Съдържание

[1. Списък с цитирани фигури 3](#_Toc195548480)

[2. Увод 4](#_Toc195548481)

[3. Глава първа: Теоретична част 4](#_Toc195548482)

[1.3. Цел на проекта 4](#_Toc195548483)

[1.4. Съществуващи решения 6](#_Toc195548484)

[4. Глава втора: Практическа част 6](#_Toc195548485)

[4.1. Проектиране на системата 6](#_Toc195548486)

[4.1.1. Архитектура на приложението 6](#_Toc195548487)

[4.1.2. Диаграма на базата данни 7](#_Toc195548488)

[4.1.3. Диаграма на базата данни 10](#_Toc195548489)

[4.1.4. Създаване на приложение 12](#_Toc195548490)

[4.1.5. Структура на проекта и основни файлове 13](#_Toc195548491)

[4.1.6. Изтегляне на пакети 14](#_Toc195548492)

[4.2. Създаване и разпределяне на папки 15](#_Toc195548493)

[4.2.1. Файл **.env** 15](#_Toc195548494)

[4.2.2. Папката lib 16](#_Toc195548495)

[4.2.3. Папката UI 17](#_Toc195548496)

[4.2.4. Папката Hooks 17](#_Toc195548497)

[4.2.5. Папката Utils 18](#_Toc195548498)

[4.2.6. Папката Context 18](#_Toc195548499)

[4.2.7. Папката Public 18](#_Toc195548500)

[4.3. Навигация в приложението 18](#_Toc195548501)

[4.3.1. Конфигуриране на **next.config.mjs** 18](#_Toc195548502)

[4.3.2. Как работи навигацията в Next.js 19](#_Toc195548503)

[4.3.3. Падащо меню за навигация 21](#_Toc195548504)

[4.3.4. Навигация с **Link** 22](#_Toc195548505)

[4.4. Форми за Вход и Регистрация 23](#_Toc195548506)

[4.4.1. Компонент за вход 23](#_Toc195548507)

[4.4.2. Компонент за регистрация 23](#_Toc195548508)

[4.4.3. Компонент за профил 24](#_Toc195548509)

[4.4.4. Функция за регистрация на потребител 25](#_Toc195548510)

[4.4.5. Актуализиране на профил 25](#_Toc195548511)

[4.4.6. Функция за вход 26](#_Toc195548512)

[4.5. Изпращане на съобщения между два потребителя 26](#_Toc195548513)

[4.5.1. Функцията getMessages 27](#_Toc195548514)

[4.5.2. Функцията getCurrentUser 27](#_Toc195548515)

[4.5.3. Съобщения в реално време 28](#_Toc195548516)

[4.5.4. Изписване на съобщенията 29](#_Toc195548517)

# Списък с цитирани фигури

[Фигура 4.1.2.1.: Дефиниция на таблица за потребителски профили](#_Toc194921170)

[Фигура 4.1.2.2: Дефиниция на таблица за предлагане на приятелства](#_Toc194921171)

[Фигура 4.1.2.3.: Дефиниция на таблица за чат стаи](#_Toc194921172)

[Фигура 4.1.2.4:Дефиниция на таблица за съобщения](#_Toc194921173)

[Фигура 4.1.4.1.:Избор при създаване на уеб приложение](#_Toc194921174)

[Фигура 4.2.1.1.: .env файла](#_Toc194921175)

[Фигура 4.2.2.1.: Инициализиране на базата данни](#_Toc194921176)

# Увод

В съвременният свят комуникацията в реално време е от голямо значение за

разбирателството между хората. С постоянното развитие на технологии нараства необходимостта от гладка, бърза и надеждна комуникация, чат системите играя важна роля както в личния, така и в професионалния живот.

В дипломният проект, ще представя своята разработена чат система, като използвам технологиите Next.js, Supabase и Tailwind CSS. Основната цел на проекта е създаването на функционална и удобна платформа за изпращане и получаване на съобщения между потребителите, както и управление на приятелства и покани.

Очакваните резултати включват изграждане на стабилна и сигурна система, която осигурява гладка комуникация, оптимизирано потребителско изживяване и ефективно управление на потребителските данни.

# Глава първа: Теоретична част

## Цел на проекта

В съвременния дигитален свят комуникацията в реално време е неизменна част от ежедневието на хората. Съществуват множество платформи като Discord, Viber, Messenger и други, които осигуряват възможност за бърза и ефективна връзка между потребителите. Настоящият дипломен проект има за цел разработването на уеб-базирана чат система, която позволява обмен на съобщения в реално време, както и управление на списък с контакти чрез функционалност за изпращане и приемане на приятелски покани.

За постигането на тази цел е необходимо реализирането на следните ключови задачи: Създаване на уеб приложение чрез Next.js

* + 1. Създаване на уеб приложение с Next.js

Проектът ще бъде разработен с помощта на Next.js – популярен framework за React, който осигурява висока производителност, оптимизация и ефективно изграждане на динамични уеб приложения. Използването му ще допринесе за подобряване на цялостното потребителско изживяване и бързодействието на системата.

* + 1. Изграждане на база данни със Supabase

Данните, свързани с потребителите, изпратените съобщения и приятелските връзки, ще бъдат съхранявани в база данни, изградена със Supabase – облачна платформа, предоставяща back-end услуги и база данни в реално време. Supabase предлага също така автентикация и управление на потребителски достъп, което допринася за сигурността на системата.

* + 1. Реализиране на комуникация в реално време

Основна функционалност на приложението е изпращането и получаването на съобщения в реално време, без необходимост от презареждане на страницата. Това ще бъде постигнато чрез Supabase Realtime, който осигурява автоматична синхронизация на данните между участниците в разговора.

* + 1. Разработване на система за приятелства

За да могат потребителите да комуникират, ще бъде изградена система за изпращане, приемане и отказване на приятелски покани. Тази информация ще бъде организирана в отделни таблици в базата данни, като ще се осигури интуитивен интерфейс за управление на контактите.

* + 1. Имплементиране на система за регистрация и автентикация

Достъпът до платформата ще бъде ограничен до регистрирани потребители. Регистрацията и входът в системата ще се осъществяват чрез Supabase Auth, който ще предоставя сигурен механизъм за удостоверяване на потребителите чрез имейл и парола.

* + 1. Разработване на интуитивен потребителски интерфейс

Визията и удобството на потребителския интерфейс са от съществено значение за успешното използване на системата. За тази цел ще бъде приложен Tailwind CSS, който позволява изграждане на модерен, адаптивен и интуитивен дизайн. Визията и удобството на потребителския интерфейс са от съществено значение за успешното използване на системата. За тази цел ще бъде приложен Tailwind CSS, който позволява изграждане на модерен, адаптивен и интуитивен дизайн.

