Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Дмитрієва Олексія Ярославовича*
(ПІБ)

академічної групи *122-20-ск1*
(шифр)

напряму підготовки *122 Комп'ютерні науки*
(код і назва напряму підготовки)

на тему: *Розробка програми месенджера за допомогою MERN Stack*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Кабак Л.В*			
розділів:				
спеціальний	*доц. Кабак Л.В*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*ст. викл.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-ск1* (група) *Дмитрієв О.Я.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програми месенджера за допомогою MERN Stack*

затверджена наказом ректора НТУ «ДП» від *16 . 05 .2023 р.* № *350 -с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *доц.Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Дмитрієв О.Я.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.07.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 86 с., 17 рис., 3 дод., 24 джерела.

Об'єкт розробки: Програма-месенджер з використанням стеку MERN.

Метою даної кваліфікаційної роботи є розробка веб-месенджера з використанням React, Express.js, Node.js та MongoDB. Додаток має на меті надати користувачам інтуїтивно зрозумілий інтерфейс та повний набір функцій для ефективного та безпечного спілкування.

У вступі даної кваліфікаційної роботи проаналізовано та розглянуто сучасний стан програм-месенджерів, визначено конкретну мету дослідження, окреслено сферу застосування проекту та обґрунтовано актуальність обраної теми. Також представлено постановку задачі, яка висвітлює цілі та завдання процесу розробки, а також вимоги до програмного забезпечення, технології та інструментальні засоби, які будуть використовуватися.

У першому розділі кваліфікаційної роботи проводиться детальний аналіз існуючих програм-месенджерів, підкреслюється важливість та актуальність розробки веб-орієнтованого рішення. У розділі визначено конкретні вимоги до функціональності програми-месенджера.

У другому розділі обрано відповідні платформи для розробки та описано процес проектування та розробки. Також описано алгоритм та структуру функціоналу додатку. Крім того, в розділі висвітлено процес розгортання, визначено вхідні та вихідні дані, а також надано інформацію про вимоги до апаратного забезпечення для запуску програми-месенджера.

В економічному розділі кваліфікаційної роботи визначено трудомісткість розробки програми-месенджера. Вона включає в себе розрахунок вартості робіт з розробки та оцінку часу, необхідного для їх виконання.

Практичне значення проекту полягає в розробці веб-програми-месенджера зі зручним інтерфейсом. Додаток має на меті надати користувачам безпечну та надійну платформу для спілкування в режимі реального часу, обміну повідомленнями та спільного використання мультимедіа. Використовуючи такі технології, як React, Express.js, Node.js та MongoDB, програма-месенджер пропонує безперебійну роботу з такими функціями, як автентифікація користувача, шифрування повідомлень та сповіщення.

Актуальність цього програмного продукту зумовлена зростаючим попитом на ефективні та безпечні засоби комунікації в сучасну цифрову епоху. Зважаючи на зростаючу залежність від додатків для обміну повідомленнями в особистих і професійних цілях, веб-месенджер пропонує зручне та ефективне рішення.

Ключові слова: ПРОГРАМА-МЕСЕНДЖЕР, СТЕК MERN, REACT, EXPRESS.JS, NODE.JS, MONGODB.

ABSTRACT

Explanatory note: 86 p., 17 figs., 3 appx., 24 sources.

Object of development: A messenger program using the MERN stack.

The purpose of this thesis is to develop a web messenger using React, Express.js, Node.js, and MongoDB. The application aims to provide users with an intuitive interface and a full set of features for efficient and secure communication.

The introduction of this thesis analyzes and discusses the current state of messenger applications, defines the specific purpose of the study, outlines the scope of the project, and justifies the relevance of the chosen topic. It also presents the problem statement, which highlights the goals and objectives of the development process, as well as the requirements for software, technology, and tools to be used.

In the first chapter of the thesis, a detailed analysis of existing messenger programs is carried out, emphasizing the importance and relevance of developing a web-based solution. The section defines specific requirements for the functionality of the messenger program.

The second section selects the appropriate development platforms and describes the design and development process. The algorithm and structure of the application functionality are also described. In addition, the section highlights the deployment process, defines input and output data, and provides information on the hardware requirements for running the messenger application.

The economic section of the qualification work determines the labor intensity of developing a messenger program. It includes calculating the cost of development work and estimating the time required to complete it.

The practical significance of the project is to develop a web-based messenger application with a user-friendly interface. The application aims to provide users with a secure and reliable platform for real-time communication, messaging, and multimedia sharing. Using technologies such as React, Express.js, Node.js, and MongoDB, the messenger program offers a seamless experience with features such as user authentication, message encryption, and notifications.

The relevance of this software product is driven by the growing demand for efficient and secure means of communication in the modern digital age. Given the growing dependence on messaging applications for personal and professional purposes, web messenger offers a convenient and effective solution.

Keywords: MESSENGER PROGRAM, MERNA STACK, WEB APPLICATION, REACT, EXPRESS.JS, NODE.JS, MONGODB.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система;

ПК – персональний комп’ютер;

ПЗ – програмне забезпечення;

UML – Unified Modeling Language;

RAM – Random Access Memory;

HTML – HyperText Markup Language;

JSON – JavaScript Object Notation;

CSS – Cascading Style Sheets.

ВСТУП

Метою цієї кваліфікаційної роботи є розробка веб-месенджера з використанням стеку MERN, який включає MongoDB, Express.js, React та Node.js. Основною метою є створення сучасного та ефективного інструменту комунікації, який дозволяє безперешкодно обмінюватися повідомленнями та співпрацювати між користувачами.

У сучасному взаємопов'язаному світі додатки для обміну повідомленнями відіграють життєво важливу роль у полегшенні спілкування та обміну інформацією. Використовуючи можливості стеку MERN, ця кваліфікаційна робота має на меті використати можливості MongoDB як гнучкої та масштабованої бази даних, Express.js як надійного фреймворку для веб-додатків, React як універсальної бібліотеки JavaScript для побудови користувацьких інтерфейсів та Node.js як надійного середовища виконання.

Розроблений месенджер запропонує користувачам інтуїтивно зрозумілий інтерфейс та широкий набір функцій для ефективного та безпечного спілкування. Це включає обмін повідомленнями в реальному часі, обмін мультимедійними даними, автентифікацію користувачів та системи сповіщень. Інтегруючи ці функціональні можливості, програма-месенджер має на меті підвищити продуктивність та оптимізувати робочі процеси комунікації.

Вибір стеку MERN для розробки забезпечує безперебійну та послідовну роботу користувачів на різних платформах. Поєднання React для динамічних та інтерактивних користувацьких інтерфейсів, Express.js для серверної логіки та MongoDB для ефективного зберігання даних дозволяє створити всебічне та масштабоване рішення.

В цілому, розроблене програмне забезпечення має на меті зробити внесок у розробку програм-месенджерів, використовуючи можливості та найкращі практики стеку MERN, надаючи зручний інтерфейс та розширений функціонал.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

За останні роки світ додатків для обміну повідомленнями зазнав значного зростання і революціонізував комунікацію, захопивши людей, компанії та спільноти в усьому світі. Програми для обміну повідомленнями, такі як WhatsApp, Facebook Messenger і Telegram, стали потужними інструментами для миттєвого спілкування, полегшуючи розмови в режимі реального часу, обмін файлами та спільну роботу між користувачами.

Зі зростанням залежності від додатків для обміну повідомленнями зростає потреба в розробці комплексних та ефективних програм-месенджерів. Ці програми слугують важливими платформами для приватних осіб та організацій для безперешкодного зв'язку, обміну повідомленнями та співпраці. Використовуючи стек MERN (MongoDB, Express.js, React та Node.js), розробники можуть створювати багатофункціональні та масштабовані програми-месенджери, які задовольняють різноманітні комунікаційні потреби.

Програми-месенджери, розроблені з використанням MERN Stack, мають безліч переваг. MongoDB, база даних NoSQL, забезпечує гнучкість у зберіганні та пошуку повідомлень і даних користувача. Express.js, фреймворк веб-додатків для Node.js, дозволяє створювати надійні та безпечні внутрішні API для обробки повідомлень, автентифікації користувачів та управління даними. React, бібліотека JavaScript, полегшує розробку динамічних та інтерактивних користувацьких інтерфейсів, дозволяючи користувачам надсилати повідомлення, переглядати бесіди та ефективно керувати контактами. Node.js, середовище виконання, забезпечує роботу внутрішньої частини програми-месенджера, гарантуючи швидку та ефективну передачу повідомлень і обробку одночасних користувацьких з'єднань.

Природа програм обміну повідомленнями вимагає функціональності в реальному часі, щоб забезпечити миттєву доставку повідомлень і сповіщень. Впровадження веб-сокетів, таких як Socket.IO, у поєднанні зі стеком MERN дає можливість розробникам створювати додатки для обміну повідомленнями в реальному часі. Веб-сокети полегшують двосторонній зв'язок між сервером і клієнтом, забезпечуючи миттєве оновлення повідомлень і покращуючи користувацький досвід.

Крім того, безпека та конфіденційність є першочерговими міркуваннями при розробці програм-месенджерів. Впровадження протоколів шифрування, таких як SSL/TLS, та механізмів автентифікації, таких як JWT (JSON Web Tokens), підвищує конфіденційність даних і захищає від несанкціонованого доступу. Крім того, впровадження таких заходів, як шифрування повідомлень і безпечна автентифікація користувачів, допомагає захистити конфіденційну інформацію та забезпечити безпечне середовище обміну повідомленнями.

Програма-месенджер, розроблена з використанням стеку MERN, пропонує широкий спектр можливостей для покращення користувацького досвіду та продуктивності. Ці функції можуть включати синхронізацію повідомлень у реальному часі між кількома пристроями, підтримку мультимедійних повідомлень, керування контактами, пошук, архівування повідомлень і статус присутності користувача. Крім того, інтеграція push-сповіщень дозволяє користувачам залишатися в курсі нових повідомлень навіть тоді, коли програма-месенджер не є активно відкритою.

У наступних розділах ми заглибимося у процес проектування, розробки та реалізації програми-месенджера з використанням MERN Stack. Вивчаючи основні компоненти, архітектурні міркування та стратегії реалізації, ми прагнемо створити надійну та зручну програму-месенджер, яка забезпечить безперешкодне спілкування та співпрацю для окремих осіб та організацій.

1.2 Призначення розробки та галузь застосування

Метою розробки програми-месенджера з використанням стеку MERN є створення надійної та багатофункціональної комунікаційної платформи, яка забезпечує безперебійний обмін повідомленнями в режимі реального часу для приватних осіб та організацій. Месенджер має на меті надати користувачам надійний та ефективний інструмент для обміну миттєвими повідомленнями, обміну файлами та спільної роботи.

Сфера застосування програми-месенджера є широкою, орієнтованою на різноманітне коло користувачів і сценаріїв, включаючи:

Особисте спілкування: Програма-месенджер дозволяє людям зв'язуватися зі своїми друзями, родиною та колегами для особистих розмов. Вона надає такі функції, як обмін повідомленнями один на один, групові чати, обмін мультимедійними файлами та смайлики для покращення досвіду спілкування.

Професійне спілкування: Програма для обміну повідомленнями задовольняє потреби професіоналів та бізнесу в ефективному та безпечному спілкуванні. Він пропонує такі функції, як безпечний обмін повідомленнями, голосові та відеодзвінки, спільний доступ до екрану та спільна робота над документами, що дозволяє професіоналам ефективно спілкуватися та оптимізувати свої робочі процеси.

Командна співпраця: Програма-месенджер полегшує співпрацю між командами та проектними групами. Він надає такі функції, як групові чати, канали, управління завданнями та інтеграцію з інструментами підвищення продуктивності, такими як програмне забезпечення для управління проектами та платформи для зберігання файлів. Це сприяє ефективній командній роботі, комунікації та координації.

Залучення громади: Програма-месенджер може використовуватися спільнотами, групами за інтересами або організаціями для взаємодії зі своїми членами або аудиторією. Вона пропонує такі функції, як канали спільноти,

сповіщення про події, опитування та оголошення, що дозволяють лідерам спільнот спілкуватися зі своїми членами та сприяти їхній активній участі.

Розробка програми-месенджера з використанням MERN Stack має на меті задовольнити комунікаційні потреби різних користувачів та сценаріїв. Завдяки зручному інтерфейсу, можливостям обміну повідомленнями в реальному часі та забезпеченню безпеки і конфіденційності даних, програма-месенджер стає універсальним інструментом для спілкування та співпраці в особистому, професійному та громадському середовищі.

1.3 Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується з наказом ректора.

Таким чином підставами для розробки (виконанням кваліфікаційної роботи) є:

– освітня програма спеціальності 122 «Комп`ютерні Науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;

– завдання на кваліфікаційну роботу на тему «Розробка програми месенджера за допомогою MERN Stack».

1.4 Постановка завдання

Метою цього проекту є розробка програми-месенджера з використанням стеку MERN, яка надасть користувачам надійну та ефективну платформу для безперешкодного спілкування. Програма буде включати в себе різні функції,

такі як обмін повідомленнями в режимі реального часу, обмін мультимедійними даними та управління користувачами.

Основними мовами програмування та технологіями, які будуть використані при розробці цього месенджера, будуть:

MongoDB: технологія баз даних NoSQL буде використовуватися для зберігання профілів користувачів, розмов і даних повідомлень. Гнучкість і масштабованість MongoDB роблять її ідеальним вибором для роботи з динамічною природою додатків для обміну повідомленнями.

Express.js: Цей фреймворк для веб-додатків буде використано для побудови внутрішнього сервера, який обробляє запити API, автентифікацію та взаємодію з базами даних. Express.js пропонує простий та ефективний спосіб створення RESTful API та обробки HTTP-запитів.

React.js: Фронтенд програми-месенджера буде побудовано за допомогою React.js, популярної JavaScript-бібліотеки для побудови користувацьких інтерфейсів. React.js забезпечує багаторазове використання компонентів інтерфейсу та ефективний рендеринг, забезпечуючи чуйний та інтерактивний користувацький досвід.

Node.js: Середовище виконання буде засноване на Node.js, що дозволяє виконувати JavaScript на стороні сервера та полегшує розробку як фронтенд, так і бекенд компонентів за допомогою уніфікованої мови.

Програма-месенджер буде підтримувати ключові функціональні можливості, в тому числі:

Реєстрація та аутентифікація користувачів: Користувачі зможуть створювати облікові записи, безпечно входити в систему та керувати інформацією свого профілю.

Обмін повідомленнями в режимі реального часу: Користувачі матимуть можливість надсилати та отримувати миттєві повідомлення в режимі реального часу, що забезпечить безперешкодне та інтерактивне спілкування.

Обмін мультимедійними даними: Програма дозволить користувачам обмінюватися різними типами медіа, такими як зображення, відео та файли, покращуючи досвід спілкування.

Контакти та розмови користувачів: Користувачі зможуть керувати своїми контактами, створювати бесіди та брати участь як в індивідуальних, так і в групових чатах.

Сповіщення та оповіщення: Програма надаватиме механізми сповіщень для попередження користувачів про нові повідомлення, оновлення та інші важливі події.

1.5 Вимоги до програми або програмного виробу

1.5.1 Вимоги до функціональних характеристик

Програма-месенджер, розроблена з використанням стеку MERN, матиме низку функціональних характеристик для забезпечення ефективної та безперебійної комунікації. Очікується, що кінцевий продукт матиме наступні функціональні характеристики:

– Обмін повідомленнями в режимі реального часу: Програма-месенджер надаватиме можливості обміну повідомленнями в режимі реального часу, дозволяючи користувачам миттєво надсилати та отримувати повідомлення.

– Аутентифікація та авторизація користувачів: Програма включатиме автентифікацію користувачів для забезпечення безпечного доступу до платформи обміну повідомленнями. Також буде реалізована авторизація користувачів для управління ролями та дозволами користувачів.

– Індивідуальні та групові розмови: Користувачі зможуть вести індивідуальні бесіди, а також створювати групові бесіди з кількома учасниками та брати участь у них.

– Сповіщення про повідомлення: Програма надаватиме сповіщення про нові повідомлення, щоб користувачі були оперативно поінформовані про вхідні повідомлення.

– Форматування повідомлень: Користувачі матимуть можливість форматувати свої повідомлення, додаючи смайлики, прикріплюючи файли та використовуючи основні параметри форматування тексту, такі як жирний шрифт та курсив.

– Підтримка мультимедіа: Програма підтримуватиме обмін мультимедійним контентом, включаючи зображення, відео та аудіофайли, покращуючи досвід спілкування.

–Синхронізація повідомлень: Програма-месенджер забезпечить синхронізацію повідомлень на різних пристроях, що дозволить користувачам легко переключатися між пристроями, зберігаючи при цьому послідовну історію розмов.

1.5.2 Вимоги до інформаційної безпеки

Щоб забезпечити конфіденційність, цілісність і приватність даних користувачів і комунікацій у програмі-месенджері, необхідно дотримуватися наступних вимог інформаційної безпеки:

Наскрізне шифрування: Повідомлення, якими обмінюються користувачі, повинні бути зашифровані, щоб запобігти несанкціонованому доступу або перехопленню третіми особами.

Захист даних користувачів: Дані користувачів, включаючи особисту інформацію та вміст повідомлень, повинні надійно зберігатися та бути захищеними від несанкціонованого доступу або витоку даних.

