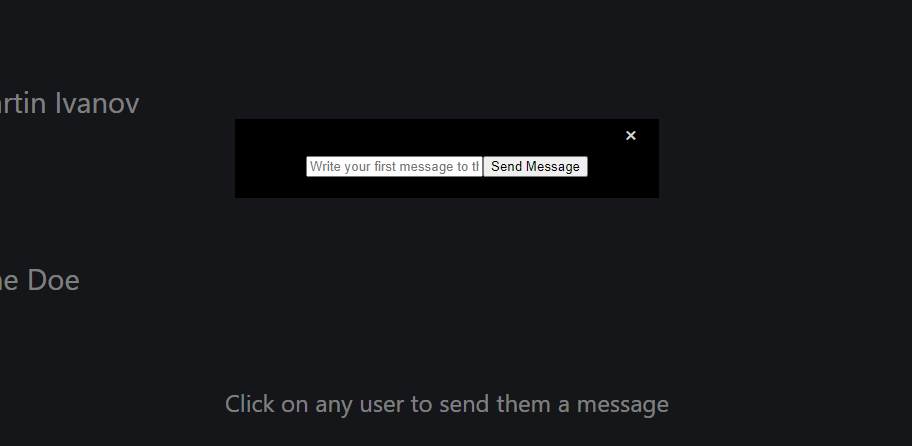
Фигура 4-2: Форма за влизане в профил

При влизане в своя профил, потребителят може да види своите скрити чат стаи, както и потребителите, с които такива стаи не съществуват. За да се създаде нова скрита стая, трябва да се кликне върху потребителят, с който искаме да бъде създадена стая. Отваря се диалогов прозорец, през който изпращаме съобщение.

На фигура 4-3 и фигура 4-4 са показани гореспоменатите неща:



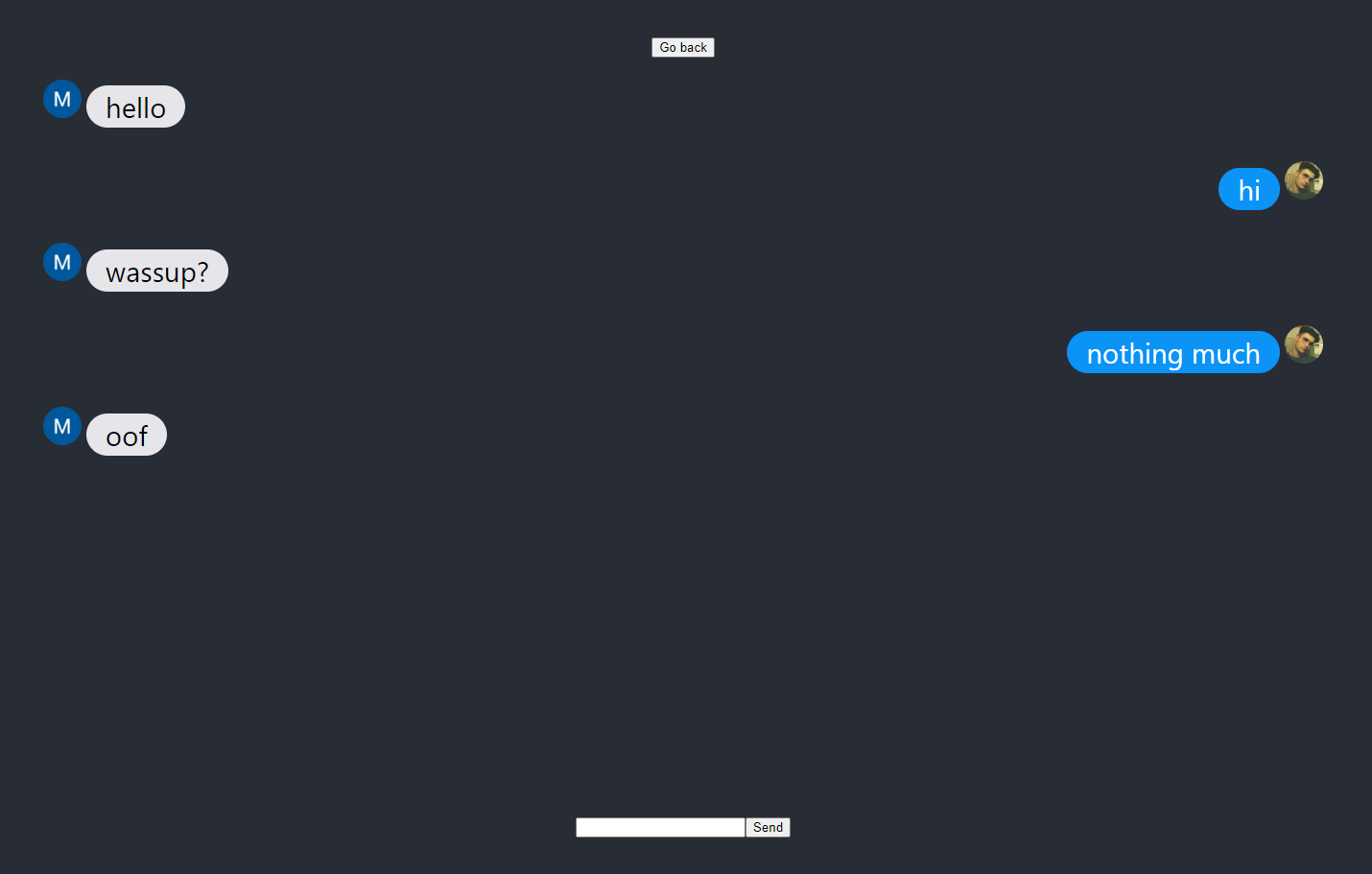
Фигура 4-3: Страницата със скритите чат стаи