* + 1. Хостване на проекта и осигуряване на достъпност

За да бъде достъпно за потребителите, уеб приложението ще бъде хостнато чрез Vercel – платформа, специализирана за разгръщане на Next.js проекти. Това ще осигури лесен достъп до системата, както и непрекъсната наличност на услугите. За да бъде достъпно за потребителите, уеб приложението ще бъде хостнато чрез Vercel – платформа, специализирана за разгръщане на Next.js проекти. Това ще осигури лесен достъп до системата, както и непрекъсната наличност на услугите.

* + 1. Очаквани резултати

В резултат от изпълнението на горепосочените задачи се очаква създаването на функционална, надеждна и сигурна уеб-базирана чат система, която осигурява ефективна комуникация в реално време. Проектът ще демонстрира възможностите на Next.js и Supabase за изграждане на модерни уеб приложения, като същевременно гарантира добро потребителско изживяване и висока производителност.

## Съществуващи решения

В съвременния дигитален свят съществуват множество чат платформи, които предоставят възможност за комуникация между потребители. Сред най-популярните са Messenger, Viber, Discord и други. Тези системи предлагат разнообразни функции, включително текстова и гласова комуникация, групови чатове, мултимедийни съобщения и интеграция с различни услуги.

Въпреки широката им популярност, съществуващите решения често са обвързани със специфични екосистеми, изискват регистрация с телефонен номер или социален профил и понякога налагат ограничения върху начина, по който потребителите могат да взаимодействат.

Създаването на нова чат система дава възможност за персонализиран подход и внедряване на специфични функции според нуждите на проекта. Основният фокус на настоящото решение е опростената комуникация в реално време, лесното управление на приятелства и съобщения, както и интеграцията на модерни технологии като Next.js, Supabase и Tailwind CSS за оптимална работа и удобен интерфейс.

# Глава втора: Практическа част

## Проектиране на системата

### Архитектура на приложението

Проектът е разделен на три основни части:

Frontend частта от приложението е тази, с която потребителят взаимодейства. За нейното изграждане ще се използват Next.js и Tailwind CSS. Next.js ще бъде използван за създаване на интерфейса, навигацията и динамичното зареждане на съдържание, докато Tailwind CSS ще се използва за дизайна на самия интерфейс, което ще позволи бързо създаване и адаптиране на компоненти към различни устройства. Компонентите на frontend частта включват чат интерфейс, който включва списък с чатове, поле за въвеждане на съобщения и секция за показване на съобщения, интерфейс за потребителите, включващ регистрация, вход, управление на приятелства и профил на потребителя и страница за обновяване в реално време, която ще осигурява синхронизация на съобщенията между потребителите в реално време.

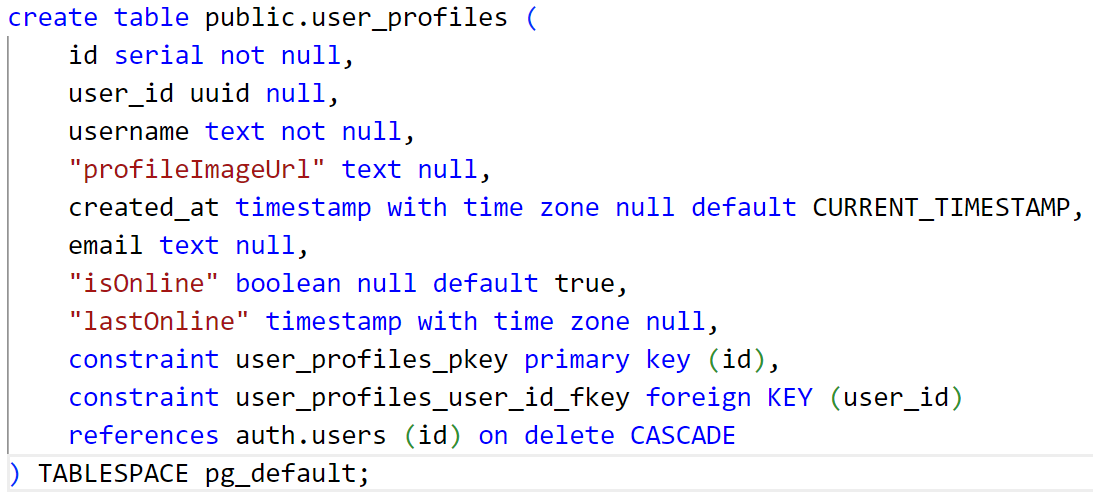
Backend (Сървър) частта ще използва Supabase, който предоставя база данни, автентикация и синхронизация в реално време. Базата данни ще съхранява информация за потребителите, съобщенията, поканите за приятелства и състоянието на всеки чат. Supabase Realtime ще се използва за автоматично обновяване на съобщенията между потребителите, което е основна част за поддържане на активни чатове. Supabase Auth ще се грижи за автентикацията на потребителите чрез имейл и парола, осигурявайки сигурност на платформата.

API Layer ще служи като комуникация между frontend и backend частите на приложението. В този слой ще се изградят основни функции като регистрация и вход на потребителите, изпращане на съобщения и управление на приятелства.

Хостинг ще се осъществява чрез платформа като Vercel, която позволява лесно качване на приложението в интернет и го прави достъпно за потребителите. Платформата е оптимизирана за Next.js, което гарантира бързо и ефективно хостване на проекта.