Безпечна автентифікація та авторизація: Необхідно впровадити надійні механізми автентифікації для перевірки особи користувачів і забезпечення доступу до платформи обміну повідомленнями лише авторизованим особам.

Безпечна передача: Повідомлення та дані користувачів повинні передаватися захищеними каналами з використанням таких протоколів, як HTTPS, щоб запобігти підслуховуванню або несанкціонованому втручанню під час транзиту.

Безпечне зберігання мультимедійного контенту: Будь-який мультимедійний контент, яким обмінюються в програмі-месенджері, має бути надійно збережений і захищений, щоб запобігти несанкціонованому доступу або розповсюдженню.

Регулярні оновлення безпеки: Програма-месенджер повинна регулярно отримувати оновлення безпеки для усунення будь-яких потенційних вразливостей або нових загроз безпеці.

1.5.3 Вимоги до складу та параметрів технічних засобів

Для ефективного використання веб-трекера криптовалютного ринку необхідно дотримуватися певних технічних специфікацій та вимог. Ці специфікації охоплюють як апаратні, так і програмні компоненти, необхідні для безперебійної роботи. Нижче наведені рекомендовані вимоги до компонентів системи:

Веб-браузер: Для роботи веб-додатку потрібен сумісний веб-браузер. Рекомендується використовувати популярні браузери, сумісні з операційною системою Windows. Мінімальні специфікації для більшості браузерів включають

- Процесор: Pentium 4 або еквівалент;
- Графічний адаптер: nVidia, Intel, AMD/ATI 3D адаптер;
- Відеопам'ять: 128 МБ;
- Сховище для зберігання даних: Не менше 150 ГБ;
- Оперативна пам’ять: Мінімум 1 ГБ.

Мобільні платформи: Для користувачів, які отримують доступ до веб-трекера через мобільні пристрої, для оптимальної роботи потрібні певні версії операційних систем. Мінімально підтримувані версії

- iOS: Версія 13.0 і вище;
- Android: Версія 10 і вище.

Дотримання цих специфікацій забезпечує безперебійну роботу з сайтом, дозволяючи користувачам отримувати доступ до інформації про ринок криптовалют в режимі реального часу і ефективно відстежувати свої інвестиції на різних платформах і пристроях.

1.5.4 Вимоги до інформаційної та програмної сумісності

Програма-месенджер має бути сумісною з широким спектром пристроїв і платформ, щоб забезпечити широку доступність. Повинні бути дотримані наступні вимоги до сумісності:

Мобільна сумісність: Програма-месенджер повинна мати адаптивний дизайн, який підлаштовується під різні розміри та роздільну здатність екрану, забезпечуючи зручність користування на мобільних пристроях, таких як смартфони та планшети.

Сумісність з браузерами: Програма повинна бути сумісною з популярними веб-браузерами, включаючи Google Chrome, Mozilla Firefox, Safari та Microsoft Edge, забезпечуючи однакову роботу в різних браузерах.

Сумісність API: API програми має відповідати галузевим стандартам і найкращим практикам, щоб забезпечити інтеграцію зі сторонніми додатками або сервісами, розширюючи функціональність і сумісність.

Відповідаючи цим вимогам, програма-месенджер, розроблена з використанням MERN Stack, надасть користувачам безпечну, ефективну та зручну комунікаційну платформу, доступну на різних пристроях та платформах.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Функціональне призначення розробленої програми-месенджера полягає в тому, щоб надати користувачам надійну та зручну платформу для безперешкодного спілкування та співпраці. Використовуючи сучасні веб-технології, такі як стек MERN, додаток має на меті запропонувати наступні ключові функціональні можливості:

Реєстрація та аутентифікація користувачів: Програма-месенджер дозволить користувачам реєструватися та створювати особисті акаунти. Механізми аутентифікації забезпечать безпечний доступ до програми, захищаючи конфіденційність користувачів та їхні дані.

Обмін повідомленнями в режимі реального часу: Додаток забезпечить обмін повідомленнями в режимі реального часу, що дозволить користувачам миттєво обмінюватися повідомленнями. Ця функціональність забезпечить безперебійну та оперативну комунікацію.

Обмін мультимедійними даними: Користувачі зможуть обмінюватися різними типами мультимедійного контенту, такими як зображення, відео та документи. Ця функція розширить комунікаційні можливості програми-месенджера, підтримуючи насичену та динамічну взаємодію.

Групові чати та канали: Додаток підтримуватиме створення групових чатів та каналів, що дозволить користувачам співпрацювати та спілкуватися в командах або спільнотах. Ця функція сприятиме ефективним груповим дискусіям та координації проектів.

Сповіщення та оповіщення: Програма-месенджер надаватиме сповіщення та оповіщення, щоб тримати користувачів в курсі важливих повідомлень, згадок або активностей. Ця функція гарантує, що користувачі

залишатимуться в курсі подій і зможуть оперативно реагувати на відповідні обговорення.

Пошук та фільтрація: Додаток включатиме можливості пошуку та фільтрації, які допоможуть користувачам швидко знаходити певні розмови, контакти або повідомлення. Ця функціональність підвищить ефективність пошуку повідомлень та навігації всередині програми-месенджера.

Загалом, розроблений веб-месенджер пропонуватиме широкий спектр функціональних можливостей для задоволення комунікаційних потреб користувачів. Використовуючи можливості стеку MERN, додаток має на меті забезпечити надійну та ефективну платформу для безперешкодного обміну повідомленнями та спільної роботи.

2.2. Опис застосованих математичних методів

Проектування та розробка веб-месенджера з використанням стеку MERN не потребує застосування складних математичних методів чи алгоритмів. Основна увага при розробці цього додатку зосереджена на реалізації ефективного функціоналу обміну повідомленнями в реальному часі, а не на складних математичних розрахунках. Математичні операції, що застосовуються в програмі, в основному обертаються навколо базових арифметичних обчислень.

Хоча програма-месенджер може містити статистичні та аналітичні функції, які допомагають користувачам зрозуміти закономірності комунікації, ці функції не покладаються на складні математичні моделі. Основна мета - надати користувачам інтуїтивно зрозумілі інструменти для спілкування та співпраці, а не обширний математичний аналіз.

2.3. Опис використаної архітектури та шаблонів проектування

При розробці веб-месенджера з використанням MERNStack ми застосуємо комбінацію архітектурних та проектних патернів для забезпечення

ефективної та масштабованої роботи. Одним з ключових компонентів архітектури буде використання принципів REST (Representational State Transfer), HTTP-запитів та API.

REST-архітектура широко визнана як ефективний архітектурний стиль для проектування мережевих додатків. Вона забезпечує набір принципів, які сприяють масштабованості та вільному зв'язку всередині систем. У нашій програмі-месенджері ми будемо дотримуватися принципів RESTful архітектури, створюючи ресурсно-орієнтовані API та здійснюючи запити до цих API. Такий підхід сприятиме безшовній інтеграції з іншими системами та забезпечить стандартизований спосіб взаємодії з нашим месенджером. Фундаментальні принципи та характеристики REST включають в себе:

Клієнт-серверна архітектура: Наша програма-месенджер буде розділяти клієнтські та серверні компоненти, що уможливить незалежну розробку. Клієнт буде ініціювати запити до сервера, який буде обробляти ці запити та надсилати відповіді.

Уніфікований інтерфейс: Ми розробимо наші RESTful API, щоб дотримуватися узгодженого інтерфейсу для взаємодії. Це передбачає використання стандартних методів HTTP (таких як GET, POST, PUT, DELETE) для виконання різних операцій над ресурсами. Ми також будемо використовувати ідентифікатори ресурсів (URL) для унікальної ідентифікації ресурсів і стандартні медіа-типи (наприклад, JSON або XML) для представлення даних.[16]

Ресурсо-орієнтованість: Ресурси будуть фундаментальним поняттям у нашій програмі-месенджері. Ресурсом може бути будь-яка сутність, яку можна ідентифікувати та маніпулювати нею в системі, наприклад, користувачі, історії чатів або інші релевантні дані. Кожен ресурс матиме унікальний ідентифікатор (URI/URL), а наш API надаватиме операції для виконання дій над цими ресурсами.

Зв'язок без статусу: Кожен запит від клієнта до сервера буде містити всю необхідну інформацію для розуміння та обробки сервером. Сервер не

зберігатиме жодної клієнтської інформації між запитами, що сприяє масштабованості та спрощує реалізацію сервера.

RESTful API зазвичай використовуються для створення веб-сервісів, до яких можуть мати доступ різні клієнти, включаючи веб-браузери, мобільні додатки та інші сервери. Вони пропонують гнучкий і масштабований підхід до надання даних і функціональності через Інтернет у стандартизований і сумісний спосіб.[15]

У контексті нашої програми-месенджера ми можемо використовувати RESTful API для доступу та отримання даних, пов'язаних з інформацією про користувачів, історіями чатів та іншими відповідними ресурсами. Ми розробимо кінцеві точки API, які представлятимуть різні ресурси в системі, дозволяючи клієнтам взаємодіяти з цими кінцевими точками, надсилаючи відповідні HTTP-запити (наприклад, GET, POST, PUT, DELETE) для виконання операцій над ресурсами та отримання необхідних даних.

HTTP (Hypertext Transfer Protocol - протокол передачі гіпертексту) визначає набір методів запитів або HTTP-дієслів, які вказують на бажану дію, що має бути виконана на даному ресурсі. Ці методи HTTP забезпечують стандартизований спосіб взаємодії клієнтів із серверами та виконання різних операцій. Ось найпоширеніші методи HTTP-запитів:[17]

- GET: метод GET буде використовуватися для отримання представлення ресурсу з сервера. Це безпечний і недієздатний метод, що означає, що декілька ідентичних GET-запитів повинні мати той самий ефект, що й один запит. Відповідь зазвичай містить запитуваний ресурс у тілі повідомлення.
- POST: метод POST використовується для передачі даних, які мають бути оброблені певним ресурсом. Він часто використовується для створення нових ресурсів на сервері. На відміну від GET, POST не є ідемпотентним, що означає, що декілька однакових POST-запитів можуть мати різні результати. Передані дані будуть включені в тіло повідомлення.

- PUT: метод PUT використовується для оновлення існуючого ресурсу на сервері. Він замінює все представлення ресурсу корисним навантаженням запиту. Подібно до GET, PUT не має повноважень, що гарантує, що декілька ідентичних запитів PUT матимуть такий самий ефект, як і один запит.
- DELETE: метод DELETE використовується для видалення вказаного ресурсу на сервері. Він видаляє ресурс, ідентифікований URL-адресою запиту. DELETE також є ідемпотентним, тобто декілька однакових запитів DELETE матимуть той самий ефект, що і один запит.
- PATCH: метод PATCH використовується для часткового оновлення існуючого ресурсу. Він схожий на PUT, але замість того, щоб замінити весь ресурс, PATCH оновлює тільки певні поля або властивості ресурсу.
- HEAD: метод HEAD схожий на GET, але він отримує тільки заголовки відповіді, без тіла повідомлення. Він часто використовується для отримання метаданих або перевірки стану ресурсу.
- OPTIONS: метод OPTIONS використовується для отримання підтримуваних параметрів зв'язку для даного ресурсу або сервера. Він може бути використаний для визначення доступних методів запиту, заголовків або інших можливостей.

API (Application Programming Interface - інтерфейс прикладного програмування) - це набір правил і протоколів, який дозволяє різним програмним додаткам спілкуватися і взаємодіяти один з одним. Він визначає методи та формати даних, що використовуються для обміну інформацією та виконання операцій між різними системами або компонентами.[13]

API можуть слугувати різним цілям, зокрема:

Інтеграція: API дозволяють різним програмним системам з'єднуватися та обмінюватися даними. Вони дозволяють програмам обмінюватися

інформацією, отримувати доступ до ресурсів і виконувати дії на різних платформах. Наприклад, платформи соціальних мереж часто надають API, які дозволяють розробникам інтегрувати свої додатки з такими функціями, як автентифікація, публікація контенту або отримання інформації про користувача.

Розробка: API зазвичай використовуються розробниками як будівельні блоки для створення нових додатків. Надаючи набір чітко визначених методів і структур даних, API надають розробникам необхідні інструменти для взаємодії з певною програмною системою або сервісом. Це прискорює процес розробки і дозволяє розробникам використовувати існуючу функціональність, а не починати з нуля.

Для зв'язку із зовнішніми API та отримання актуальних даних використовувалися HTTP-запити (Hypertext Transfer Protocol). Веб-трекер криптовалютного ринку надсилав HTTP-запити до кінцевих точок API різних криптовалютних бірж або постачальників ринкових даних. Ці запити, як правило, робилися з використанням методу GET для отримання такої інформації, як ціни на криптовалюту, обсяги торгів і ринкові тенденції.[17]

Принципи RESTful-дизайну забезпечили чітку структуру запитів до API, а HTTP-запити дозволили безперешкодно інтегруватися із зовнішніми джерелами даних.

2.4. Опис використаних технологій та мов програмування

В розробці цього проекту були використані такі технології та мови:

- Java-Script;
- React;
- Axios;
- Material-UI;
- React-Router-DOM;
- Socket.io;
- MongoDB.

JavaScript - це високорівнева інтерпретована мова програмування, яка використовується переважно для розробки інтерфейсів. Це універсальна мова, яка також може бути використана для бекенд-розробки, розробки мобільних додатків, ігор тощо. Ось кілька ключових моментів про JavaScript:[18-20]

Крім того, JavaScript має широку підтримку в усіх основних веб-браузерах, включаючи Chrome, Firefox, Safari та Edge. Ця кросбраузерна сумісність гарантує, що веб-додатки на JavaScript можуть безперешкодно працювати на різних платформах і пристроях, забезпечуючи універсальну мову для веб-розробки.

Поєднання можливостей JavaScript маніпулювати DOM та обробляти події користувача, а також його кросбраузерна сумісність роблять його важливою мовою для створення динамічних та інтерактивних веб-додатків, які задовольняють потреби широкого кола користувачів.

Об'єктно-орієнтоване програмування (ООП): JavaScript підтримує принципи об'єктно-орієнтованого програмування, такі як інкапсуляція, успадкування та поліморфізм. Це дозволяє розробникам створювати багаторазовий і модульний код, визначаючи класи, об'єкти та методи.

Асинхронне програмування: JavaScript підтримує асинхронне програмування за допомогою таких функцій, як callbacks, promises, та async/await. Це дозволяє обробляти трудомісткі операції, такі як мережеві запити або доступ до баз даних, не блокуючи виконання іншого коду.

Бібліотеки та фреймворки: JavaScript має величезну екосистему бібліотек і фреймворків, які спрощують і покращують веб-розробку. Популярні фреймворки, такі як React.js, Angular та Vue.js, надають потужні інструменти для створення складних веб-додатків з багаторазовими компонентами та ефективним рендерингом.

Спільнота та підтримка: JavaScript має велику та активну спільноту розробників, які сприяють його розвитку та еволюції. Онлайн-ресурси, форуми та спільноти надають широку документацію, навчальні посібники та підтримку для розробників усіх рівнів кваліфікації.

JavaScript - це потужна і широко використовувана мова програмування, яка зробила революцію у веб-розробці. Її універсальність, сумісність з багатьма браузерами та розгалужена екосистема роблять її важливим інструментом для створення інтерактивних та динамічних веб-додатків. Незалежно від того, чи це маніпулювання елементами веб-сторінки, обробка взаємодії з користувачем або створення складних серверних систем, JavaScript продовжує залишатися на передньому краї сучасної розробки програмного забезпечення.

React - це популярна бібліотека JavaScript, що використовується для створення користувацьких інтерфейсів (UI) у веб-додатках. Розроблена та підтримувана компанією Facebook, React набула широкого розповсюдження завдяки своїй компонентній архітектурі, маніпуляціям з віртуальною DOM (Document Object Model) та ефективним можливостям рендерингу. Ось деякі ключові аспекти React:[1-5]

Компонентна архітектура: React дотримується компонентного підходу, де інтерфейс користувача розбивається на незалежні компоненти, які можна використовувати повторно. Кожен компонент інкапсулює власну логіку, стан та рендеринг, що полегшує створення складних користувацьких інтерфейсів та сприяє повторному використанню коду.

Віртуальний DOM: React використовує віртуальний DOM, який слугує полегшеним представленням реального DOM. Порівнюючи віртуальний DOM з реальним DOM, React може ефективно визначити мінімальні необхідні зміни та оновити лише необхідні компоненти. Така оптимізація мінімізує кількість перезавантажень браузера, що призводить до покращення продуктивності та більш плавного користувацького досвіду.

Синтаксис JSX: React представляє JSX, розширення до JavaScript, яке дозволяє розробникам писати HTML-подібний синтаксис у JavaScript-коді. JSX надає стислий та декларативний спосіб опису структури та зовнішнього вигляду компонентів інтерфейсу користувача. Це дозволяє розробникам легко

поєднувати логіку JavaScript та розмітку інтерфейсу, покращуючи читабельність коду та зручність його супроводу.

Методи життєвого циклу компонентів: React пропонує набір методів життєвого циклу, які дозволяють розробникам підключатися до різних етапів життєвого циклу компонента. Ці методи включають ініціалізацію, монтування, оновлення та демонтаж компонентів. Використовуючи методи життєвого циклу, розробники можуть виконувати певні дії у відповідні моменти життя компонента, наприклад, встановлювати початковий стан, отримувати дані або очищати ресурси, коли компонент видаляється з DOM.