Фигура 4-4: Диалогов прозорец за създаване на нова чат стая

При натискане на една от тези стаи, потребителят бива изпратен в съответната стая където може да вижда всички съобщения. Изпращане на съобщение става чрез въвеждане на съобщението в текстовото поле и след това кликване на бутона Send.

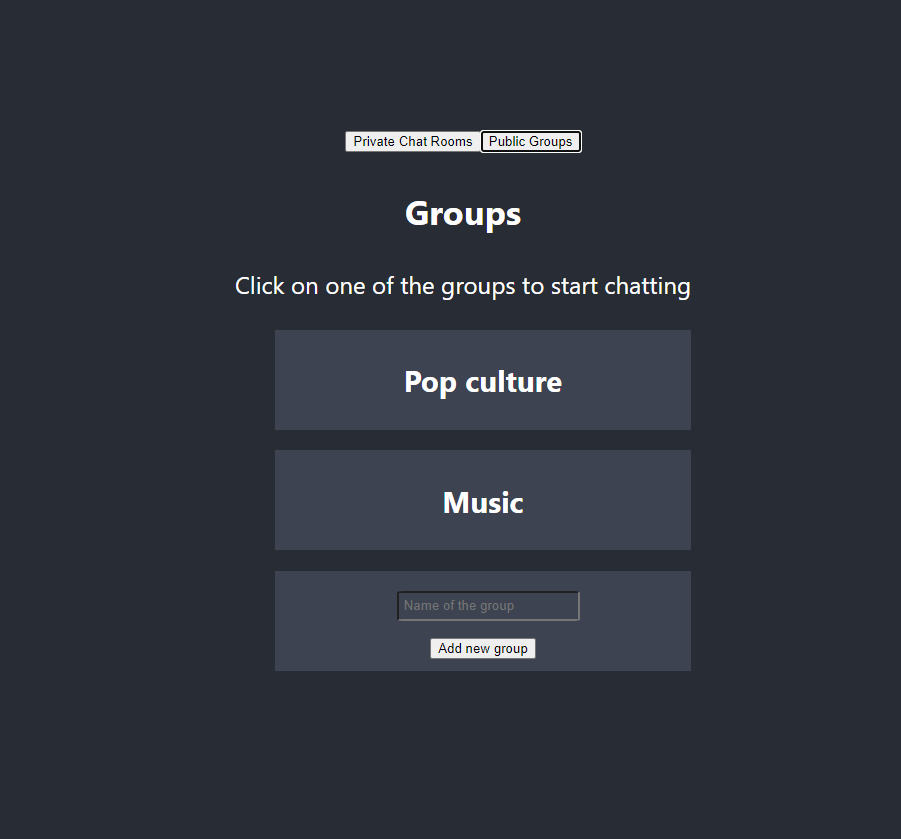
На фигура 4-5 е показана една от тези стаи:



Фигура 4-5: Чат стая

Натискайки бутона **Go Back**, потребителят се връща назад в началната страница на приложението. Ако натисне в началната страница бутона **Public Groups**, ще види всички съществуващи публични групи, като кликвайки върху една от тези групи, ще бъде препратен в съответната стая. Най-отдолу има възможността да направи нова група. Единствено се изисква да се напише нейното име в текстовото поле и да кликне бутона до него.

На фигура 4-6 е показано това:



Фигура 4-6: Страницата с груповите чат стаи

# Заключение

В рамките на настоящия дипломен проект успях да разработя работещо приложение за изпращане на съобщения в уеб среда ChatApp. Макар и главните цели на настоящата разработка да бяха изпълнени, проектът може да се доразвие, като най-напред оптимизация при правене на заявки към базата данни би била нужна, за да може приложението да работи по-бързо и по-ефикасно и бих внедрил приложението, използвайки Firebase CLI. Освен това бих въвел други функционалности като изпращане на снимки, правене на затворени групи с определени потребители и филтриране на съобщенията и към уеб приложението би могло да се разработи мобилно приложение.

# Информационни източници

1. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7439968/> - проучване за ефекта на социалната дистанция над психическото здраве на хората
2. <https://pubmed.ncbi.nlm.nih.gov/32304649/> - проучване за ефекта на противоепидемичните мерки над здравето на хората
3. <https://en.wikipedia.org/wiki/Messenger_(software)> – информация за приложението Messenger
4. <https://en.wikipedia.org/wiki/Discord> - информация за приложението Discord
5. <https://en.wikipedia.org/wiki/Viber> - информация за приложението Viber
6. <https://reactjs.org/docs/getting-started.html> - документацията на React JS
7. <https://firebase.google.com/docs> - документацията на Firebase
8. <https://github.com/csfrequency/react-firebase-hooks/tree/09bf06b28c82b4c3c1beabb1b32a8007232ed045/> - документацията към react-firebase-hooks
9. <https://reactrouter.com/en/main> - документацията към react-router

# Приложение

Линк към GitHub repository, съдържащо сорс кода на проекта: <https://github.com/martinrivanov/chat-app>

# Рецензия на дипломен проект

|  |  |  |  |
| --- | --- | --- | --- |
| **Тема на дипломния проект** |  | | |
| **Ученик** |  | | |
| **Клас** |  | | |
| **Професия** |  | | |
| **Специалност** |  | | |
| **Ръководител- консултант** |  | | |
| **Рецензент** |  | | |
| **Критерии за допускане до защита на дипломен проект** | | **Да** | **Не** |
| Съответствие на съдържанието и точките от заданието | |  |  |
| Съответствие между тема и съдържание | |  |  |
| Спазване на препоръчителния обем на дипломния проект | |  |  |
| Спазване на изискванията за оформление на дипломния проект | |  |  |
| Готовност за защита на дипломния проект | |  |  |
| Силни страни на дипломния проект |  | | |
| Допуснати основни слабости |  | | |
| Въпроси и препоръки към дипломния проект |  | | |

**ЗАКЛЮЧЕНИЕ:**

Качествата на дипломния проект дават основание ученикът/ ученичката ................................................... да бъде допуснат/а до защита пред членовете на комисията за подготовка, провеждане и оценяване на изпит чрез защита на дипломен проект- част по теория на професията.

.........05.2023г. Рецензент:................................

град София