### Диаграма на базата данни

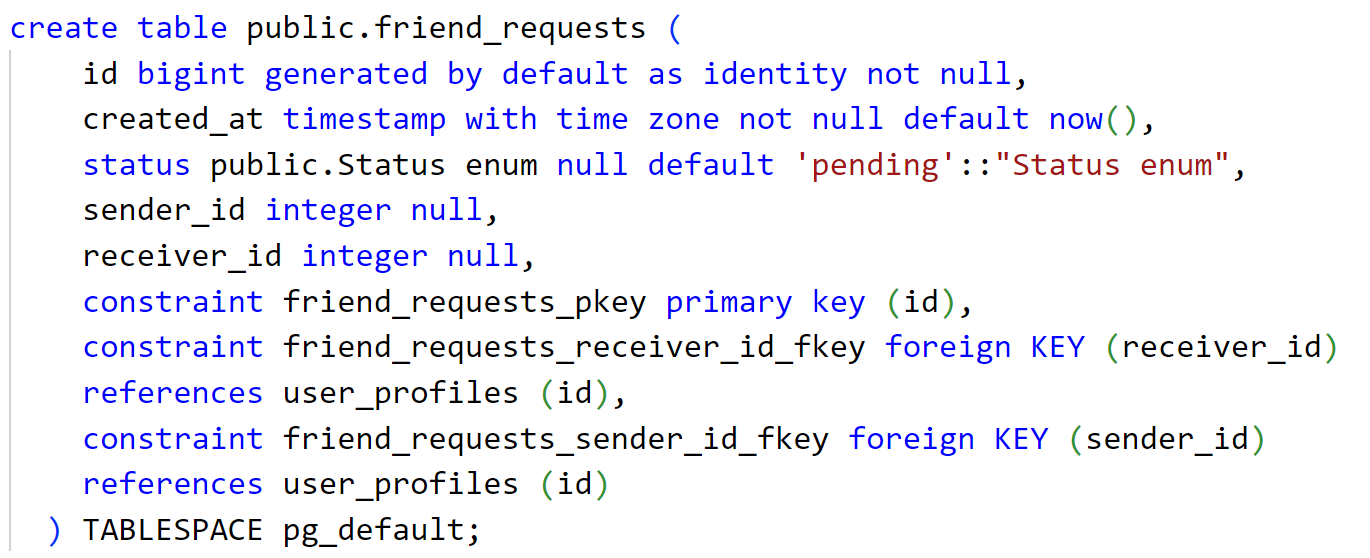
Таблицата user\_profiles съхранява данни за потребителските профили. Полето id е уникален идентификатор за профила и е първичен ключ. User\_id е свързано с уникален потребител от таблицата auth.users. Username е потребителското име и е задължително. profileImageUrl съдържа URL на профилната снимка, което не е задължително, но е пожелателно. Created\_at указва времето на създаване на профила, като по подразбиране е текущото време. Email е имейл адресът на потребителя и също не е задължително поле. Полето isOnline показва дали потребителят е онлайн, като по подразбиране е true. LastOnline съдържа времето на последно онлайн присъствие и не е задължително. Повечето от тези полета като user\_id, profileImageUrl и email може да не са задължителни, но когато се попълват данните в приложението, няма как да останат празни в базата данни.

[](#Ф3121) Таблицата има две ограничения: първичен ключ за id и външен ключ за user\_id, който се свързва с таблицата auth.users и изтрива профила автоматично при изтриване на потребителя.

Фигура 4.1.2.1.: Дефиниция на таблица за потребителски профили

Таблицата friend\_requests съхранява заявките за приятелство между потребители. Полето id е уникален идентификатор за всяка заявка и е първичен ключ. created\_at указва времето на създаване на заявката и по подразбиране приема текущото време.

Колоната status представлява състоянието на заявката и използва Status enum, който може да съдържа стойности като `pending`, `accepted` и `declined`. По подразбиране заявката започва със статус `pending`, тъй като очаква действие от получателя – той може да я приеме или откаже.

Полетата sender\_id и receiver\_id указват съответно кой е изпратил и кой е получил заявката. Те са външни ключове, свързани с таблицата user\_profiles, което гарантира, че само съществуващи потребители могат да изпращат и получават заявки. 

Фигура 4.1.2.2: Дефиниция на таблица за предлагане на приятелства

Таблицата chat\_room съхранява информация за личните чатове между двама потребители. Полето id е уникален идентификатор за всеки чат и е първичен ключ. created\_at указва времето на създаване на чата и по подразбиране приема текущото време.

Полетата user1 и user2 представляват участниците в чата. Те са външни ключове, свързани с таблицата user\_profiles, което гарантира, че само съществуващи потребители могат да бъдат част от чат.

Тази таблица осигурява структура за съхранение на разговори между двама потребители, като всеки ред представлява един уникален чат между тях.

A screenshot of a computer code

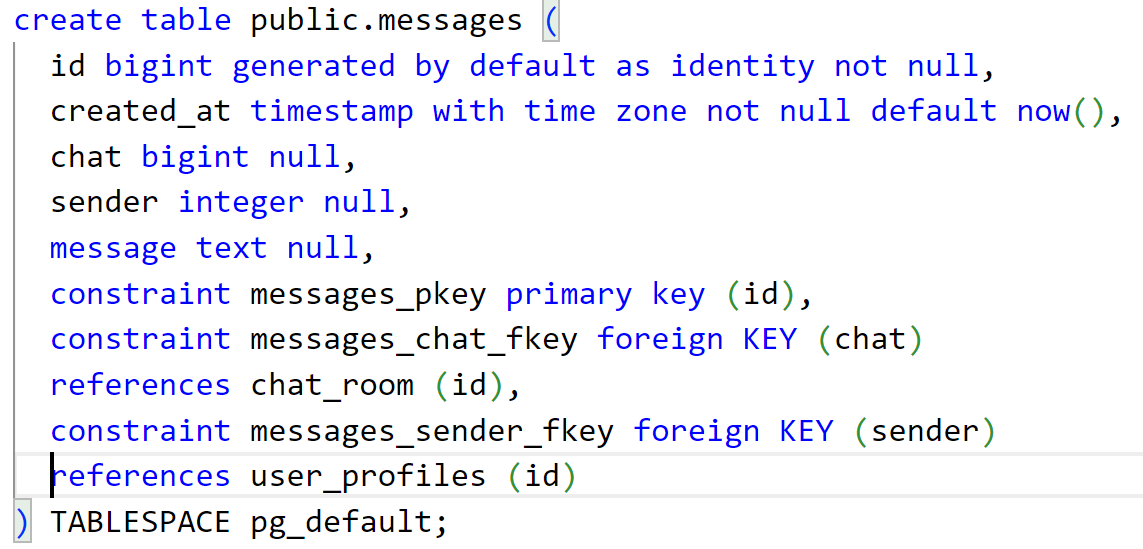
AI-generated content may be incorrect.

Фигура 4.1.2.3.: Дефиниция на таблица за чат стаи

Таблицата messages съхранява съобщенията, изпратени в чат стаите. Полето id е уникален идентификатор за всяко съобщение и служи като първичен ключ. created\_at записва времето на изпращане на съобщението и по подразбиране приема текущото време.

chat указва в кой чат е изпратено съобщението и е външен ключ, свързан с chat\_room(id), което гарантира, че всяко съобщение принадлежи на валидна чат стая. sender определя кой потребител е изпратил съобщението и е външен ключ, свързан с user\_profiles(id).

Полето message съдържа съдържанието на самото съобщение, като може да бъде текстово съобщение или друг тип съдържание, в зависимост от разширенията на системата.