React Native: React виходить за рамки веб-розробки і також використовується для розробки мобільних додатків завдяки React Native. React Native дозволяє створювати крос-платформні мобільні додатки з використанням React та JavaScript, орієнтовані на платформи iOS та Android з єдиною кодовою базою.[7]

Велика та активна спільнота: React має велику та енергійну спільноту розробників, які активно сприяють його розвитку. Спільнота надає обширну документацію, навчальні посібники та бібліотеки з відкритим кодом, які підвищують продуктивність розробки та відповідають різним варіантам використання.

Сила React полягає в його здатності створювати інтерактивні та адаптивні інтерфейси, ефективно оновлювати компоненти та полегшувати повторне використання коду. Його популярність та підтримка спільноти призвели до створення численних сторонніх бібліотек, інструментів та інтеграцій, що робить React універсальним вибором для створення сучасних веб-додатків.

Axios - це широко використовувана бібліотека JavaScript, яка пропонує простий та інтуїтивно зрозумілий API для виконання HTTP-запитів з веб-браузерів або Node.js. Це популярний вибір для обробки AJAX-запитів та взаємодії з API завдяки своїй простоті та універсальності. Ось кілька ключових деталей про Axios:

HTTP-запити: Axios підтримує поширені методи HTTP, такі як GET, POST, PUT, DELETE та інші. Це дозволяє розробникам легко робити запити до серверів і отримувати дані з API.

На основі обіцянок: Axios побудований на JavaScript Promises, які забезпечують елегантний спосіб обробки асинхронних операцій. Axios повертає обіцянки при виконанні запитів, що дозволяє розробникам використовувати методи then() і catch() для структурованої обробки відповідей на успіх і помилки.

Конфігурація запитів: Axios дозволяє розробникам конфігурувати кожен запит за допомогою різних параметрів, включаючи заголовки, параметри запиту, тіло запиту, токени автентифікації тощо. Ці параметри можуть бути передані як об'єкт при створенні запиту, що забезпечує гнучкість і можливість кастомізації.

Перехоплювачі: Axios надає перехоплювачі, які дозволяють розробникам перехоплювати і змінювати запити або відповіді до того, як вони будуть оброблені. Перехоплювачі корисні для таких завдань, як додавання загальних заголовків, обробка автентифікації, ведення журналів або послідовна обробка помилок у декількох запитах.

Обробка відповідей: Axios автоматично аналізує дані відповіді на основі типу контенту і надає їх в об'єкті відповіді. Він підтримує ряд форматів даних, включаючи JSON, XML, HTML та інші. Розробники можуть отримати доступ до статусу відповіді, заголовків і даних, використовуючи властивості об'єкта відповіді.

Скасування запиту: Axios підтримує скасування запитів, що дозволяє розробникам скасовувати поточні запити, якщо вони більше не потрібні. Ця функція особливо корисна, коли користувач переходить зі сторінки або перериває запит вручну.

Спільне використання ресурсів різного походження (CORS): Axios за замовчуванням обробляє Cross-Origin Resource Sharing (CORS), що дозволяє розробникам робити запити до різних доменів або субдоменів, не стикаючись

з проблемами перехресного походження. Він включає необхідні заголовки для CORS і автоматично керує попередніми запитами.

Інтеграція з відомими фреймворками: Axios часто використовується з популярними JavaScript-фреймворками, такими як React, Vue.js та Angular. Його можна легко інтегрувати в ці фреймворки для обробки даних та взаємодії з API.

Загалом, Axios спрощує процес створення HTTP-запитів, надаючи чистий API, надійну обробку помилок, синтаксис на основі обіцянок і підтримку розширених функцій, таких як перехоплювачі та скасування запитів. Його популярність зумовлена простотою використання, гнучкістю та сумісністю з різними JavaScript-фреймворками та середовищами.

Material-UI - це широко використовувана бібліотека з відкритим вихідним кодом, яка надає набір попередньо розроблених компонентів і стилів для побудови користувацьких інтерфейсів у React-додатках. Вона заснована на принципах Material Design від Google, які сприяють створенню чистого, сучасного та візуально привабливого дизайну. [20]

React Router DOM - це популярна бібліотека, яка надає можливості маршрутизації для React-додатків. Вона дозволяє створювати динамічні та інтерактивні односторінкові додатки (SPA), забезпечуючи навігацію між різними поданнями або компонентами на основі URL-адреси.

Firebase - це комплексна платформа для мобільної та веб-розробки від Google. Вона пропонує набір хмарних сервісів та інструментів, які дозволяють розробникам швидко та ефективно створювати та розгортати високоякісні додатки. Firebase надає різні сервіси, але в кваліфікаційній роботі я зосередився на базі даних Firebase та аутентифікації.[8-12]

База даних MongoDB:

MongoDB - MongoDB - це масштабована та орієнтована на документи база даних NoSQL, яка забезпечує гнучкий та безсхематичний підхід до зберігання даних. Ось ключові особливості та функціональність MongoDB:

Документно-орієнтована: MongoDB зберігає дані у вигляді гнучких і самоописових JSON-подібних документів, які називаються BSON (Binary JSON). Це дозволяє розробникам зберігати, запитувати та маніпулювати даними у спосіб, що відповідає структурі їхніх додатків.

Масштабованість і продуктивність: MongoDB призначена для роботи з великими обсягами даних і додатками з високим трафіком. Вона пропонує горизонтальну масштабованість завдяки шардингу та наборам реплік, що дозволяє додаткам безперешкодно масштабуватися зі збільшенням обсягу даних та трафіку. MongoDB також надає потужні можливості запитів та індексування для ефективного пошуку даних.

Синхронізація даних у режимі реального часу: MongoDB можна використовувати в поєднанні з іншими технологіями, такими як Change Streams та WebSockets, для досягнення синхронізації даних в реальному часі між пристроями або клієнтами. Це дозволяє співпрацювати в режимі реального часу і гарантує, що зміни, внесені в базу даних, негайно відображаються у всіх підключених додатках.

Гнучке моделювання даних: Гнучка структура даних MongoDB дозволяє розробникам адаптувати свої моделі даних по мірі розвитку додатків без необхідності складної та дорогої міграції. Ця гнучкість особливо корисна у швидкозмінних середовищах розробки, де вимоги можуть часто змінюватися.

Безпека та контроль доступу: MongoDB надає надійні засоби безпеки, включаючи механізми автентифікації та авторизації. Розробники можуть визначати контроль доступу та ролі користувачів для забезпечення безпеки даних і впровадження суворих політик доступу.

Інтеграція з іншими технологіями: MongoDB добре інтегрується з різними технологіями та фреймворками, які зазвичай використовуються у веб-розробці, що дозволяє розробникам використовувати її можливості в рамках свого улюбленого технологічного стеку.

Таким чином, MongoDB пропонує потужну і гнучку альтернативу СУБД Firebase для вашої кваліфікаційної роботи. Її документно-орієнтований підхід,

масштабованість, синхронізація даних в режимі реального часу і сильні функції безпеки роблять її підходящим вибором для зберігання і управління даними у вашому веб-додатку. Крім того, сумісність MongoDB з популярними технологіями веб-розробки дозволяє легко інтегрувати її у ваш улюблений технологічний стек.

Socket.IO - це широко використовувана бібліотека JavaScript, яка забезпечує двосторонній зв'язок між веб-клієнтами та серверами в режимі реального часу. Вона забезпечує безперебійний та ефективний спосіб створення додатків, які потребують миттєвої передачі даних та зв'язку між декількома клієнтами. Ось ключові особливості та функціональні можливості Socket.IO:

Двонаправлений зв'язок у реальному часі: Socket.IO встановлює постійне з'єднання між клієнтом і сервером, забезпечуючи комунікацію в режимі реального часу на основі подій. Це дозволяє як серверу, так і клієнту миттєво надсилати та отримувати дані, полегшуючи роботу спільних додатків, чат-систем, аналітики в реальному часі тощо.

WebSocket та резервні варіанти: Socket.IO використовує WebSocket, комунікаційний протокол, який забезпечує повнодуплексне з'єднання з низькою затримкою між клієнтом і сервером. WebSocket підтримується більшістю сучасних браузерів. У випадках, коли WebSocket не підтримується, Socket.IO граціозно повертається до альтернативних транспортних механізмів, таких як довге опитування або AJAX, забезпечуючи сумісність з широким спектром пристроїв і браузерів.

Архітектура, керована подіями: Socket.IO працює на основі архітектури, керованої подіями. Це дозволяє розробникам визначати власні події та обробники подій як на стороні клієнта, так і на стороні сервера. Такий підхід спрощує комунікаційний потік і дозволяє розробникам створювати додатки, які миттєво реагують на певні події або тригери.

Простори та імена: Socket.IO надає концепцію кімнат та просторів імен, що дозволяє розробникам групувати клієнтів на основі спільних критеріїв або

інтересів. Кімнати дозволяють відправляти цільові повідомлення та ефективно транслювати події певним групам клієнтів. Простори імен допомагають відокремити різні частини програми, забезпечуючи модульну та організовану структуру.

Масштабованість та інтеграція: Socket.IO можна масштабувати для обробки великої кількості одночасних з'єднань завдяки використанню розподіленої архітектури, балансувальників навантаження та методів кластеризації. Він легко інтегрується з популярними веб-фреймворками та технологіями, включаючи Node.js, Express та інші, що робить його універсальним та адаптованим до різних середовищ розробки.

Спільнота та підтримка: Socket.IO користується перевагами активної спільноти розробників, що підтримує його. Спільнота надає вичерпну документацію, навчальні посібники та ресурси, щоб допомогти розробникам розпочати роботу, вирішити проблеми та використати весь потенціал Socket.IO.

Отже, Socket.IO - це потужна бібліотека JavaScript, яка спрощує двосторонній зв'язок в режимі реального часу між веб-клієнтами та серверами. Завдяки підтримці WebSocket та резервних варіантів, архітектурі, керованій подіями, кімнатам та просторам імен, обробці помилок та повторному підключенню, масштабованості та потужній підтримці спільноти, Socket.IO дає можливість розробникам створювати інтерактивні додатки в режимі реального часу, які потребують миттєвої передачі даних та співпраці.

2.5. Опис структури програми та алгоритмів її функціонування

Для наочного зображення алгоритмів, що використовуються у MERN месенджері, було використано UML-діаграму. Ця діаграма відображає взаємодію користувача з інтерфейсом застосунку. Як видно з рисунку 2.1, взаємодія користувача включає в себе різні дії, такі як пошук користувачив,

налаштування облікового запису та відправка буд`якого типу повідомлень. Така взаємодія дозволяє користувачам ефективно використовувати месенджер як для ділових, так і для особистих цілей.

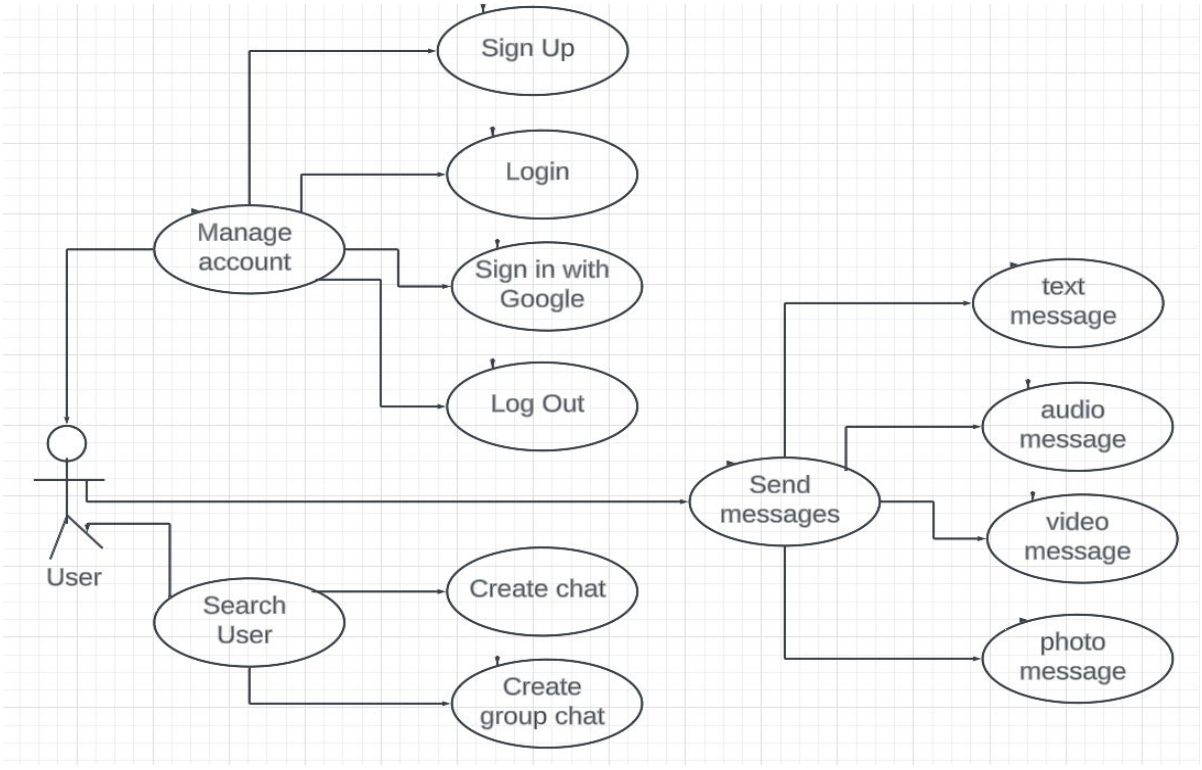

Рис. 2.1. UML-діаграма варіантів використання

При розробці цього месенджеру було використано компонентний підхід, що є фундаментальним аспектом розробки на React. У цьому підході кожен елемент, присутній на веб-сторінці, розглядається як окремий компонент, що дозволяє повторно використовувати його в різних частинах проекту. Важливо, що ці компоненти є незалежними сутностями, що полегшує управління та обслуговування.

Прийняття компонентного підходу в React не тільки рекомендується, але й підкреслюється в офіційній документації. Цей підхід значно полегшує процес розробки, пропонуючи масштабованість. Компоненти можуть варіюватися від невеликих і тривіальних до великих розділів, блоків або навіть цілих сторінок, що робить його дуже гнучким і адаптованим до вимог проекту.

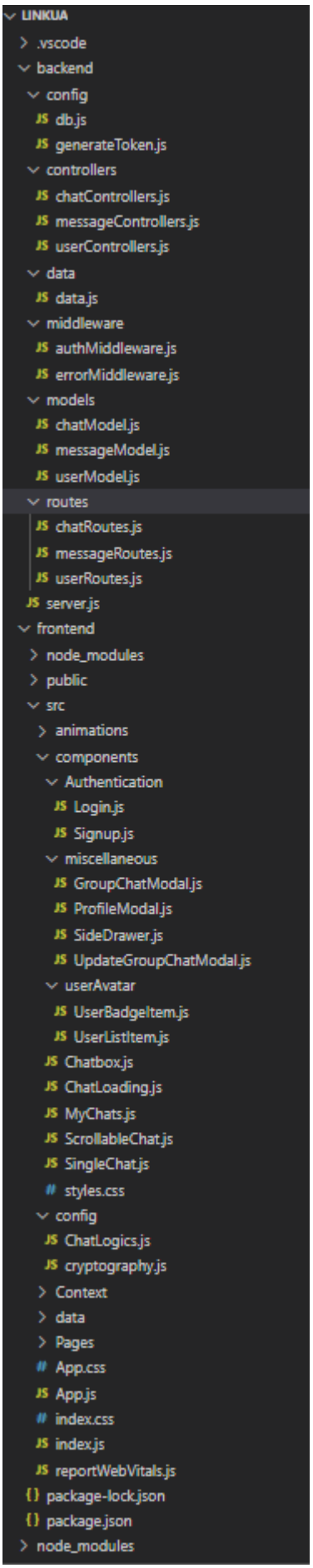

Рис. 2.2. Перелік файлів проекту

2.6. Обґрунтування та організація вхідних та вихідних даних програми

У веб-месенджері, розробленому з використанням MERNStack, обмін даними також може здійснюватися у форматі JSON, який добре підходить для обробки як на стороні сервера, так і на стороні клієнта.

Для цього проекту вхідні дані охоплюватимуть різні аспекти, пов'язані з функціональністю месенджера. Користувачі зможуть реєструватися і створювати персоналізовані акаунти, використовуючи свої електронні адреси. Створення облікового запису надасть їм доступ до додаткових функцій і забезпечить більш персоналізований досвід роботи в месенджері. Крім того, користувачі можуть взаємодіяти з месенджером, надсилаючи та отримуючи повідомлення, створюючи чат-групи та керуючи своїм списком контактів.

На виході програма-месенджер надасть користувачам можливість безперешкодного спілкування та обміну повідомленнями. Користувачі зможуть надсилати та отримувати повідомлення в режимі реального часу, переглядати історію чату та ефективно керувати своїми контактами. Програма також може включати в себе такі функції, як сповіщення про повідомлення, пошук повідомлень та можливість обмінюватися медіа-файлами.