Фигура 4.1.2.4:Дефиниция на таблица за съобщения

### Диаграма на базата данни

На фигурата може да видим диаграма, тя представлява релационната структура на базата данни за чат система, като визуализира връзките между основните таблици. Тя показва как потребителите комуникират помежду си чрез чат стаи и съобщения, както и как се изпращат покани за приятелство. Всяка таблица има своя роля в цялостното функциониране на системата и е свързана с други таблици чрез външни ключове.

Централна за системата е таблицата user\_profiles, която съдържа информация за всеки потребител, като идентификатор, потребителско име, имейл и статус на активност. Тази таблица е свързана с останалите, тъй като всеки потребител може да изпраща и получава приятелски покани (friend\_requests), да участва в чат стаи (chat\_room) и да изпраща съобщения (messages).

Връзките между потребителите се управляват чрез таблицата friend\_requests, която записва изпратените покани за приятелство и техния статус (изчакваща, приета или отказана). След като двама потребители станат приятели, те могат да започнат чат комуникация чрез таблицата chat\_room, която създава уникална стая за всеки разговор.

Всяка чат стая съдържа множество съобщения, които се записват в таблицата messages. Всяко съобщение има идентификатор, референция към съответната чат стая, изпращач и текстово съдържание. Чрез тази структура се гарантира, че всички съобщения са свързани със съответните разговори и участници.

Диаграмата ясно показва логическите връзки между таблиците и как данните се организират в системата. Чрез използването на външни ключове се гарантира целостта на данните, като например невъзможността да съществува съобщение без валидна чат стая или потребител. Тази организация прави базата данни ефективна и лесна за разширяване с допълнителни функционалности.

A screenshot of a computer

AI-generated content may be incorrect.

Фигура 4.1.3.: Диаграма на базата данни

### Създаване на приложение

A computer screen shot of a computer code

AI-generated content may be incorrect. За да създадем проекта първо отваряме своята среда за разработка. За текущия проект ще използвам Visual Studio code. Когато отворим своята среда, отваряме терминала и пишем командата **„npx create-next-app@latest chat-app”. npx** е инструмент от Node.js, който позволява да инсталираме пакети директно, без да ги изтегляме глобално. **create-next-app@latest** е официална команда, позволяваща автоматичното създаване на проект с всички необходими файлове и настройки, докато **@latest** означава че ще използваме последната и най-нова версия за нашето приложение. **Chat-app** e името на папката, в която се намира всичко. Когато се напише цялата команда ще започнат да се теглят пакетите. След това ще видим допълнителни въпроси.

Фигура 4.1.4.1.:Избор при създаване на уеб приложение

За този проект няма да използвам TypeScript, тъй като JavaScript е по-лесен за започване и не изисква допълнителна конфигурация. Това улеснява бързото разработване, без да се налага да дефинираме типове за всяка променлива или функция.

Избрах да използвам ESLint, защото той ще помага за поддържането на чист и подреден код. Този инструмент открива синтактични грешки, несъответствия в стила и други потенциални проблеми, което ще подобри качеството и четимостта на кода по време на разработката.

Tailwind CSS е много важно да го включим. Това е библиотека, която ще помогне за бързото писане и стилизиране на проекта. Tailwind ускорява разработката на UI, като позволява бързо стилизиране с малко код.

Файловете на проекта ще бъдат организирани в папка src/, което е стандартна практика за по-добра структура и разделяне на кода. Това позволява по-лесно управление на компонентите и страниците в приложението.

Избрах да използвам App Router, който е новата система за маршрутизация в Next.js. Вместо класическата pages/ директория, app/ позволява по-гъвкаво управление на сървърни компоненти, подобрено кеширане и по-добра производителност. Така ще си осигурим повече контрол върху структурата и сървърните компоненти на приложението.

Turbopack е другото нещо, което ще вземем под внимание. Не го включвам, защото той все още е в експериментален стадий. Въпреки че обещава значително по-бърза компилация в сравнение с Webpack, засега предпочитам да използвам стабилното решение на Webpack, което е тествано и широко използвано.

Последното нещо е как искаме да импортваме файлове. В Next.js това става по подразбиране с **@/**, това улеснява пътя. Вместо да пишем точки, в зависимост от местоположението на файла само въвеждаме @/ това ни праща в най-горната папка. От там може по лесно да навигираме до файла, който търсим.

С тези настройки проектът е подготвен за работа с Next.js, ESLint и Tailwind CSS, като кодът ще бъде организиран в src/ директория. Използваме App Router за по-ефективна маршрутизация и Webpack като стабилно решение. След приключване на конфигурацията, можем да продължим с разработката на функционалностите на приложението.

### Структура на проекта и основни файлове

В нашия новосъздаден проект ще видим следните файлове:

node\_modules – Съдържа всички инсталирани зависимости (пакети), необходими за проекта.

public – Папка за статични файлове като изображения, шрифтове и икони. Файловете тук се достъпват директно чрез URL.

src/ – Основната директория за кода на приложението.

Вътре в src/app/:

layout.js – Дефинира общото оформление (layout) на страниците. Използва се за споделени елементи като хедър и футър.

page.js – Главната начална страница на приложението.

Конфигурационни и системни файлове

favicon.ico – Иконата на сайта, която се показва в браузъра.

globals.css – Глобален CSS файл, където се задават основните стилове за целия проект.

.gitignore – Определя кои файлове и папки да бъдат игнорирани от Git (например node\_modules).

eslint.config.mjs – Конфигурация за ESLint, който анализира кода и помага да се поддържат добри практики.

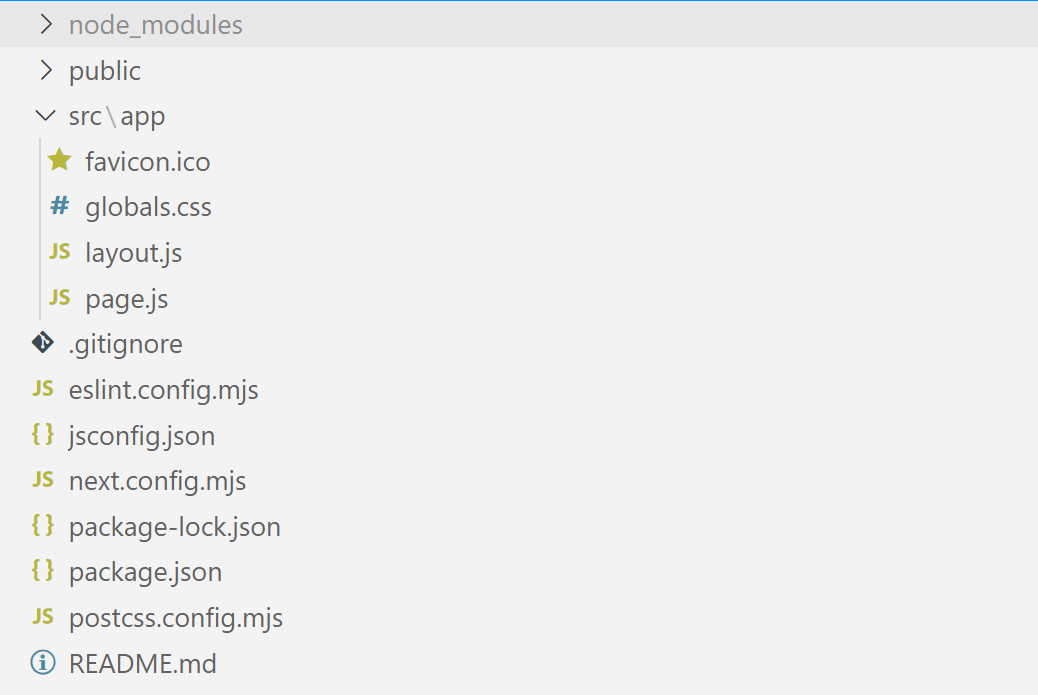
jsconfig.json – Конфигурация за JavaScript, която улеснява импортите и работата с пътищата в проекта.

next.config.mjs – Основният конфигурационен файл за Next.js, където могат да се зададат настройки като оптимизация на изображения и API маршрути.

package.json – Описва проекта, съдържа информация за зависимостите и командите за изпълнение (скриптове).

package-lock.json – Заключва версиите на зависимостите, за да се гарантира, че всички в екипа използват една и съща версия на пакетите.

postcss.config.mjs – Конфигурация за PostCSS, който се използва за обработка на CSS (необходим за Tailwind CSS).

README.md – Документ с инструкции и описание на проекта.

Фигура 4.1.5.: Папки и файлове в новосъздаден проект

### Изтегляне на пакети

A black and blue text

AI-generated content may be incorrect. За текущия проект ни трябва само един допълнителен пакет и той е библиотеката на Supabase. **@supabase/supabase-js** е официалната JavaScript библиотека (SDK) с която ще работим. С нея можем да се свързваме с базата данни, да пращаме заявки за четене и писане, да работим в реално време и много други.

Фигура 4.1.6.:Команда за изтегляне на пакет

## Създаване и разпределяне на папки

Има няколко важни папки, които да си създадем преди да започнем самото писане на код за проекта.

### Файл **.env**

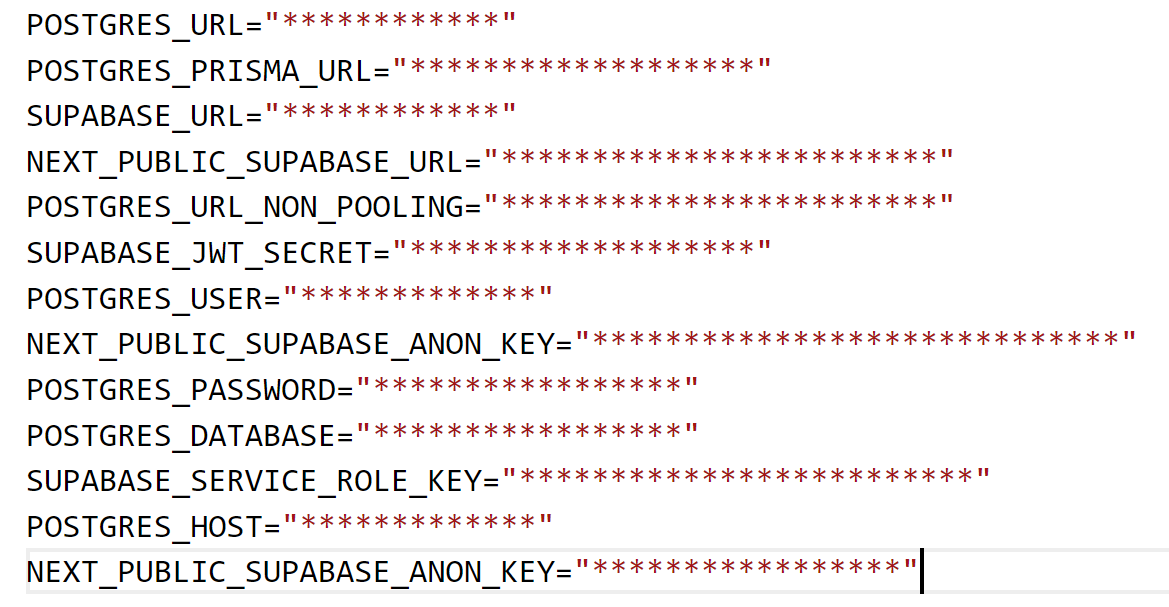
Файлът .env (от англ. "environment") е специален текстов файл, който се използва за съхраняване на околните променливи в проектите. Околните променливи са стойности, които дефинират конфигурацията на приложението и могат да включват чувствителни данни, като API ключове, потребителски имена, пароли, токени за автентикация и други важни данни, които не трябва да бъдат споделяни публично.

Значение на .env файла:

Сигурност и конфиденциалност на данните: Основната причина за използването на .env файл е, че той предоставя начин за безопасно съхранение на чувствителни данни. Данни като API ключове и пароли не трябва да бъдат включвани директно в изходния код на проекта, особено когато проектът е публикуван на публични репозитории, например в GitHub. С .env файла тези данни се държат локално на машината или сървъра и не се излагат на външни лица.

Гъвкавост и конфигурируемост: Чрез използването на .env файл, разработчиците могат да дефинират различни стойности за различни среди. Например, едни стойности могат да бъдат използвани в локалната среда на разработчика, други в тестовата среда, а трети в продукционната среда. Това позволява на проектите да бъдат по-гъвкави и лесно конфигурируеми в зависимост от нуждите на съответната среда, без да е необходимо да се променя кодът.

Лесно управление на конфигурациите: Използването на .env файл улеснява конфигурирането на проектите, като същевременно намалява риска от грешки. Вместо да въвеждат ръчно чувствителни данни на различни места в кода, разработчиците могат просто да добавят стойностите в .env файла и да използват съответните променливи в приложението. Това прави кода по-четим и лесен за поддръжка.

За нашият пример ни трябват най-важната информация, която се отнася до базата ни данни. Нея може да я намерим в Vercel, мястото което ще хоства нашия проект. 

Фигура 4.2.1.1.: .env файла

Така трябва да изглежда нашият файл, разбира се без звездичките, но няма да разкривам ценна информация в проекта.