Всі ці функції спрямовані на те, щоб задовольнити потреби людей, які шукають надійний та ефективний засіб спілкування. Надаючи зручний інтерфейс, можливості обміну повідомленнями в режимі реального часу та різні корисні функції, програма-месенджер має на меті покращити загальний користувацький досвід та сприяти ефективному спілкуванню між користувачами.

Важливо зазначити, що згадані вхідні та вихідні дані базуються на загальній функціональності програми-месенджера. Ви можете додатково налаштувати та вдосконалити ці вхідні та вихідні дані, виходячи з ваших конкретних вимог та цілей розробки програми-месенджера за допомогою MERNStack.

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Для комфортної роботи з сучасними веб-браузерами розробники рекомендують такі мінімальні вимоги до комп'ютера:

1. Процесор (CPU): двоядерний процесор (наприклад, Intel Core i3) або еквівалент.
2. Графічний адаптер: інтегрована графіка або спеціальна відеокарта з оновленими драйверами.
3. Оперативна пам'ять: щонайменше 4 ГБ оперативної пам'яті для безперебійної роботи в багатозадачному режимі та керування вкладками.
4. Сховище: достатньо вільного місця для зберігання інсталяцій браузера та тимчасових файлів.
5. Операційна система: остання версія підтримуваної операційної системи (наприклад, Windows 10, macOS, Linux).
6. Дисплей: монітор з роздільною здатністю 1280x800 пікселів або вище для кращого перегляду.
7. Підключення до Інтернету: стабільне та надійне з'єднання з інтернетом для перегляду та доступу до веб-контенту.

Ось специфікації обладнання, яке використовувалось під час створення кваліфікаційної роботи:

1. Процесор (CPU): Intel Core I5 5600K.
2. Графічний адаптер: GTX 1060.
3. Відеопам'ять: 6 ГБ.
4. Сховище: накопичувач на 500 ГБ.
5. Оперативна пам’ять: 8 ГБ.
6. Периферійні пристрої: клавіатура, миша та монітор.

2.7.2. Використані програмні засоби

Під час розробки проекту для кваліфікаційної роботи були використані наступні програмні засоби:

- Visual Studio Code
- Node.js
- Git

Visual Studio Code (VS Code) - це легкий, універсальний і широко використовуваний редактор вихідного коду, розроблений компанією Microsoft. Він є безкоштовним і має відкритий вихідний код, надаючи розробникам потужний інструмент для написання, редагування та налагодження коду на різних мовах програмування та платформах. Ось деякі ключові особливості та функціональні можливості Visual Studio Code:[21]

Крос-платформенна сумісність: Visual Studio Code (VS Code) - це універсальний редактор коду, доступний для операційних систем Windows, macOS та Linux. Ця крос-платформенна підтримка дозволяє розробникам безперешкодно працювати в різних середовищах і співпрацювати з членами команди, які використовують різні операційні системи.

Інтуїтивно зрозумілий інтерфейс: VS Code може похвалитися чистим і сучасним користувацьким інтерфейсом, який підкреслює простоту і продуктивність. Інтерфейс легко налаштовується, що дозволяє розробникам пристосувати середовище кодування до своїх уподобань. Такі функції, як розділене представлення, настроювані макети та міні-карти, покращують навігацію коду та його читабельність, сприяючи тому, щоб ви не відволікалися під час кодування.

Широка мовна підтримка: Visual Studio Code підтримує широкий спектр мов програмування з коробки. Вона забезпечує підсвічування синтаксису, завершення коду (IntelliSense) та мовні розширення для таких популярних мов, як JavaScript, Python, C++, Java, HTML, CSS та інших. Така широка мовна

підтримка дозволяє розробникам працювати над різноманітними проектами без необхідності перемикатися між різними редакторами.

Розширення та маркетплейс: VS Code має динамічну екосистему розширень, створених спільнотою, які доступні через маркетплейс VS Code. Ці розширення розширюють функціональність редактора, надаючи фрагменти коду, лінчери, інструменти для налагодження, інтеграцію з системами контролю версій та підтримку різних фреймворків і бібліотек. Маркетплейс дозволяє легко знаходити, встановлювати та керувати розширеннями безпосередньо з редактора.

Вбудований термінал: Visual Studio Code містить вбудований термінал, що дозволяє розробникам виконувати команди і запускати скрипти без необхідності використання зовнішнього вікна терміналу. Цей інтегрований термінал підвищує ефективність робочого процесу і полегшує безперешкодне виконання команд і налагодження в самому редакторі.

Інтеграція з Git: VS Code пропонує безшовну інтеграцію з системою контролю версій Git. Вона надає надійні функції для управління вихідним кодом, візуалізації гілок, вивчення історії комітів та порівняння відмінностей. Ця інтеграція спрощує спільну роботу, перегляд коду та контроль версій для розробників, які працюють з репозиторіями Git.

Можливості налагодження: Visual Studio Code надає потужні вбудовані засоби налагодження для багатьох мов програмування. Розробники можуть встановлювати точки зупинки, переходити по коду, перевіряти змінні та ефективно налагоджувати свої програми. Крім того, розширення для конкретних мов покращують процес налагодження, забезпечуючи адаптовану підтримку налагодження для конкретних мов і фреймворків.

Розширюваність та кастомізація: Розширюваність VS Code є визначною особливістю. Розробники можуть налаштувати майже кожен аспект редактора, від тем та іконок до комбінацій клавіш та налаштувань. Такий рівень кастомізації дозволяє розробникам персоналізувати своє середовище

кодування відповідно до своїх уподобань, оптимізувати робочі процеси та створити індивідуальний досвід розробки.

Таким чином, Visual Studio Code пропонує багатофункціональний і гнучкий інструмент для редагування коду, який легко налаштовується. Завдяки крос-платформенній сумісності, інтуїтивно зрозумілому інтерфейсу, широкій мовній підтримці, розвиненій екосистемі розширень, інтегрованому терміналу, інтеграції з Git'ом, можливостям налагодження та широким можливостям кастомізації, VS Code дозволяє розробникам бути продуктивними та ефективними у своїй роботі з кодуванням

Node.js - це потужне і широко використовуване середовище виконання з відкритим вихідним кодом, яке дозволяє розробникам запускати JavaScript-код на стороні сервера. Воно побудоване на JavaScript-движку V8, тому самому, на якому працює браузер Google Chrome, і забезпечує керовану подіями, неблокуючу модель вводу/виводу, що робить його високоефективним і масштабованим для створення мережевих додатків.

Ось деякі ключові аспекти та особливості Node.js:[22]

JavaScript на стороні сервера: Node.js дозволяє розробникам писати серверні додатки за допомогою JavaScript, мови, яка традиційно асоціюється з клієнтськими скриптами. Ця уніфікація мови програмування дозволяє розробникам використовувати одну і ту ж мову і кодову базу як на стороні клієнта, так і на стороні сервера, що сприяє повторному використанню коду, узгодженості та підвищенню продуктивності.

Асинхронний та неблокуючий ввід/вивід: Однією з ключових переваг Node.js є його асинхронна, неблокуюча модель вводу/виводу. Замість того, щоб блокувати і чекати завершення операцій вводу/виводу, Node.js використовує архітектуру, керовану подіями, і зворотні виклики для асинхронної обробки операцій вводу/виводу. Такий підхід дозволяє Node.js ефективно керувати паралельними з'єднаннями та обробляти велику кількість запитів, не блокуючи виконання інших завдань. Він особливо добре підходить для додатків

реального часу, потокового передавання даних та сценаріїв з високим рівнем паралелізму.

Масштабованість і продуктивність: Неблокуюча модель вводу/виводу Node.js в поєднанні з архітектурою, керованою подіями, робить його дуже масштабованим і продуктивним. Він може обробляти тисячі одночасних з'єднань з мінімальним споживанням ресурсів, що робить його чудовим вибором для створення масштабованих веб-додатків, API та мікросервісів. Легка та ефективна природа Node.js забезпечує високу пропускну здатність та низьку затримку відповідей, що сприяє підвищенню продуктивності.

Широка екосистема пакетів: Node.js виграє від розгалуженої екосистеми пакетів, доступної через реєстр npm (Node Package Manager). npm пропонує величезну колекцію модулів та бібліотек з відкритим кодом, які розробники можуть легко інтегрувати у свої додатки. Ця багата екосистема прискорює розробку, надає готові до використання рішення для різних функціональних можливостей і сприяє співпраці між розробниками.

Підтримка веб-протоколів: Node.js надає вбудовану підтримку різних веб-протоколів, включаючи HTTP, HTTPS, WebSocket та TCP/IP. Ця вбудована підтримка дозволяє розробникам без особливих зусиль створювати веб-сервери, RESTful API, додатки для чату в реальному часі, потокові сервери та інші мережеві додатки. Можливість роботи з цими протоколами спрощує процес розробки та покращує інтероперабельність.

Розширюваність: Node.js пропонує C++ API, який дозволяє розробникам створювати власні доповнення та розширювати функціональність Node.js. Ця можливість забезпечує інтеграцію з існуючими бібліотеками та системами C/C++, розширюючи можливості та універсальність Node.js. Розробники можуть використовувати Node.js як платформу для створення додатків, які інтегруються з застарілими системами або використовують нативний код для специфічних вимог.

Git - це широко використовувана розподілена система контролю версій, яка дозволяє розробникам відстежувати зміни, співпрацювати над проектами

та ефективно керувати вихідним кодом. Розроблений Лінусом Торвальдсом, творцем Linux, Git відомий своєю швидкістю, гнучкістю та надійністю. Він став важливим інструментом для розробки програмного забезпечення і використовується окремими особами та командами всіх розмірів.

Ось деякі ключові аспекти та особливості Git'у:[23-24]

Розподілений контроль версій: Git - це потужна розподілена система контролю версій, яка дозволяє кожному розробнику мати повну копію сховища, включаючи всю його історію. Така децентралізація пропонує кілька переваг, таких як можливість працювати в автономному режимі, підвищена швидкість і безперешкодна співпраця. Розробники можуть вносити локальні комміти, а потім синхронізувати свої зміни з віддаленими репозиторіями.

Розгалуження та злиття: Git надає надійні можливості розгалуження та злиття, що дозволяє розробникам створювати легкі гілки для незалежної роботи над конкретними функціями або виправленнями помилок. Гілки дозволяють здійснювати паралельну розробку, не впливаючи на основну кодову базу. Після завершення змін гілки можуть бути об'єднані назад в основну гілку, що полегшує інтеграцію нових функцій або виправлення помилок.

Історія коммітів та відстеження: Git зберігає детальну історію коммітів, яка відстежує кожну зміну, внесену до сховища. Кожна фіксація являє собою знімок проекту в певний момент часу. Ця повна історія коммітів дозволяє розробникам легко переглядати, порівнювати і повертатися до попередніх версій коду. Git також збирає метадані, такі як авторство, мітки часу і повідомлення про комміти, забезпечуючи повну і відстежувану історію проекту.

Вибіркова стадія та фіксація: Git дозволяє розробникам вибірково стадіювати зміни перед тим, як фіксувати їх. Такий поетапний підхід забезпечує точний контроль над тим, які зміни включати до кожного коміту. Розробники можуть розбивати логічні зміни на менші, сфокусовані коміти, що полегшує перегляд, повернення або застосування певних наборів змін

незалежно. Такий дрібнозернистий контроль підвищує якість коду, полегшує перевірку коду і покращує загальний процес розробки.

Розподілений контроль версій, можливості розгалуження та об'єднання, історія та відстеження коммітів, співпраця через віддалені сховища, а також вибіркові опції стадій та коммітів Git'у дають розробникам можливість ефективно керувати своєю кодовою базою, безперешкодно співпрацювати, ефективно відстежувати зміни та зберігати повну та організовану історію своїх проектів.

2.7.3. Виклик та завантаження програми

Розроблений веб-трекер ринку криптовалют доступний в Інтернеті за стандартизованою адресою (URL: http://localhost:3000) за допомогою сучасних браузерів, що підтримуються різними операційними системами. Для настільних операційних систем, таких як Windows 10, Linux та macOS, підтримуються такі популярні браузери, як Google Chrome, Firefox та Brave. На мобільних платформах сумісні такі браузери, як Google Chrome, Dolphin, Opera Mobile, Mozilla Firefox і Safari.

Додаток призначений для роботи на останніх двох версіях всіх підтримуваних браузерів, як визначено офіційними розробниками. Хоча додаток може працювати і на старіших версіях браузерів, його успішна робота в них не гарантується.

Тому користувачі можуть отримати доступ до веб-трекеру через свій улюблений браузер, що забезпечує широку сумісність і доступність на різних платформах, операційних системах і пристроях.

2.7.4. Опис інтерфейсу користувача

MERN Stack месенджер має зручний та інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам зручно відстежувати та відправляти дані іншип користувачам. Після запуску застосунку користувачу відкриється сторінка Login/Sign Up, де користувачі можуть або увійти в систему, або зареєструвати новий обліковий запис. Крім того, для більшої зручності користувачі можуть увійти, використовуючи свої облікові дані Google.

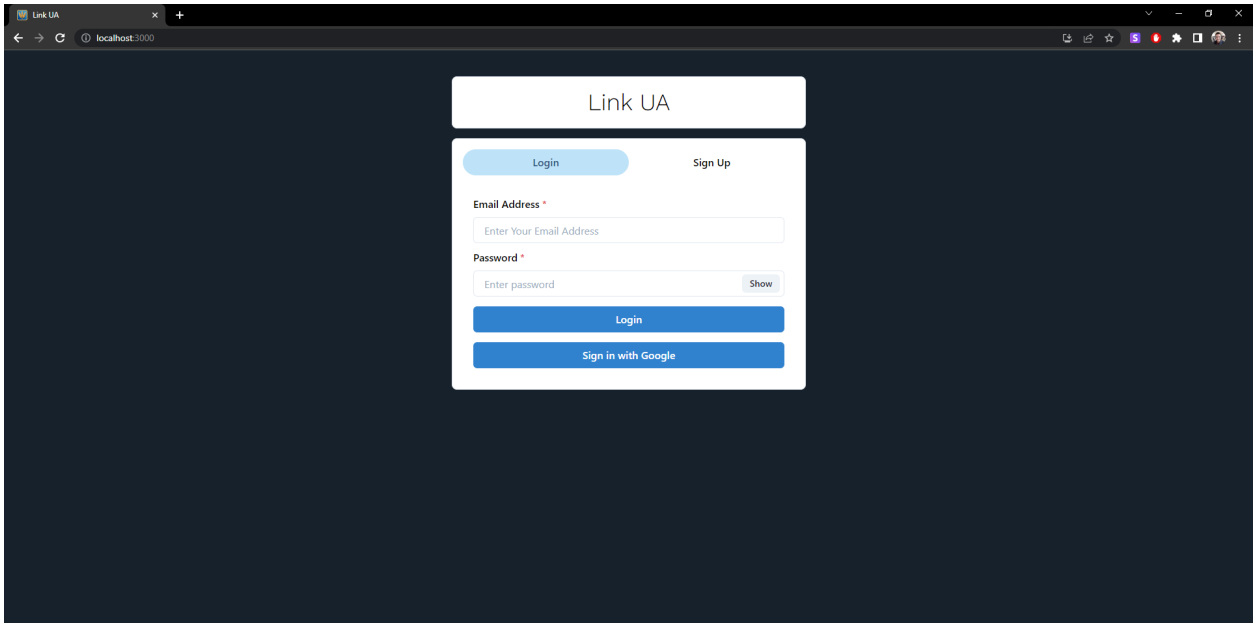

Рис. 2.3. Login сторінка

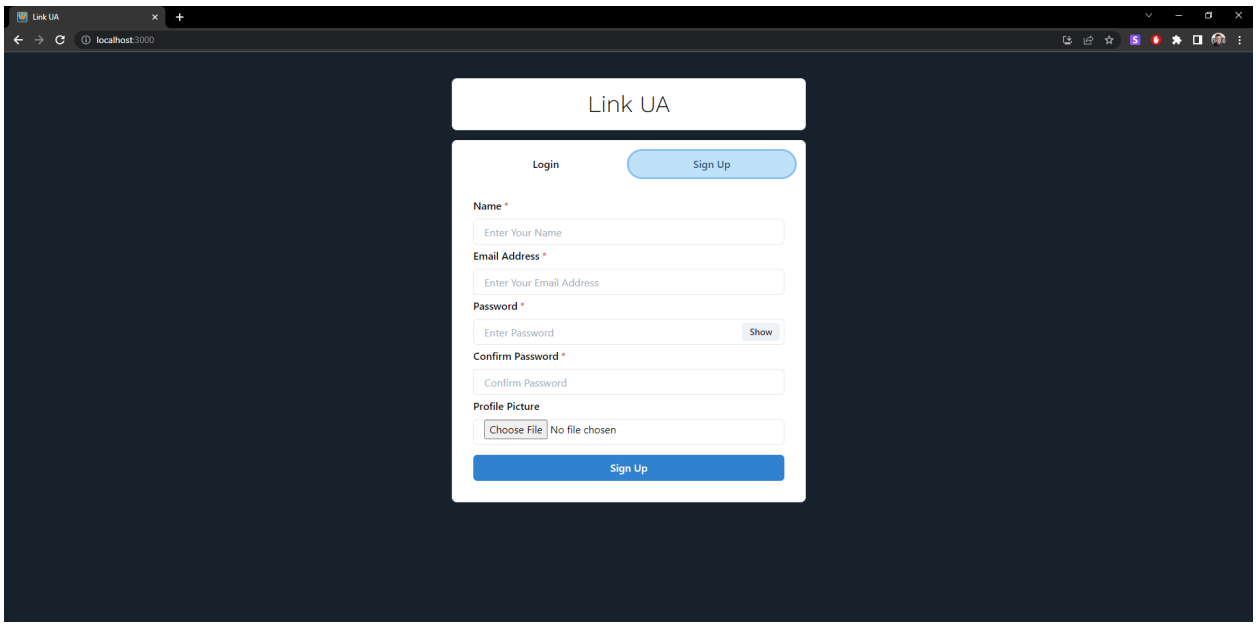

Рис. 2.4. Sign Up сторінка

На цих сторінках поля мають перевірку на корректність введених даних.