### Папката lib

Папката lib е другата много важна. В нея ще държим всички файлове, свързани с базата данни и работата с профили. Един от важните файлове в нея е **setUp.js.** Това е файлът, в който инициализираме базата.

import { createClient } from '@supabase/supabase-js'

A close up of a computer code

AI-generated content may be incorrect.Тази линия импортира функцията createClient от библиотеката @supabase/supabase-js, която е JavaScript SDK на Supabase. Тази функция се използва за създаване на клиент, който ще се свърже с Supabase базата данни.

Фигура 4.2.2.1.: Инициализиране на базата данни

export const supabase = createClient(process.env.NEXT\_PUBLIC\_SUPABASE\_URL, process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY);

Тази линия създава клиентски обект supabase, който се свързва със Supabase проекта, използвайки стойности за URL и анонимния ключ на проекта.

process.env.NEXT\_PUBLIC\_SUPABASE\_URL:

Това е URL адресът на нашия Supabase проект. Той се съхранява в променлива на средата и може да бъде достъпен чрез process.env.

process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY

Това е анонимният ключ за достъп до проекта, който позволява на клиентите да изпълняват основни операции без нужда от автентикация. Този ключ също е съхранен в променлива на средата.

### Папката UI

Папката UI (потребителски интерфейс) съдържа всички компоненти и ресурси, които са отговорни за визуализацията и взаимодействието с потребителя в проекта. Тя обикновено включва файлове като:

React компоненти: Това са отделни части от интерфейса, като бутони, форми, навигационни панели и други елементи, които се използват многократно в приложението.

CSS/Tailwind CSS класове: Стилове, които дефинират визуалната част на компонентите, включително оформление, цветове, шрифтове и анимации.

Изображения и икони: Ресурси, които са необходими за графичната част на интерфейса, като икони, лога и изображения, използвани в компоненти.

Целта на папката UI е да предостави лесен начин за управление на всички визуални компоненти, като по този начин се улеснява поддръжката и разширяването на интерфейса на проекта. Това позволява централизирано управление на визуалните елементи и улеснява тяхното повторно използване в различни части на приложението.

### Папката Hooks

Папката Hooks съдържа персонализирани React hook-ове, които капсулират логика за многократна употреба. В настоящия проект в тази папка се намира hook-ът useAuth, който отговаря за управлението на автентикацията на потребителите. Той обединява логиката за вход, изход и следене на текущата сесия на потребителя чрез Supabase. С използването на useAuth се улеснява достъпът до информация за потребителя във всички компоненти на приложението и се осигурява по-чист и структуриран код.

### Папката Utils

Папката Utils съдържа помощни функции и скриптове, които подпомагат основната логика на приложението. Те не са свързани директно с визуалната част или с конкретни компоненти, а изпълняват отделни логически задачи, които се използват на различни места в проекта. Извеждането на такава логика в отделни файлове улеснява повторната ѝ употреба, прави кода по-организиран и по-лесен за поддръжка.

В тази папка се намират функции, свързани с обработка на пътища, локално съхранение на данни и форматиране на време. Благодарение на папката utils, проектът е по-модулен, а логиката е отделена от интерфейсната част, което улеснява и бъдещи разширения и промени.

### Папката Context

Папката Context съдържа глобални състояния и логика за споделяне на данни между различни компоненти в приложението чрез React Context API. Основната ѝ цел е да улесни достъпа до определени данни – като информация за текущия потребител – без да е необходимо те да се предават ръчно през всяко ниво на компоненти (props). Така се осигурява по-ефективно управление на състоянието и по-добра структура на проекта, особено при по-големи приложения с много взаимосвързани елементи.

### Папката Public

Папката Public съдържа статични файлове, които са достъпни за всички потребители на приложението. Това са ресурси, които не преминават през Webpack или други билд процеси и се предоставят директно на клиентската страна. В нея обикновено се намират изображения, икони, шрифтове, и други файлове, които трябва да бъдат достъпни чрез URL адреси, като например favicon.ico или изображения, използвани в приложението.

Файловете в public директно се обслужват от сървъра и могат да бъдат достъпни чрез пътя /името-на-файла. Това позволява бърз достъп до тези ресурси, без да е нужно те да се обработват или пакетирани.

## Навигация в приложението

### Конфигуриране на **next.config.mjs**

Файлът **next.config.mjs** е основният конфигурационен файл в Next.js, чрез който се

A screen shot of a computer code

AI-generated content may be incorrect.задават глобални настройки за приложението. В проекта съм използвал само настройка, която позволява зареждането на изображения от всякакви източници, което е необходимо за слагане на профилна снимка. В случая съм оставил **hostname** да е две звездички, защото това позволява повече гъвкавост, тъй като така може снимките да идват от всички домейни и не са лимитирани по никакъв начин. Това осигурява максимална съвместимост по време на разработка, като в бъдеще може да се ограничи до конкретни източници с цел сигурност. Другото, което можем да отбележим, е че файлът е **.mjs**, а не **.js**, това разширение показва, че файлът ще използва **export/import** синтаксис, който е стандартен в съвременните приложения.

Фигура 4.3.1.1.: Конфигурация на next.config.mjs

### Как работи навигацията в Next.js

Next.js използва своя вградена навигационна система, базирана на файловата структура

на проекта. В папката app, която представлява основната директория за страниците при използване на App Router, всяка поддиректория или файл с име **page.jsx** автоматично се счита за отделна страница. Това позволява лесно и логично организиране на навигацията в приложението, без използването на допълнителна библиотека.

Next.js предлага прости и сложни пътища:

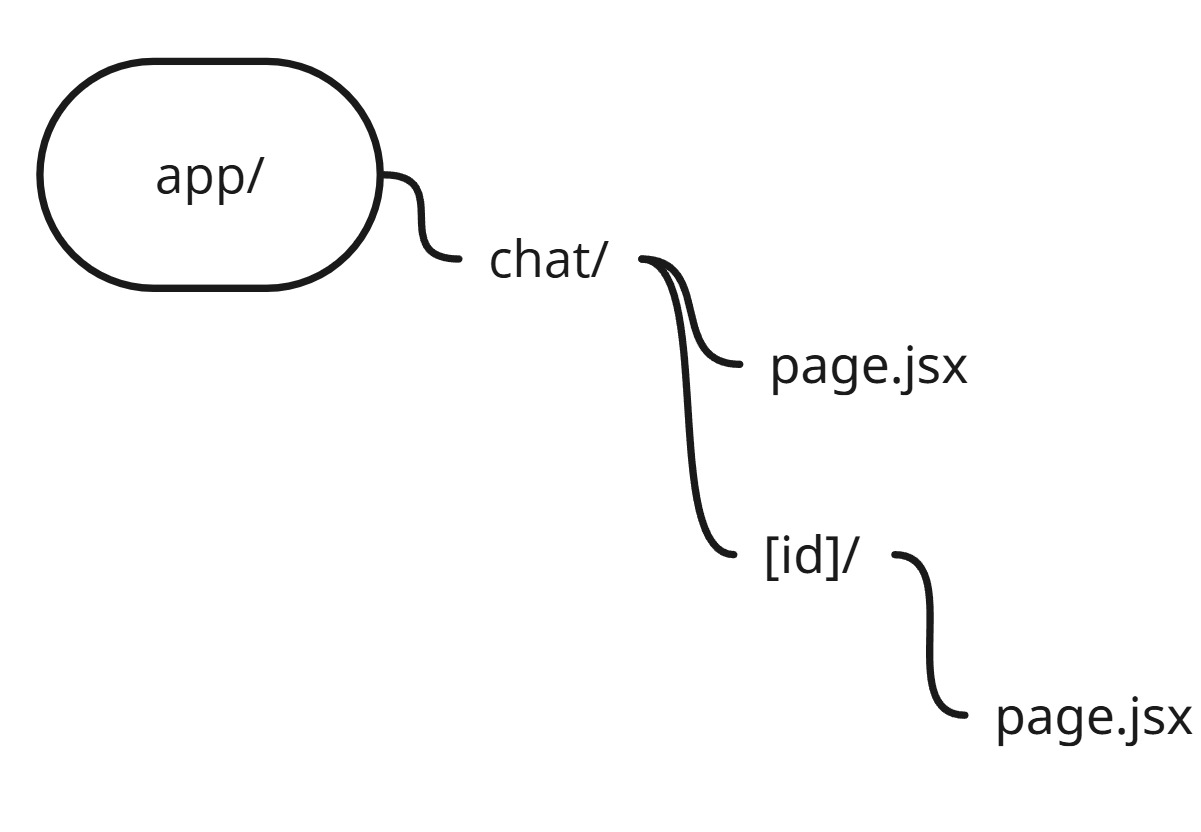
A diagram of a application

AI-generated content may be incorrect.Простите пътища в Next.js се създават чрез създаването на директория с файл page.jsx. Такива пътища водят към статични страници, които не се променят в зависимост от входящите параметри. Например, началната страница може да бъде дефинирана с файл app/page.jsx, който ще бъде достъпен на URL адреса **/**.

Фигура 4.3.2.1.: Структура на навигация за начална страница

Сложните пътища, известни още като динамични пътища, позволяват създаването на страници, чието съдържание се променя в зависимост от параметрите, подадени в URL адреса. Динамичните пътища са много полезни за приложения, които предлагат персонализирани или различни страници в зависимост от данни, като например продукти, потребителски профили или статии.

Динамичните пътища се дефинират чрез използването на квадратни скоби [ ] за създаване на параметри, които могат да се използват в URL. Например, за сайт с продукти, може да имате път, който води до конкретен продукт чрез параметър **id**.

 Това означава, че ако потребителят посети /chat/123, параметърът id ще бъде 123, и съдържанието на страницата ще се зареди динамично в зависимост от продукта с този идентификатор. Това служи, за да можем да вземем конкретния чат и съобщенията в него.

Фигура 4.3.2.2.: Пример за динамична навигация

### Падащо меню за навигация

В проекта съм използвал падащо меню, реализирано чрез компонент

DropdownMenu.jsx, намиращ се в папка **ui.** Представлява бърз и лесен начин на потребителя да навигира между различните страници.

A white background with black text

AI-generated content may be incorrect.Всички линкове, които ще се използват в навигацията се намират в масив от обекти, това позволява бързото им редактиране, в случай на промяна.

Фигура 4.3.3.1.: Масив от обекти с линкове

Всеки от тези линкове после се показва на екрана, чрез метод **.map()**

За всяка навигационна връзка се използва href, който задава към коя страница ще води. Функцията **clsx**, предоставена от Next.js, се използва за динамично задаване на CSS класове – включително условни стилове. Например, ако текущата страница съвпада с някой от линковете, той се оцветява в ярко червено и става неактивен за мишката. По този начин потребителят ясно вижда къде се намира и не може случайно да избере отново същата страница.A computer code with text

AI-generated content may be incorrect.

Фигура 4.3.3.2.: Метод map

Фигура 4.3.3.3.: Примерен Link

### Навигация с **Link**

Next.js предоставя специален компонент с име Link, който се използва за създаване на вътрешна навигация между страниците в приложението. Той идва от пакета **next/link** и се използва вместо стандартния HTML елемент **<a>**. Причината за това е, че Link осигурява клиентска навигация, което означава, че страниците се зареждат по-бързо и без цялостно презареждане на сайта.

A close-up of text

AI-generated content may be incorrect.Синтаксисът е прост — около елемента, който искаме да е линк (например текст или бутон), слагаме компонента **<Link>** и му подаваме атрибут href с пътя, към който искаме да води.

Фигура 4.3.4.1.: Разликата между a и Link

Основните разлики между **Link** и **a:**

Елементът <a> в HTML се използва за създаване на връзки между страници, но при кликване води до пълно презареждане на сайта, което забавя навигацията. За разлика от него, **Link** е специален компонент, предоставен от Next.js, който позволява бърза и плавна навигация между вътрешните страници на приложението, без да се презарежда цялата страница.

## Форми за Вход и Регистрация

### Компонент за вход

A computer screen shot of a program

AI-generated content may be incorrect.Компонента за вход включва логика за самото влизане в профила, както и формата, включваща две полета: едно за емайл и едно за парола. След въвеждане на своите данни и при успешно влизане, системата пренасочва потребителя към страницата му с чатове.

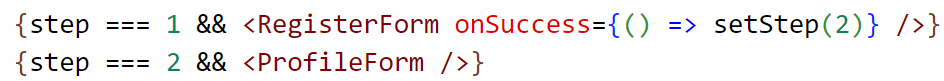
Фигура 4.4.1.1.: Функция login

Функцията **login** в този компонент. В момента, в който потребителя натисне бутона Login, системата изпълнява тази функция. Първо тя взима всички данни от полетата, после пробва да впише потребителя и след това, ако всичко е наред запазва данните му и го пренасочва, в противен случай хвърля грешка.

### Компонент за регистрация

Компонентът за регистрация включва логика за създаване на нов потребителски акаунт, както и форма, която изисква въвеждането на имейл и парола. След като потребителят попълни тези данни и натисне бутона за регистрация, системата изпълнява функцията за регистрация. Тя взима въведените данни, опитва се да създаде нов акаунт в системата и ако всичко е наред, преминава към следващата стъпка, при която потребителят трябва да попълни допълнителни данни за своя профил (например потребителско име и URL на профилна снимка)

Ако регистрацията е успешна, потребителят се пренасочва към страницата за попълване на профилни данни, а ако възникне грешка (например, вече съществуващ акаунт с този имейл), функцията хвърля грешка и потребителят не може да завърши процеса на регистрация.

Тази логика осигурява създаването на нов акаунт и прехвърляне към последващата стъпка за завършване на регистрационния процес, като гарантира, че данните на новите потребители са валидни.

Фигура 4.4.2.1.: Проверка между регистрация и профилна форма

### Компонент за профил

Компонентът за профилната форма позволява на потребителя да завърши попълването на своя профил, като въведе потребителско име и URL за профилна снимка.

Когато потребителят натисне бутона „Save Profile“, системата изпълнява функцията за актуализиране на профила, която взима въведените данни (потребителско име и URL на снимката), и ги актуализира в базата данни. След успешна актуализация, потребителските данни се обновяват в контекста на приложението, а потребителят бива пренасочен към страницата с чатове. Ако има грешка системата показва съобщението и позволява на потребителя да редактира и изпрати отново.

### Функция за регистрация на потребител

A screen shot of a computer code

AI-generated content may be incorrect.Тази функция **registerUser** е отговорна за регистрацията.Тя приема имейла и паролата попълнени във формата и запазва потребителя с помощта на **supabase.** След това при грешка, то я връща, а ако няма такава, запазва сесията и връща данните.

Фигура 4.4.4.1.: Функция registerUser

### Актуализиране на профил

A computer code with text

AI-generated content may be incorrect.Функцията updateProfile е отговорна за актуализирането на данните на потребителския профил в базата данни. Тя приема обект с данни за потребителския профил и ги записва в таблицата user\_profiles с помощта на **supabase**. Ако възникне грешка по време на процеса, тя се хвърля и се връща съобщение за грешката. Ако операцията е успешна, се връщат актуализираните данни на профила.

Фигура 4.4.5.1.: Функция updateProfile

### Функция за вход

Функцията **loginUser** е отговорна за влизането на потребителя в системата. Тя приема имейл и парола, въведени от потребителя, и използва Supabase за аутентикация на потребителя. Ако влизането е успешно, функцията актуализира състоянието на потребителя в таблицата **user\_profiles**, задавайки стойности за полетата **isOnline** и **lastOnline**. Ако възникне грешка по време на аутентикацията или при актуализацията на данни в базата, тя хвърля грешка и я връща. Ако всичко е наред, функцията запазва сесията на потребителя и връща данни за потребителския профил и достъпния токен.

A computer screen shot of a program

AI-generated content may be incorrect.

Фигура 4.4.6.1.: Функцията loginUser

## Изпращане на съобщения между два потребителя

Когато потребителят избере чат стая със свой приятел, той бива пренасочен към линк с формат **chat/[id]/[userId].** Първият параметър id се използва, за да намерим чат стаята от базата данни и се използва, за да се извлекат всички съобщения, свързани с нея. Вторият параметър userId се използва, за да можем да заредим данните за приятеля, с когото потребителят води разговор.

### Функцията getMessages

A screenshot of a computer code

AI-generated content may be incorrect.Функцията getMessages(chatId) се използва за извличане на всички съобщения, свързани с конкретен чат, като приема идентификатора на чата (chatId) и изпраща заявка към таблицата **messages** в базата данни чрез Supabase. С помощта на метода .eq("chat", chatId) се филтрират съобщенията, така че да се върнат само тези, които принадлежат към съответния чат. При грешка заявката прекъсва и се извежда съобщение за нея, а ако няма намерени съобщения, функцията връща празен масив. Ако заявката е успешна, връща се масив от обекти, съдържащи информацията за съобщенията, който се използва за визуализирането им в потребителския интерфейс.

Фигура 4.5.1.1.: Функцията getMessages

### Функцията getCurrentUser

Функцията **getCurrentUser** е отговорна за извличането на данни за конкретен потребител от таблицата user\_profiles в базата данни. Тя приема един параметър – user\_id, който е уникален идентификатор на потребителя. В зависимост от това дали стойността на user\_id е числова или не, тя използва динамичен ключ (user\_id или id), за да направи правилната заявка към базата данни. Тази проверка се прави, защото има и други функции, които я използват, но те могат да дадат данни от тип **UUID**, което е друг вид стандарт за уникални идентификатори, които се използват за осигуряване на уникалност на елементи в различни системa, с основна идея да е уникален, дори и между различни бази данни без нищо общо. Ако няма грешка, функцията връща данните на потребителя като масив от обекти, които съдържат информация за съответния потребител. В случай на грешка, тя я хваща и връща като обект с полето error, което съдържа съобщението за грешката. A screen shot of a computer code

AI-generated content may be incorrect.

Фигура 4.5.2.1.: Функцията getCurrentUser

### Съобщения в реално време

За да можем да изпращаме съобщения, първо трябва да създадем канал. Това става чрез функция на Supabase, наречена **channel**. Като аргумент тя получава името на канала, който да наблюдава – така всеки отделен разговор ще има собствен канал и няма да се пресичат. Докато е активен, каналът следи за промени. Подаваме му обект, в който се съдържат типът на събитието – **INSERT**, схемата, таблицата и филтър. Филтърът служи, за да знае точно кой чат да следи. Накрая получаваме **payload**, чрез който обработваме новото съобщение и го добавяме към състоянието на чата. Последното действие е **subscribe**, което гарантира, че ще слушаме този канал, докато не се отпишем от него, чрез излизане от самата страница, това може да го видим чрез функцията return.

A screen shot of a computer code

AI-generated content may be incorrect.

Фигура 4.5.3.1.: Записване за съобщения в реално време

### Изписване на съобщенията

Този код показва как се визуализират съобщенията в чата. Първо се извършва проверка дали масивът от съобщения (**chat**) съдържа поне един елемент. Ако има, се преминава през всяко съобщение и се извършва сравнение между **message.sender** и идентификатора на текущия потребител (**userData?.id**). Ако стойностите съвпадат, значи съобщението е изпратено от текущия потребител и се използва компонентът **UserChat**, който го визуализира от дясната страна на екрана. Ако идентификаторите не съвпадат, значи съобщението е получено от приятеля, и се използва компонентът **FriendChat**, който го показва от лявата страна, заедно с информация за подателя.

A computer code with text

AI-generated content may be incorrect.

Фигура 4.5.4.1.: Изписване на съобщения