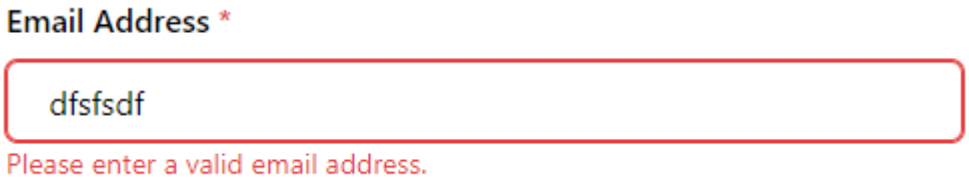

Рис. 2.5. Попередження про недійну електронну пошту

Рис. 2.6. Попередження про відсутність даних в обов`язковому полі

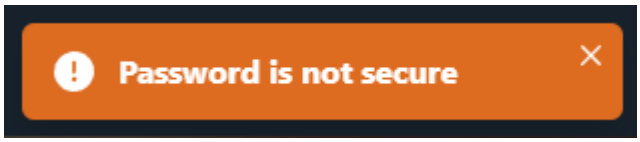

Рис. 2.7. Попередження про ненадійний пароль

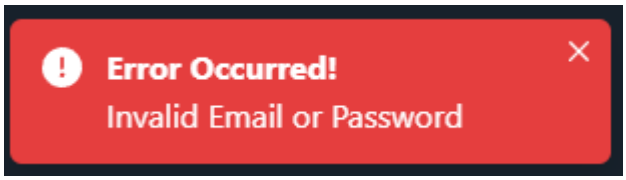

Рис. 2.8. Помилка що такі дані відсудтні на сервері

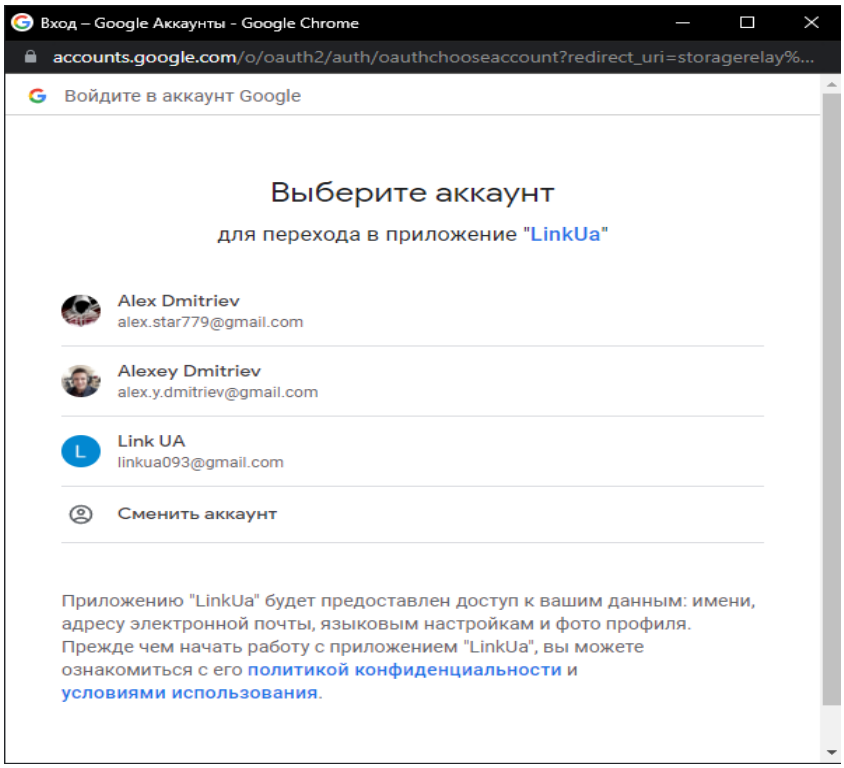

Рис. 2.9. Логін за допомогою Google

Після успішного входу перед користувачем з`являється нове вікно з усіма чатами користувача в яких скорочено відображено останнє повідомлення і час його відправлення.

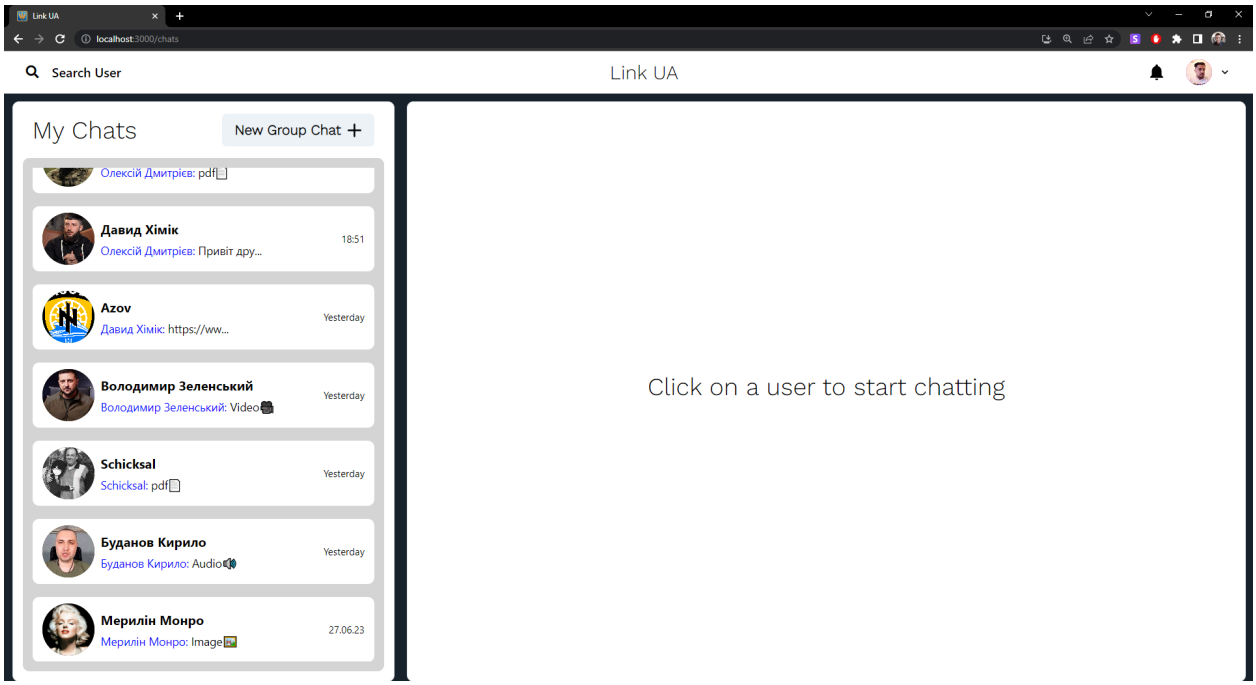

Рис. 2.10. Вікно вибору чату

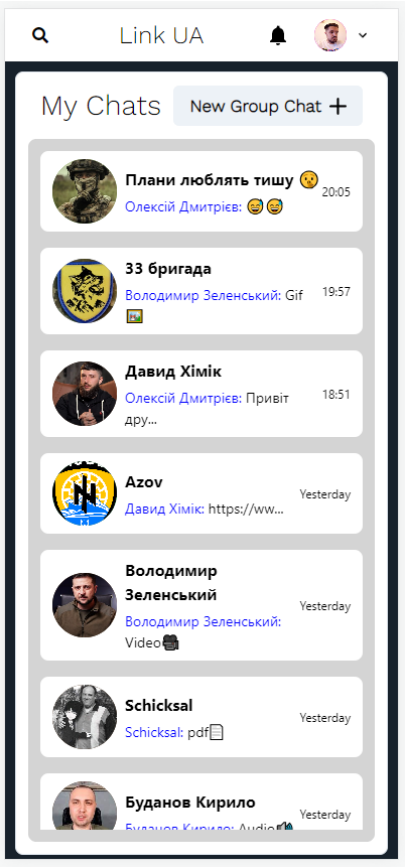

Рис. 2.11. Вікно вибору чату мобільна версія

Щоб розпочати діалог необхідно знайти свого співрозмовника натиснувши на кнопку Search User.

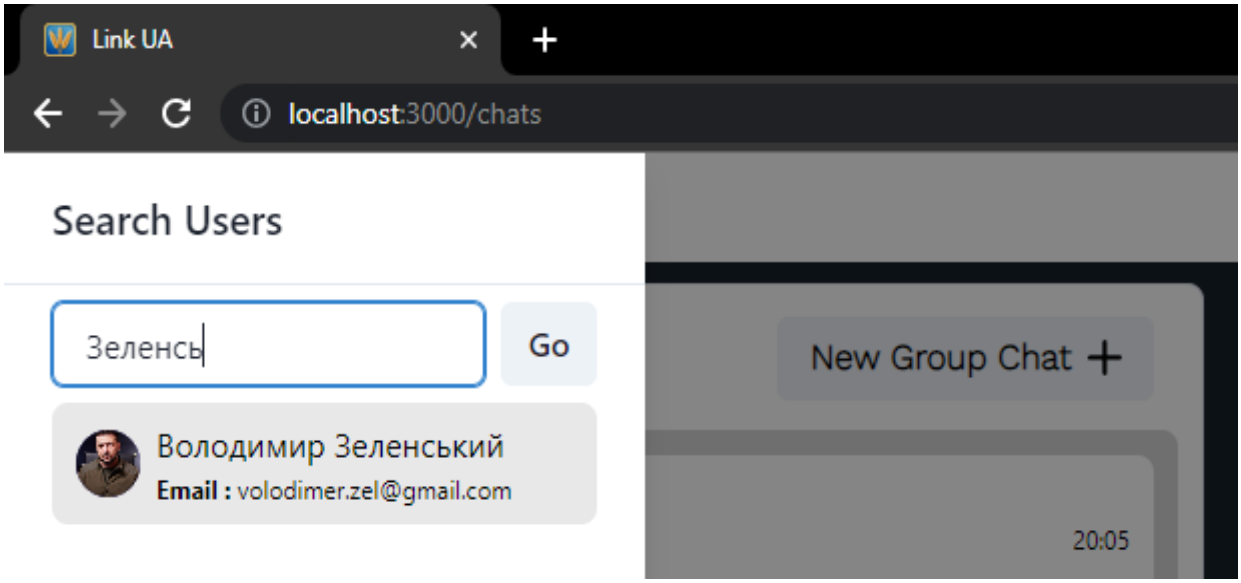

Рис. 2.10. Вікно пошуку співрозмовника

Після того, як користувач вибрав співрозмовника, праворуч з'явиться вікно з листуванням. У цьому вікні можна відправити текстове повідомлення, поділитися будь-яким файлом та оглянути профіль співрозмовника.

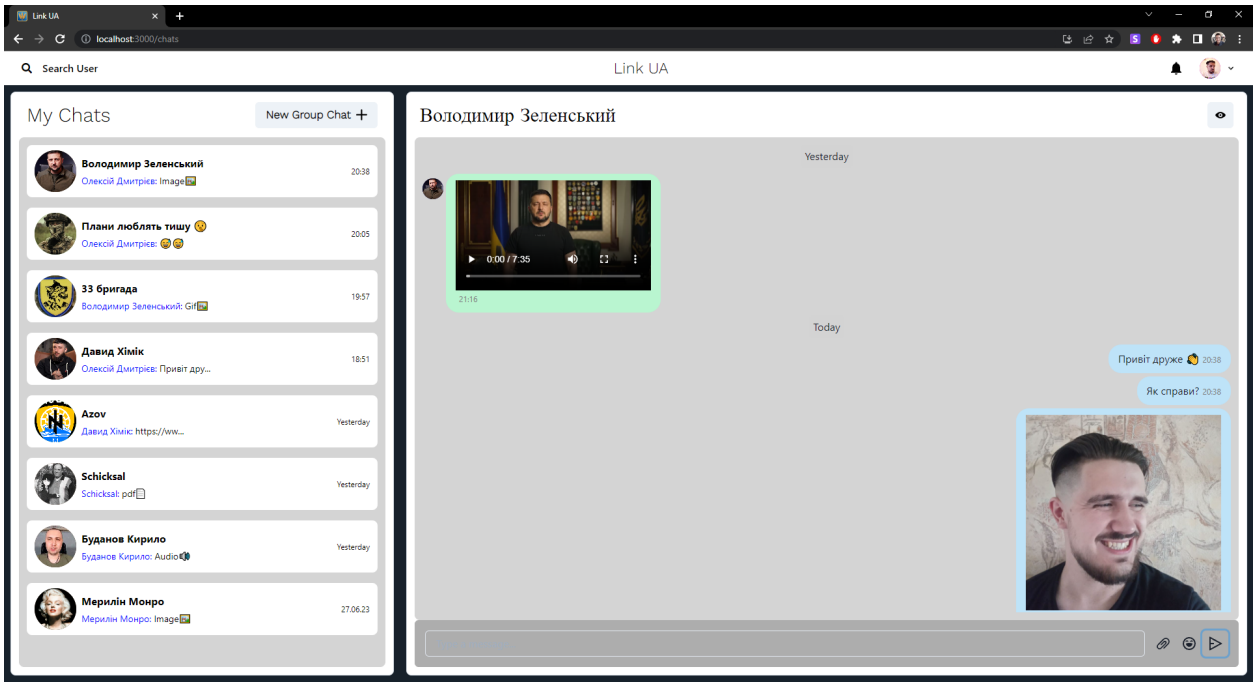

Рис. 2.11. Вікно з листуванням

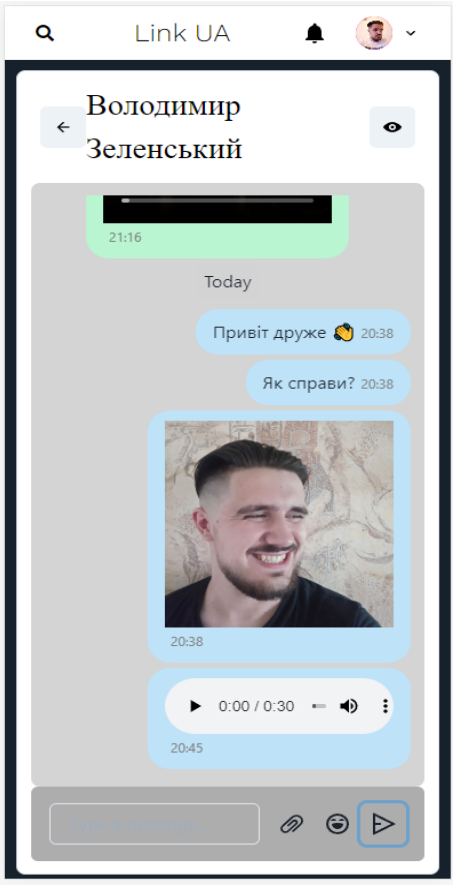

Рис. 2.12. Вікно з листуванням мобільна версія

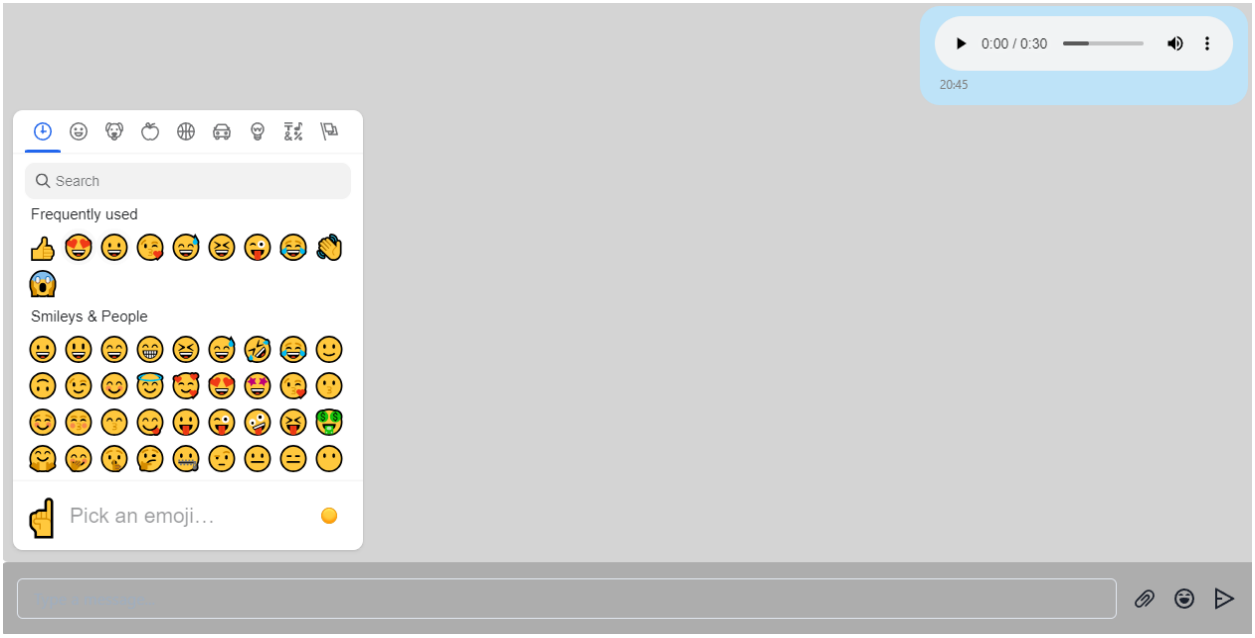

Рис. 2.13. Вибір емодзі

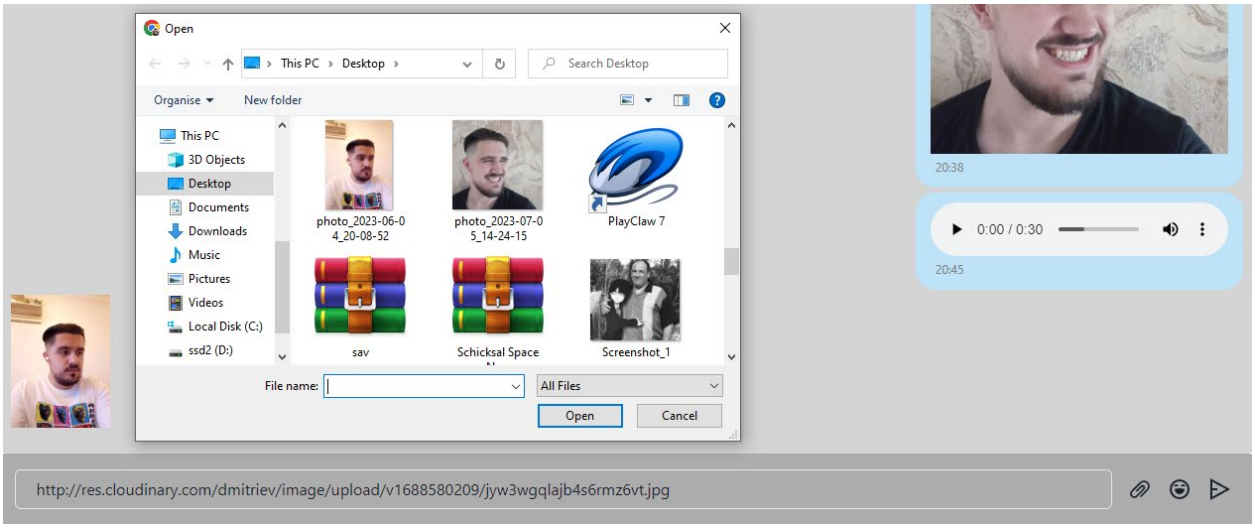

Рис. 2.14. Вибір та завантаження файлу на сервер

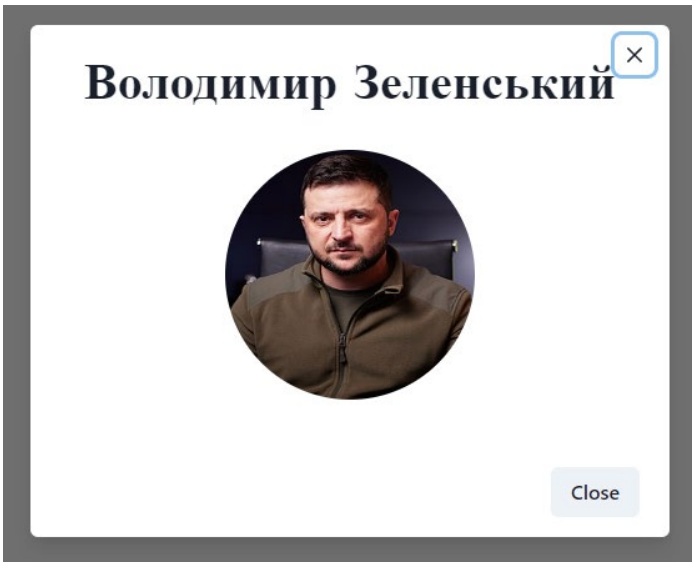

Рис. 2.15. Інформація про співрозмовника

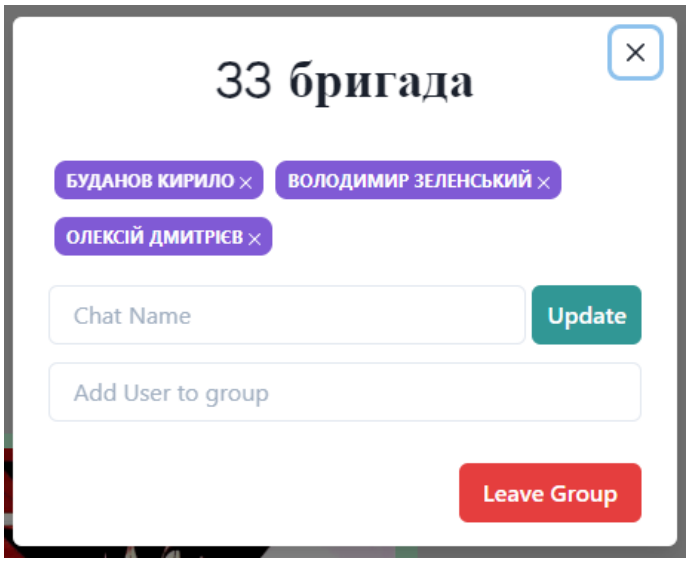

Рис. 2.16. Інформація про груповий чат

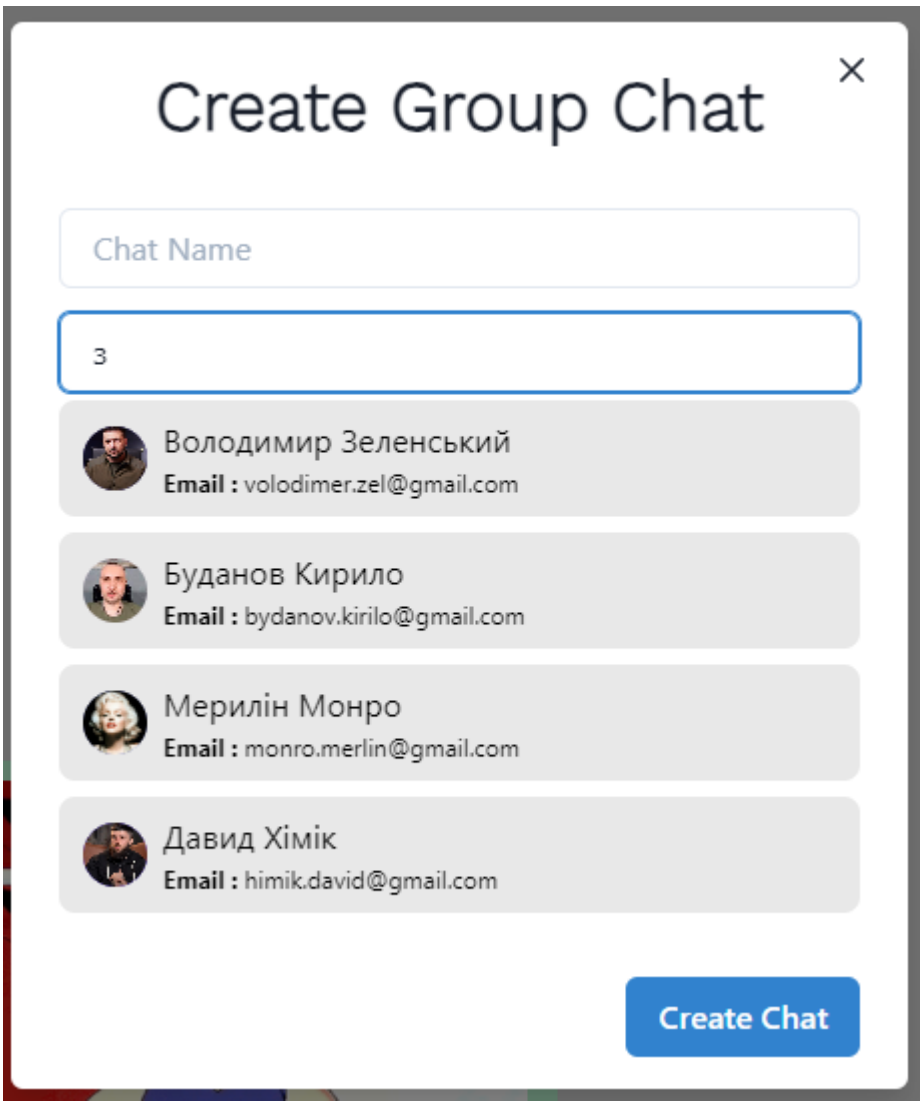

Рис. 2.17. Створення групового чату

В цілому, інтерфейс MERN Stack месенджера розроблений таким чином, щоб забезпечити всіх користувачів доступом до необхідних інструментів для ефективного та комфортного листування.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1703;
2. коефіцієнт складності програми – 1,4;
3. коефіцієнт корекції програми в ході її розробки – 0,3;
4. годинна заробітна плата програміста – 185 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,1;
7. вартість машино-години ЕОМ – 19 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

ti – витрати праці на дослідження алгоритму рішення задачі;

ta – витрати праці на розробку блок-схеми алгоритму;

tп – витрати праці на програмування по готовій блок-схемі;

tотл – витрати праці на налагодження програми на ЕОМ;

tд – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \tag{3.2}$$

де q – передбачуване число операторів;

С – коефіцієнт складності програми;

р – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1703 \cdot 1{,}4 \cdot (1 + 0{,}3) = 3099$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \tag{3.3}$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = (3099 \cdot 1{,}3) / (85 \cdot 1{,}1) = 43 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot K}, \tag{3.4}$$

t_a = 3099 / (20 · 1,1) = 141 людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_п = \frac{Q}{(20...25)\cdot K}, \quad (3.5)$$

$t_п$ = 3099 / (23 · 1,1) = 122,49 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

− за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)\cdot K}, \quad (3.6)$$

$t_{отл}$ = 3099 / (5 · 1,1) = 563,45 людино-годин.

− за умови комплексного налагодження завдання:

$$t^{к}_{отл} = 1,2 \cdot t_{отл}, \quad (3.7)$$

$t_{отл}^{k}$ = 1,4 · 563,45 = 788,83 людино-годин.

Витрати праці на підготовку документації визначається за формулою:

$$t_д = t_{др} + t_{до}\text{ , людино-годин,}$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15...20)*k}\text{ , людино-годин.}$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 * t_д \text{ , людино-годин.}$$

Підставляючи відповідні значення, отримуємо:

$$t_{др} = \frac{3099}{20*1{,}1} = 140{,}86, \text{ людино-годин.}$$

$$t_{до} = 0{,}75 * 140{,}86 = 105{,}64, \text{ людино-годин.}$$

$$t_д = 140{,}86 + 105{,}64 = 246{,}5, \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 43 + 141 + 122{,}49 + 563{,}45 + 246{,}5 = 1166{,}44 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ включають витрати на заробітну плату виконавця програми З/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ} \text{ грн.,} \qquad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР} \text{ грн.,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{зп} = 1166{,}44 * 185 = 215791 \text{ грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М} \text{ грн.,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$С_{Мч}$ – вартість машино-години ЕОМ, грн/год.

$$З_{мв} = 563{,}45 * 19 = 10705 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{по} = 215791 + 10705 = 226496 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.,} \qquad (3.14)$$

де $В_k$- число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні $F_р$=176 годин).

$$T = \frac{1166{,}44}{1 \cdot 176} \approx 4{,}9 \text{міс.}$$

Висновок: вартість розробки програми месенджера за допомогою MERN Stack становить 226496 грн. Приблизний час, який буде витрачено на розробку одним виконавцем – 4,9 місяців при 40-годинному робочому тижні. Такий термін включає в собі час необхідний для дослідження галузі, технологій, розробки клієнтської частини, налагодження серверної, розробки алгоритму розв’язання задачі, створення документації. Загальна кількість людино-годин, яку буде витрачено на розробку – 1166,44.

ВИСНОВКИ

Метою цієї кваліфікаційної роботи була розробка зручної та ефективної програми-месенджера з використанням стеку MERN, що дозволяє користувачам ефективно спілкуватися в цифровому середовищі. В процесі розробки було успішно вирішено кілька ключових завдань.

Перш за все, додаток було розроблено та реалізовано з використанням сучасних технологій, пов'язаних зі стеком MERN. Ці технології забезпечили надійну основу для створення адаптивного та інтерактивного користувацького інтерфейсу в програмі-месенджері.

Інтеграція функції обміну повідомленнями в режимі реального часу забезпечила безперебійну комунікацію між користувачами. Завдяки використанню таких технологій, як Socket.io, користувачі могли обмінюватися повідомленнями в режимі реального часу, покращуючи загальний користувацький досвід.

Використання MongoDB сприяло ефективному зберіганню та пошуку повідомлень користувачів та інших важливих даних. Ця база даних NoSQL виявилася масштабованою та надійною, забезпечуючи узгодженість і безпеку даних у програмі-месенджері.

Включення автентифікації та авторизації користувачів було здійснено за допомогою впровадження Passport.js, широко прийнятого проміжного програмного забезпечення для автентифікації. Це дозволило користувачам безпечно реєструватися, входити в систему та отримувати доступ до своїх персональних облікових записів обміну повідомленнями.

Протягом усього процесу розробки використовувалися різні патерни проектування та бібліотеки, пов'язані зі стеком MERN. Ці патерни, такі як компонентна розробка та використання React Router DOM і Material-UI, сприяли підвищенню зручності супроводу, модульності та повторного використання коду.

Протягом усього процесу розробки використовувалися різні патерни проектування та бібліотеки, пов'язані зі стеком MERN. Ці патерни, такі як компонентна розробка та використання React Router DOM і Material-UI, сприяли підвищенню зручності супроводу, модульності та повторного використання коду.

Таким чином, розробка веб-трекера криптовалютного ринку відкриває нові можливості для інвесторів, трейдерів та ентузіастів залишатися в курсі подій та приймати обґрунтовані рішення в динамічному світі криптовалют. Він служить інструментом для навігації в криптовалютному ринку і надає користувачам необхідну інформацію для прийняття розумних інвестиційних рішень.

Зрештою, розробка програми-месенджера з використанням стеку MERN створює нові можливості для безперешкодної та ефективної комунікації в цифровому контексті. Він слугує цінним інструментом для сприяння ефективному та безпечному спілкуванню в сучасному взаємопов'язаному світі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gordon, Z. React Explained: Your Step-by-Step Guide to React. Independently published, 2020. (Author), Hill, M. A. (Editor), Adair, R. (Editor). — 366 с.
2. Banks, A., Porcello, E. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020. — 307 с.
3. Lim, G. Beginning React (incl. Redux and React Hooks). Greg Lim, 2020.— 164 с.
4. React. URL: https://react.dev
5. Wieruch, R. The Road to React: Your journey to master plain yet pragmatic React.js. Independently published, 2018. — 292 с.
6. Griffiths, D. and Griffiths, D. React Cookbook: Recipes for Mastering the React Framework. O'Reilly Media, 2021. — 510 с.
7. Kuttig, A. B. Professional React Native: Expert techniques and solutions for building high-quality, cross-platform, production-ready apps. Packt Publishing, 2022. — 268 с.
8. Blokdyk, G. Firebase: The Ultimate Step-By-Step Guide. 5STARCooks, 2022. — 302 с.
9. Wieruch, R. The Road to React with Firebase: Your journey to master advanced React for business web applications. Independently published, 2019. — 199 с.
10. Scott Domes. Progressive Web Apps with React: Create lightning fast web apps with native power using React and Firebase. — Packt Publishing, 2017. — 302 с.
11. MongoDB. URL: https://www.mongodb.com/docs/
12. Yahiaoui, H. Firebase Cookbook: Over 70 recipes to help you create real-time web and mobile applications with Firebase. Packt Publishing, 2017. — 288 с.
13. JJ Geewax. API Design Patterns. — Manning, 2021. — 480 с.

14. Subramanian, H. and Raj, P. Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs. Packt Publishing, 2019. — 378 c.
15. Doglio, F. REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development. Apress, 2018. — 402 c.
16. Biehl, M. RESTful API Design: Best Practices in API Design with REST (API-University Series Book 3). API-University Press, 2017. — 207c.
17. Ludin, S. and Garza, J. Learning HTTP/2: A Practical Guide for Beginners. O'Reilly Media, 2017. — 156 c.
18. Flanagan, D. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. O'Reilly Media, 2020. — 704 c.
19. Ackermann, P. JavaScript: The Comprehensive Guide. Rheinwerk Computing, 2023. — 982 c.
20. Boduch, A. JavaScript: React Material-UI Cookbook: Build captivating user experiences using React and Material-UI. Packt Publishing, 2019. — 534 c.
21. Johnson, B. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers. Wiley, 2019. — 192 c.
22. Wexler, J. Get Programming with Node.js. Manning, 2019. — 480c.
23. Git. URL: https://git-scm.com
24. Santacroce, F. Git Essentials: Create, merge, and distribute code with Git, the most powerful and flexible versioning system available, 2nd Edition. Packt Publishing, 2017. — 238 c.

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг SingleChat.js:

```
import { FormControl } from "@chakra-ui/form-control";
import { Input } from "@chakra-ui/input";
import { Box, Text } from "@chakra-ui/layout";
import "./styles.css";
import { IconButton, Spinner, useToast } from "@chakra-ui/react";
import { getSender, getSenderFull } from "../config/ChatLogics";
import axios from "axios";
import { ArrowBackIcon, AttachmentIcon } from '@chakra-ui/icons';
import ProfileModal from "./miscellaneous/ProfileModal";
import ScrollableChat from "./ScrollableChat";
import Lottie from "react-lottie";
import animationData from "../animations/typing.json";
import io from "socket.io-client";
import UpdateGroupChatModal from "./miscellaneous/UpdateGroupChatModal";
import { ChatState } from "../Context/ChatProvider";
import data from '@emoji-mart/data'
import Picker from '@emoji-mart/react'
import { RiEmotionLaughLine } from 'react-icons/ri';
import { RiSendPlane2Line } from 'react-icons/ri';
import ScrollToBottom from 'react-scroll-to-bottom';
import React, { useEffect, useState, useRef } from 'react';
import { encryptMessage, decryptMessage } from "../config/cryptography"; // Import the encryption/decryption
functions
// Endpoint for the socket.io server
const ENDPOINT = "https://link-ua-messenger-f8abb2e22ca4.herokuapp.com/"; // "https://talk-a-
tive.herokuapp.com"; -> After deployment
var socket, selectedChatCompare;

const SingleChat = ({ fetchAgain, setFetchAgain, boxColor }) => {
  const [messages, setMessages] = useState([]); // State for storing messages
  const [loading, setLoading] = useState(false); // State for loading status
  const [newMessage, setNewMessage] = useState(""); // State for new message input
  const [socketConnected, setSocketConnected] = useState(false); // State for socket connection status
  const [typing, setTyping] = useState(false); // State for typing status
  const [istyping, setIsTyping] = useState(false); // State for istyping status
  const toast = useToast(); // Toast notification
  const scrollRef = useRef(null); // Reference to the scroll element
  const [urls, setUrls] = useState([]); // State for storing URLs
  const [pic, setPic] = useState(''); // State for storing picture URL
  const [audioUrl, setAudioUrl] = useState(''); // State for storing audio URL
  const [showEmojiPicker, setShowEmojiPicker] = useState(false); // State for showing/hiding emoji picker
  const [selectedEmoji, setSelectedEmoji] = useState(''); // State for selected emoji
  const [inputValue, setInputValue] = useState(''); // State for input value
  const fileInputRef = useRef(null); // Reference to the file input

  // Function to toggle emoji picker visibility
  const toggleEmojiPicker = () => {
    setShowEmojiPicker(!showEmojiPicker);
    setTimeout(scrollToBottom, 100);
  };

  // Function to handle file selection
  const handleFileSelect = (e) => {
    setTimeout(scrollToBottom, 100);
    const file = e.target.files[0];
```

```
  postDetails(file);
};

// Function to handle emoji selection
const handleEmojiSelect = (emoji) => {
  console.log(emoji);
  const nativeContent = emoji.native;
  setNewMessage((prevMessage) => prevMessage + nativeContent);
  // Additional logic with the selected emoji
};

const handleChange = (event) => {
  setNewMessage(event.target.value);
};

const handleInputChange = (e) => {
  setInputValue(e.target.value);
};

const scrollToBottom = () => {
  if (scrollRef.current) {
    scrollRef.current.scrollIntoView({ behavior: "auto",
    block: "end"});
  }
};

  const postDetails = (file) => {
    if (!file) {
      toast({
        title: "Please Select a File!",
        status: "warning",
        duration: 5000,
        isClosable: true,
        position: "bottom",
      });
      return;
    }

    const mediaVideo = ['.mp4', '.mkv', '.webm', '.ogg', '.m4v', '.f4v', '.mov', '.mpeg', '.avi', 'wmv'];
const mediaAudio = ['.mp3', '.wav', '.ogg', '.aac', '.m4a', '.wma', '.flac', '.aiff', '.opus'];
const mediaDoc = ['.doc', '.docx', '.xls', '.xlsx', '.ppt', '.pptx', '.txt', '.txt', '.pdf'];
const mediaZip = ['.zip', '.rar', '.7zip'];

const isImage = file.type.startsWith("image/");
const isVideo = mediaVideo.some(extension => file.name.endsWith(extension));
const isGif = file.type === "image/gif";
const isAudio = mediaAudio.some(extension => file.name.endsWith(extension));
const isDocument = mediaDoc.some(extension => file.name.endsWith(extension));
const isZip = mediaZip.some(extension => file.name.endsWith(extension));

if (!isImage && !isVideo && !isGif && !isAudio && !isDocument && !isZip) {
  toast({
    title: "Please Select an Image, Video, GIF, Audio, Archive or Document!",
    status: "warning",
    duration: 5000,
    isClosable: true,
    position: "bottom",
  });
  return;
}

    const data = new FormData();
```

```
  data.append("file", file);
  data.append("upload_preset", "chat-app");
  data.append("cloud_name", "dmitriev");

  const endpoint = isImage
    ? "https://api.cloudinary.com/v1_1/dmitriev/image/upload"
    : isVideo
    ? "https://api.cloudinary.com/v1_1/dmitriev/video/upload"
    : isAudio
    ? "https://api.cloudinary.com/v1_1/dmitriev/upload" // Corrected endpoint for audio files
    : isDocument
    ? "https://api.cloudinary.com/v1_1/dmitriev/upload" // Endpoint for document files
    : "https://api.cloudinary.com/v1_1/dmitriev/image/upload" // Use the same endpoint for .gif files
    ? "https://api.cloudinary.com/v1_1/dmitriev/upload" // Corrected endpoint for audio files
    : isZip;
  axios
    .post(endpoint, data)
    .then((response) => {
      const url = response.data.url.toString();
      if (isImage) {
        // Automatically write the image URL into the input field
        setNewMessage((prevMessage) => prevMessage + " " + url + " ");
      } else if (isVideo || isGif) {
        // Automatically write the video/GIF URL into the input field
       setNewMessage((prevMessage) => prevMessage + " " + url + " ");
      } else if (isAudio) {
        // Handle audio file URL
        setNewMessage((prevMessage) => prevMessage + " " + url + " ");
        console.log("Audio URL:", url);
      } else if (isDocument) {
        // Handle document file URL
        setNewMessage((prevMessage) => prevMessage + " " + url + " ");
        console.log("Document URL:", url);
      }
      else if (isZip) {
        // Handle document file URL
        setNewMessage((prevMessage) => prevMessage + " " + url + " ");
        console.log("Zip URL:", url);
      }
      console.log(url);
    })
    .catch((error) => {
      console.log(error);
    });

};

const postVideoDetails = (video) => {
  if (video === undefined) {
    toast({
      title: "Please Select a Video!",
      status: "warning",
      duration: 5000,
      isClosable: true,
      position: "bottom",
    });
    return;
  }
  console.log(video);
  if (video.type === "video/mp4") {
    const data = new FormData();
    data.append("file", video);
```

```
    data.append("upload_preset", "chat-app");
    data.append("cloud_name", "dmitriev");
    fetch("https://api.cloudinary.com/v1_1/dmitriev/video/upload", {
      method: "post",
      body: data,
    })
      .then((res) => res.json())
      .then((data) => {
        setPic(data.url.toString());
        console.log(data.url.toString());
        // Automatically write the video URL into the input field
        setNewMessage(data.url.toString());
      })
      .catch((err) => {
        console.log(err);
      });
  } else {
    toast({
      title: "Please Select a Video!",
      status: "warning",
      duration: 5000,
      isClosable: true,
      position: "bottom",
    });
    return;
  }
};

const isImageURL = (url) => {
  const mediaExtensions = ['.png', '.jpg', '.jpeg', '.gif'];
  const extension = url.substring(url.lastIndexOf('.')).toLowerCase();
  return mediaExtensions.includes(extension);
};

const videoURL = (url) => {
  const mediaExtensions = ['.mp4','.mkv','.webm','.ogg','.m4v','.f4v','.mov','.mpeg','.avi','wmv'];
  const extension = url.substring(url.lastIndexOf('.')).toLowerCase();
  return mediaExtensions.includes(extension);
};

const musicURL = (url) => {
  const mediaExtensions = ['.mp3', '.wav', '.ogg', '.aac', '.m4a', '.wma', '.flac', '.aiff', '.opus'];
  const extension = url.substring(url.lastIndexOf('.')).toLowerCase();
  return mediaExtensions.includes(extension);
};

const docURL = (url) => {
  const mediaExtensions = ['.doc','.docx','.xls','.xlsx','.ppt','.pptx','.txt'];
  const extension = url.substring(url.lastIndexOf('.')).toLowerCase();
  return mediaExtensions.includes(extension);
};

const pdfURL = (url) => {
  const mediaExtensions = ['.pdf'];
  const extension = url.substring(url.lastIndexOf('.')).toLowerCase();
  return mediaExtensions.includes(extension);
};

const zipURL = (url) => {
  const mediaExtensions = ['.zip', '.rar', '.7zip'];
  const extension = url.substring(url.lastIndexOf('.')).toLowerCase();
  return mediaExtensions.includes(extension);
```

```
  };

  const [videos, setVideos] = useState([]);
  const inputFileRef = useRef(null);

const defaultOptions = {
  loop: true,
  autoplay: true,
  animationData: animationData,
  rendererSettings: {
    preserveAspectRatio: "xMidYMid slice",
  },
};
const { selectedChat, setSelectedChat, user, notification, setNotification } =
  ChatState();

const fetchMessages = async () => {
  if (!selectedChat) return;

  try {
    const config = {
      headers: {
        Authorization: `Bearer ${user.token}`,
      },
    };

    setLoading(true);// Set loading status to true

    const { data } = await axios.get(
      `/api/message/${selectedChat._id}`,
      config
    );
    setMessages(data);
    setLoading(false);

    socket.emit("join chat", selectedChat._id);
  } catch (error) {
    toast({
      title: "Error Occured!",
      description: "Failed to Load the Messages",
      status: "error",
      duration: 5000,
      isClosable: true,
      position: "bottom",
    });
  }
};

  const playMusic = () => {
    const audioUrl = "http://res.cloudinary.com/dmitriev/video/upload/v1687114210/y5rvp0y62s6kggvs3yo2.wav";
    const audio = new Audio(audioUrl);
    audio.play()
    .then(() => {
      // Playback started successfully
    })
    .catch((error) => {
      // Failed to start playback
      console.log(error);
    });
  };

  const sendMessage = async () => {
```

```
    if (newMessage) {
      socket.emit('stop typing', selectedChat._id);
      try {
        const config = {
          headers: {
            'Content-type': 'application/json',
            Authorization: `Bearer ${user.token}`,
          },
        };

        // Encrypt the newMessage using a secret key
        const key = 'yFSX0L7JmGtQPIWf';
        // Encrypt the message using the encryption function
        const encryptedMessage = encryptMessage(newMessage, key);
        // Clear the new message input
        setNewMessage('');

        const { data } = await axios.post(
          '/api/message',
          {
            content: encryptedMessage,
            chatId: selectedChat._id,
          },
          config
        );

        // Emit a "chat message" event with the encrypted message
        socket.emit('new message', data);
        setMessages([...messages, data]);
      } catch (error) {
        toast({
          title: 'Error Occurred!',
          description: 'Failed to send the Message',
          status: 'error',
          duration: 5000,
          isClosable: true,
          position: 'bottom',
        });
      }
    }
    setTimeout(scrollToBottom, 100);
  };

useEffect(() => {
  setTimeout(scrollToBottom, 100);
}, [messages]);

useEffect(() => {
  socket = io(ENDPOINT);
  socket.emit("setup", user);
  socket.on("connected", () => setSocketConnected(true));
  socket.on("typing", () => setIsTyping(true));
  socket.on("stop typing", () => setIsTyping(false));

  // eslint-disable-next-line
}, []);

useEffect(() => {
  fetchMessages();

  selectedChatCompare = selectedChat;
  // eslint-disable-next-line
```

```
}, [selectedChat]);

useEffect(() => {
  socket.on("message recieved", (newMessage) => {
    // Decrypt the received message
    const decryptedMessage = decryptMessage(newMessage.content);
    setMessages((prevMessages) => [...prevMessages, { ...newMessage, content: decryptedMessage }]);
  });
}, []);

const typingHandler = (e) => {
  setNewMessage(e.target.value);

  if (!socketConnected) return;

  if (!typing) {
    setTyping(true);
    socket.emit("typing", selectedChat._id);
  }
  let lastTypingTime = new Date().getTime();
  var timerLength = 3000;
  setTimeout(() => {
    var timeNow = new Date().getTime();
    var timeDiff = timeNow - lastTypingTime;
    if (timeDiff >= timerLength && typing) {
      socket.emit("stop typing", selectedChat._id);
      setTyping(false);
    }
  }, timerLength);
};

useEffect(() => {
  const urlRegex = /(https?:\/\/[^\s]+)/g;
  const matches = newMessage.match(urlRegex);
  if (matches) {
    setUrls(matches);
  } else {
    setUrls([]);
  }
}, [newMessage]);

return (
  <>
    {selectedChat ? (
      <>
        <Text
          fontSize={{ base: "28px", md: "30px" }}
          pb={3}
          px={2}
          w="100%"
          fontFamily="Work sans"
          d="flex"
          justifyContent={{ base: "space-between" }}
          alignItems="center"
          color={boxColor === "black" ? "whatsapp.100" : "black"}
        >
          <IconButton
            d={{ base: "flex", md: "none" }}
            icon={<ArrowBackIcon />}
            onClick={() => setSelectedChat("")}
          />
          {messages &&
```

```
    (!selectedChat.isGroupChat ? (
      <>
        {getSender(user, selectedChat.users)}
        <ProfileModal
          user={getSenderFull(user, selectedChat.users)}
        />
      </>
    ) : (
      <>
        {selectedChat.chatName.toUpperCase()}
        <UpdateGroupChatModal
          fetchMessages={fetchMessages}
          fetchAgain={fetchAgain}
          setFetchAgain={setFetchAgain}
        />
      </>
    ))}
  </Text>
  <Box
    display="flex"
    flexDir="column"
    justifyContent="flex-end"
    p={3}
    bg={boxColor === "black" ? "#1C1C1C" : "#d4d4d4"}
    w="100%"
    h="100%"
    borderRadius="lg"
    overflowY="hidden"
  >
    {loading ? (
      <Spinner
        size="xl"
        w={20}
        h={20}
        alignSelf="center"
        margin="auto"
      />
    ) : (
      <div className="messages">
  <ScrollToBottom className="scroll-container">
    <ScrollableChat messages={messages}/>
    <div ref={scrollRef} />
  </ScrollToBottom>
</div>
    )}
    <Box>
      {urls.map((url, index) => {
        if (isImageURL(url)) {
          return (
            <Box
              key={index}
              display="inline-block"
              maxWidth="100px"
              maxHeight="200px"
              mb={2}
            >
              <img
                src={url}
                alt="Image"
                style={{ maxWidth: '100%', maxHeight: '100%' }}
              />
            </Box>
```

```
  );
} else if (musicURL(url)) {
  return (
    <audio
     key={index}
     src={url}
     alt="audio"
     controls
    />
  );
}
else if (docURL(url)) {
  return (
    <Box
      key={index}
      display="inline-block"
      maxWidth="200px"
      maxHeight="300px"
      mb={2}
    >
      <img
        src={'https://res.cloudinary.com/dmitriev/image/upload/v1686659575/uzish466ibfhjhgavmis.png'}
        alt="Image"
        style={{ maxWidth: '100%', maxHeight: '100%' }}
      />
    </Box>
  );
}
else if (pdfURL(url)) {
  return (
    <Box
      key={index}
      display="inline-block"
      maxWidth="200px"
      maxHeight="300px"
      mb={2}
    >
      <img
        src={'http://res.cloudinary.com/dmitriev/image/upload/v1688407846/vzunssmhftp4zegrbb6p.png'}
        alt="Image"
        style={{ maxWidth: '100%', maxHeight: '100%' }}
      />
    </Box>
  );
}
else if (zipURL(url)) {
  return (
    <Box
      key={index}
      display="inline-block"
      maxWidth="200px"
      maxHeight="300px"
      mb={2}
    >
      <img
        src={'http://res.cloudinary.com/dmitriev/image/upload/v1687175207/jtucenuqv2im9sqlooyh.png'}
        alt="Image"
        style={{ maxWidth: '100%', maxHeight: '100%' }}
      />
    </Box>
  );
}
```

```
            else if (videoURL(url)) {
              return (
                <video
                key={index}
                src={url}
                alt="Video"
                controls
                style={{ maxWidth: '200px', maxHeight: '300px' }}
                />
              );
            }
          })}
        </Box>
        {showEmojiPicker && (
    <Picker data={data} onEmojiSelect={handleEmojiSelect}/>
)}
        </Box>
       <Box display="flex" bg="#adadad" p={4} borderRadius="md" width="100%">
       <FormControl>
  <Box display="flex">
  <Input
  value={newMessage}
  onChange={handleChange}
  placeholder="Type a message..."
  focusBorderColor="blue.400"
  flex="1"
  borderRadius="md"
  mr={2}
  _focus={{
    boxShadow: 'none',
  }}
  _active={{
    boxShadow: 'none',
    pointerEvents: 'none',
  }}
  onKeyDown={(event) => event.key === 'Enter' && sendMessage('')}
  autoFocus={true} // Set initial focus
/>
    <IconButton
      aria-label="Attach file"
      icon={<AttachmentIcon fontSize="xl" />}
      onClick={() => fileInputRef.current.click()}
      variant="unstyled"
      sx={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}
    />
    <IconButton
      aria-label="Emotion"
      icon={<RiEmotionLaughLine size={24}/>}
      onClick={toggleEmojiPicker}
      variant="unstyled"
      sx={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}
    />
    <IconButton
      aria-label="Send message"
```

```
      icon={<RiSendPlane2Line size={24}/>}
      onClick={sendMessage}
      variant="unstyled"
      sx={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}
    />
  </Box>
  <input
    type="file"
    ref={fileInputRef}
    style={{ display: 'none' }}
    onChange={handleFileSelect}
    data-testid="file-input"
  />
</FormControl>
</Box>
        </>
      ) : (
        <Box d="flex" alignItems="center" justifyContent="center" h="100%">
          <Text fontSize="3xl" pb={3} fontFamily="Work sans" color={boxColor === "black" ? "whatsapp.100" :
"black"}>
            Click on a user to start chatting
          </Text>
        </Box>
      )}
    </>
  );
};

export default SingleChat;
```

Лістинг Server.js:

```
const express = require("express");

const connectDB = require("./config/db");

const dotenv = require("dotenv");

const userRoutes = require("./routes/userRoutes");

const chatRoutes = require("./routes/chatRoutes");

const messageRoutes = require("./routes/messageRoutes");

const { notFound, errorHandler } = require("./middleware/errorMiddleware");

const path = require("path");

dotenv.config();

connectDB();

const app = express();

app.use(express.json()); // to accept json data

// app.get("/", (req, res) => {

//   res.send("API Running!");

// });
```

```
app.use("/api/user", userRoutes);
app.use("/api/chat", chatRoutes);
app.use("/api/message", messageRoutes);
// --------------------------deployment------------------------------
const __dirname1 = path.resolve();
if (process.env.NODE_ENV === "production") {
  app.use(express.static(path.join(__dirname1, "/frontend/build")));
  app.get("*", (req, res) =>
    res.sendFile(path.resolve(__dirname1, "frontend", "build", "index.html"))
  );
} else {
  app.get("/", (req, res) => {
    res.send("API is running..");
  });
}
// --------------------------deployment------------------------------
// Error Handling middlewares
app.use(notFound);
app.use(errorHandler);
const PORT = process.env.PORT;
const server = app.listen(
  PORT,
  console.log(`Server running on PORT ${PORT}...`.yellow.bold)
);
const io = require("socket.io")(server, {
  pingTimeout: 60000,
  cors: {
    origin: "http://localhost:3000",
    // credentials: true,
  },
});
io.on("connection", (socket) => {
  console.log("Connected to socket.io");
  socket.on("setup", (userData) => {
    socket.join(userData._id);
```

```
    socket.emit("connected");
  });
  socket.on("join chat", (room) => {
    socket.join(room);
    console.log("User Joined Room: " + room);
  });
  socket.on("typing", (room) => socket.in(room).emit("typing"));
  socket.on("stop typing", (room) => socket.in(room).emit("stop typing"));
  socket.on("new message", (newMessageRecieved) => {
    var chat = newMessageRecieved.chat;
    if (!chat.users) return console.log("chat.users not defined");
    chat.users.forEach((user) => {
      if (user._id == newMessageRecieved.sender._id) return;
      io.to(user._id).emit("message recieved", newMessageRecieved);
    });
  });

  socket.off("setup", () => {
    console.log("USER DISCONNECTED");
    socket.leave(userData._id);
  });
});
```

Лістинг Server.js:

```
import { Button } from "@chakra-ui/button";
import { FormControl, FormLabel } from "@chakra-ui/form-control";
import { Input, InputGroup, InputRightElement } from "@chakra-ui/input";
import { VStack } from "@chakra-ui/layout";
import { useState } from "react";
import axios from "axios";
import { useToast } from "@chakra-ui/react";
import { useHistory } from "react-router-dom";
import { ChatState } from "../../Context/ChatProvider";
import { GoogleLogin } from "react-google-login";
const Login = ({ boxColor }) => {
```

```
const { setUser } = ChatState();
const [show, setShow] = useState(false);
const handleClick = () => setShow(!show);
const toast = useToast();
const [email, setEmail] = useState();
const [password, setPassword] = useState();
const [loading, setLoading] = useState(false);
const history = useHistory();
const submitHandler = async () => {
 setLoading(true);
 if (!email || !password) {
  toast({
   title: "Please Fill all the Fields",
   status: "warning",
   duration: 5000,
   isClosable: true,
   position: "bottom",
  });
  setLoading(false);
  return;
 }
 try {
  const config = {
   headers: {
    "Content-type": "application/json",
   },
  };
  const { data } = await axios.post(
   "/api/user/login",
   { email, password },
   config
  );
  toast({
   title: "Login Successful",
   status: "success",
```

```
      duration: 5000,
      isClosable: true,
      position: "bottom",
    });
    setUser(data);
    localStorage.setItem("userInfo", JSON.stringify(data));
    setLoading(false);
    history.push("/chats", { boxColor: boxColor });
  } catch (error) {
    toast({
      title: "Error Occurred!",
      description: error.response.data.message,
      status: "error",
      duration: 5000,
      isClosable: true,
      position: "bottom",
    });
    setLoading(false);
  }
};

const responseGoogle = (response) => {
  // Handle the Google login response here
  console.log(response);
};

return (
  <VStack spacing="10px">
    <FormControl id="email" isRequired>
      <FormLabel color={boxColor === "black" ? "whatsapp.100" : "black"}>
        Email Address
      </FormLabel>
      <Input
        value={email}
        type="email"
```

```
    placeholder="Enter Your Email Address"
    onChange={(e) => setEmail(e.target.value)}
    color={boxColor === "black" ? "whatsapp.100" : "black"}
   />
</FormControl>
<FormControl id="password" isRequired>
  <FormLabel color={boxColor === "black" ? "whatsapp.100" : "black"}>
    Password
  </FormLabel>
  <InputGroup size="md">
    <Input
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      type={show ? "text" : "password"}
      placeholder="Enter password"
      color={boxColor === "black" ? "whatsapp.100" : "black"}
    />
    <InputRightElement width="4.5rem">
      <Button h="1.75rem" size="sm" onClick={handleClick}>
        {show ? "Hide" : "Show"}
      </Button>
    </InputRightElement>
  </InputGroup>
</FormControl>
<Button
  colorScheme="blue"
  width="100%"
  style={{ marginTop: 15 }}
  onClick={submitHandler}
  isLoading={loading}
>
  Login
</Button>
<GoogleLogin
  clientId="1040774597536-1ffi7kbmfg98b25p0cl8qckka0skt137.apps.googleusercontent.com"
```

```
        buttonText="Sign in with Google"
        onSuccess={responseGoogle}
        onFailure={responseGoogle}
        cookiePolicy={"single_host_origin"}
        render={(renderProps) => (
          <Button
            colorScheme="blue"
            width="100%"
            style={{ marginTop: 15 }}
            onClick={renderProps.onClick}
            disabled={renderProps.disabled || loading}
          >
            Sign in with Google
          </Button>
        )}
      />
    </VStack>
  );
};

export default Login;
```

Лістинг MessageControlers.js

```
const asyncHandler = require("express-async-handler");
const Message = require("../models/messageModel");
const User = require("../models/userModel");
const Chat = require("../models/chatModel");
//@description     Get all Messages
//@route           GET /api/Message/:chatId
//@access          Protected
const allMessages = asyncHandler(async (req, res) => {
  try {
    const messages = await Message.find({ chat: req.params.chatId })
      .populate("sender", "name pic email")
      .populate("chat");
```

```
    res.json(messages);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});
//@description     Create New Message
//@route           POST /api/Message/
//@access          Protected
const sendMessage = asyncHandler(async (req, res) => {
  const { content, chatId } = req.body;
  if (!content || !chatId) {
    console.log("Invalid data passed into request");
    return res.sendStatus(400);
  }
  var newMessage = {
    sender: req.user._id,
    content: content,
    chat: chatId,
  };
  try {
    var message = await Message.create(newMessage);
    message = await message.populate("sender", "name pic").execPopulate();
    message = await message.populate("chat").execPopulate();
    message = await User.populate(message, {
      path: "chat.users",
      select: "name pic email",
    });
    await Chat.findByIdAndUpdate(req.body.chatId, { latestMessage: message });
    res.json(message);
  } catch (error) {
    res.status(400);
    throw new Error(error.message);
  }
});
```

```
module.exports = { allMessages, sendMessage };
```

Лістинг MyChats.js:

```
import { AddIcon } from "@chakra-ui/icons";
import { Box, Stack, Text, Flex } from "@chakra-ui/layout";
import { useToast } from "@chakra-ui/toast";
import axios from "axios";
import { useEffect, useState } from "react";
import { getSender } from "../config/ChatLogics";
import ChatLoading from "./ChatLoading";
import GroupChatModal from "./miscellaneous/GroupChatModal";
import { Button } from "@chakra-ui/react";
import { Image } from "@chakra-ui/react";
import { Avatar } from "@chakra-ui/avatar";
import { ChatState } from "../Context/ChatProvider";
import CryptoJS from "crypto-js";

const MyChats = ({ fetchAgain, boxColor }) => {
  const [loggedUser, setLoggedUser] = useState();
  const { selectedChat, setSelectedChat, user, chats, setChats } = ChatState();
  const toast = useToast();

  // Function to fetch chats from the server
  const fetchChats = async () => {
    try {
      const config = {
        headers: {
          Authorization: `Bearer ${user.token}`,
        },
      };

      const { data } = await axios.get("/api/chat", config);
      setChats(data);
    } catch (error) {
      toast({
```

```
      title: "Error Occurred!",
      description: "Failed to Load the chats",
      status: "error",
      duration: 5000,
      isClosable: true,
      position: "bottom-left",
    });
  }
};

// Function to start polling for new messages
const startPolling = async () => {
  await fetchChats();
  setInterval(fetchChats, 700); // Fetch chats every 5 seconds (adjust the interval as needed)
};

useEffect(() => {
  const fetchData = async () => {
    setLoggedUser(JSON.parse(localStorage.getItem("userInfo")));
    await fetchChats();
  };

  fetchData();
  // eslint-disable-next-line
}, [fetchAgain]);

useEffect(() => {
  startPolling(); // Start polling for new messages
  // eslint-disable-next-line
}, []);

// Function to format the message time
const formatMessageTime = (time) => {
  const currentDate = new Date();
  const messageDate = new Date(time);
```

```
    const millisecondsPerDay = 24 * 60 * 60 * 1000;
    const daysAgo = Math.floor((currentDate - messageDate) / millisecondsPerDay);
    if (daysAgo === 0) {
      // Today
      return messageDate.toLocaleTimeString([], {
        hour: "2-digit",
        minute: "2-digit",
        hour12: false,
      });
    } else if (daysAgo === 1) {
      // Yesterday
      return "Yesterday";
    } else if (daysAgo < 7) {
      // This week
      const weekday = messageDate.toLocaleDateString([], { weekday: "short" });
      return weekday;
    } else {
      // Older than this week
      const day = messageDate.getDate().toString().padStart(2, "0");
      const month = (messageDate.getMonth() + 1).toString().padStart(2, "0");
      const year = messageDate.getFullYear().toString().substr(-2);
      return `${day}.${month}.${year}`;
    }
  };
  // Function to decrypt the message content
  const decryptMessageContent = (encryptedMessage) => {
    const secretKey = "yFSX0L7JmGtQPIWf";
    const bytes = CryptoJS.AES.decrypt(encryptedMessage, secretKey);
    const decryptedMessage = bytes.toString(CryptoJS.enc.Utf8);
    console.log(decryptedMessage);
    // Regex patterns for different types of file extensions
    const imageRegex = /(\.(png|jpg|jpeg))/i; // Case-insensitive regex pattern for image files
    const videoRegex = /(\.(mp4|mkv|webm|ogg|m4v|f4v|mov|mpeg|avi|wmv))/i; // Case-insensitive regex pattern for
video files
    const musicRegex = /(\.(mp3|wav|ogg|aac|m4a|wma|flac|aiff|opus))/i;
    const docRegex = /(\.(doc|docx|xls|xlsx|ppt|pptx|txt))/i;
```

```
  const docZip = /(\.(zip|rar|7zip))/i;
  const docGif = /(\.(gif))/i;
  const docPdf = /(\.(pdf))/i;

  if (imageRegex.test(decryptedMessage)) {
    return "Image🖼";
  } else if (videoRegex.test(decryptedMessage)) {
    return "Video📹";
  } else if (musicRegex.test(decryptedMessage)) {
    return "Audio🔊";
  } else if (docRegex.test(decryptedMessage)) {
    return "doc📄";
  } else if (docZip.test(decryptedMessage)) {
    return "Archive🗃";
  } else if (docGif.test(decryptedMessage)) {
    return "Gif🖼";
  } else if (docPdf.test(decryptedMessage)) {
    return "pdf📄";
  } else if (decryptedMessage.length > 10) {
    return decryptedMessage.slice(0, 10) + "...";
  } else {
    return decryptedMessage;
  }
};
return (
  <Box
    display={{ base: selectedChat ? "none" : "flex", md: "flex" }}
    flexDir="column"
    alignItems="center"
    p={3}
    bg={boxColor || "white"}
    w={{ base: "100%", md: "31%" }}
    borderRadius={boxColor ? "lg" : "0px"}
    borderWidth={boxColor ? "1px" : "0px"}
  >
```

```
<Box
  pb={3}
  px={3}
  fontSize={{ base: "28px", md: "30px" }}
  fontFamily="Work sans"
  display="flex"
  w="100%"
  justifyContent="space-between"
  alignItems="center"
  color={boxColor === "black" ? "whatsapp.100" : "black"}
>
  My Chats
  <GroupChatModal user={user} chats={chats} setChats={setChats}>
    <Button
      display="flex"
      fontSize={{ base: "17px", md: "10px", lg: "17px" }}
      rightIcon={<AddIcon />}
      color={"black"}
    >
      New Group Chat
    </Button>
  </GroupChatModal>
</Box>
<Box
  display="flex"
  flexDir="column"
  p={3}
  bg={"#d4d4d4"}
  w="100%"
  h="100%"
  borderRadius={boxColor ? "lg" : "0px"}
  overflowY={boxColor ? "hidden" : "auto"}
  css={
    boxColor
      ? {}
```

```
    : {
      "&::-webkit-scrollbar": {
        width: "6px",
      },
      "&::-webkit-scrollbar-track": {
        background: "#F8F8F8",
      },
      "&::-webkit-scrollbar-thumb": {
        background: "#888",
      },
      "&::-webkit-scrollbar-thumb:hover": {
        background: "#555",
      },
    }
  }
>
  {chats ? (
    <Stack overflowY="scroll" spacing={4}>
      {chats.map((chat) => (
        <Box
          onClick={() => setSelectedChat(chat)}
          cursor="pointer"
          bg={selectedChat === chat ? "#3182CE" : "#ffffff"}
          color={selectedChat === chat ? "white" : "black"}
          px={3}
          py={2}
          borderRadius="lg"
          key={chat._id}
        >
          <Flex justify="space-between" align="center">
            <Flex align="center">
              <Avatar
                size="lg"
                cursor="pointer"
                name={chat.chatName}
```

```
          src={
            chat.isGroupChat
              ? chat.chatPic
              : loggedUser &&
                chat.users.find((user) => user._id !== loggedUser._id)?.pic
          }
        />
        <Box ml={2}>
          <Text fontWeight="bold">
            {!chat.isGroupChat
              ? getSender(loggedUser, chat.users)
              : chat.chatName}
          </Text>
          {chat.latestMessage && (
            <Text fontSize="sm" mt={1}>
              <span
                style={{
                  color: selectedChat === chat ? "white" : "blue",
                }}
              >
                {chat.latestMessage.sender.name}:
              </span>{" "}
              {decryptMessageContent(chat.latestMessage.content)}
            </Text>
          )}
        </Box>
      </Flex>
      {chat.latestMessage && (
        <Text fontSize="xs" textAlign="right">
          {formatMessageTime(chat.latestMessage.createdAt)}
        </Text>
      )}
    </Flex>
  </Box>
))}
```

```
          </Stack>
        ) : (
          <ChatLoading />
        )}
      </Box>
    </Box>
  );
};
export default MyChats;
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему: «Розробка програми месенджера за допомогою MERN Stack»

студента групи 122-20-ск1

Дмитрієва Олексія Ярославовича

Керівник економічного розділу

доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна_робота_ Дмитрієв.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна_робота_ Дмитрієв.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Link-UA.rar	Архів. Містить коди програми і відкомпільовану програму
Презентація	
Презентація_ Дмитрієв.ppt	Презентація кваліфікаційної роботи