МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Веб-месенджер»

Виконав студент групи ІПЗ-20-4
ВАСЬКІВСЬКИЙ Віталій Юрійович

Керівник роботи:
ГОРДІЄНКО Ірина Валеріївна

Консультант:
ЧИЖМОТРЯ Олена Геннадіївна

Рецензент:
КРАВЧЕНКО Світлана Миколаївна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
Завідувач кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: ВАСЬКІВСЬКИЙ Віталій Юрійович
Керівник роботи: ГОРДІЄНКО Ірина Валеріївна
Тема роботи: «Веб-месенджер»,
затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74с
Вихідні дані для роботи: Об’єктом дослідження є використання інформаційних технологій для обміну повідомленнями у мережі інтернет.
Предметом дослідження є використання сучасних та веборієнтованих технологій розробки для створення вебзастосунку, що допомагає обмінюватися повідомленнями у мережі інтернет.
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	Завдання видав	Завдання прийняв
1	Гордієнко І. В. Чижмотря О. Г.	27.02.2024	27.02.2024
2	Гордієнко І. В. Чижмотря О. Г.	27.02.2024	27.02.2024
3	Гордієнко І. В. Чижмотря О. Г.	27.02.2024	27.02.2024

РЕФЕРАТ

Кваліфікаційна робота складається з програмного продукту веб-месенджер та пояснювальної записки.

Пояснювальна записка до випускної роботи містить 71 сторінка, 35 ілюстрацій, 12 таблиць, 5 додатків та 31 літературне джерело.

Метою роботи є розробка веб-месенджеру для обміну повідомленнями у мережі інтернет.

У роботі були визначені основні завдання на розробку програми, проаналізовано аналогічні програмні продукти. Було обґрунтовано використання архітектури програми та наведено сценарії її роботи. Також узагальнено структуру програмного комплексу, надано опис алгоритмів взаємодії окремих модулів системи та об’єктної структури системи. Крім того, подано результати тестування системи.

КЛЮЧОВІ СЛОВА: C#, REACT, SPA, API, ВЕБ-ДОДАТОК.

ABSTRACT

The bachelor’s thesis consists of the web-messenger software product and an explanatory note.

The explanatory note to the final thesis contains 71 pages, 35 illustrations, 12 tables, 5 appendices and 31 literary sources.

The purpose of the work is to develop messanger to exchange messages in network.

The main development tasks were defined in the work, analogous software products were analyzed. The use of the program architecture was justified, and scenarios of its operation were provided. Additionally, the structure of the software complex was summarized, descriptions of the interaction algorithms of individual system modules and the object structure of the system were provided. Furthermore, the results of the system testing were presented.

KEY WORDS: C#, REACT, SPA, API, WEB APPLICATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

MVC – Модель–вигляд–контролер (або Модель–представлення–контролер, Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ВСТУП

Актуальність теми. У сучасному світі роль месенджерів стабільно висока з ростом цифрової комунікації. З огляду на зростаючу потребу у спілкуванні онлайн, месенджери залишаються важливим інструментом для особистої та професійної комунікації. Вони не лише сприяють зручності спілкування, але й впливають на розвиток бізнесу, навчання та інші аспекти життя. Також, зростанням мобільних технологій та зміною способу сприйняття інформації, месенджери стають платформою для новацій та розвитку цифрового середовища. Тому, вивчення актуальних аспектів месенджерів залишається важливим для розуміння та вдосконалення способів комунікації в онлайн-середовищі.

Метою кваліфікаційної роботи є розробка веб-месенджеру.

Ця мета включає виконання таких завдань:

- аналіз вимог до програми;
- дослідження аналогічних програмних продуктів;
- вибір архітектури системи;
- обґрунтування та вибір засобів реалізації;
- визначення варіантів використання та проектування об’єктноорієнтованої структури додатку;
- розробка алгоритмів застосунку;
- безпосередня реалізація додатка;
- опис структури інтерфейсу;
- тестування створеного додатку.

Об’єктом дослідження є використання інформаційних технологій для обміну повідомленнями у мережі.

Предметом дослідження є використання сучасних веборієнтованих технологій розробки для створення веб-застосунку, що допомагає обмінюватися повідомленнями у мережі.

Методи дослідження. У роботі було використано методи, функціонального, об’єктно-орієнтованого та аспектно-орієнтованого програмувань.

Практичне значення одержаних результатів. Розроблений веб-застосунок може використовуватися для особистої та професійної комунікації.

За матеріалами кваліфікаційної роботи були опубліковані тези: Васьківський В. Ю., Овсеюков Є. Ю., Окунькова О. О., Розробка ВЕБ-меседжера. Тези V Всеукраїнської науково-технічної конференції «Комп’ютерні технології: інновації, проблеми, рішення» 01-02 грудня 2022 року. Житомир: «Житомирська політехніка», 2022. С.10-11 [1].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-МЕСЕНДЖЕРУ

1.1 Постановка задачі

Розробка веб-месенджеру передбачає створення бекенд частини за допомогою мови C# [2] та фреймворку ASP.NET [3]. Фронтенд частина має бути реалізовани за допомогою мови JavaScript [14] та бібліотеки React [18].

Отриманий застосунок має бути веб-додатком, що надає можливість користувачам обмінюватися повідомленнями у мережі інтернет.

Програмний продукт повинен мати наступні функціональні характеристики:

- авторизація та реєстрація;
- створювати персональні чати та групи;
- пошук по чатам;
- надсилати текстові, медіа та файлові повідомлення;
- надсилати, редагувати, видаляти, пересилати, відповідати на повідомлення;
- керування активними логін сесіями;

Програмний продукт має бути реалізований на основі клієнт-серверної архітектури. Серверна частина додатку повинна використовувати веб-сервер IIS і систему керування базами даних Microsoft SQL Server [4]. При розробці програмного комплексу слід застосувати архітектуру SPA (Single-Page Application), щоб оновлювати контент без перезавантаження веб-сторінки.

Клієнтська частина повинна бути створена відповідно до сучасних стандартів розробки односторінкових додатків. Це означає, що весь необхідний код завантажується разом зі сторінкою або динамічно підвантажується в міру потреби, зазвичай у відповідь на дії користувача. Під час роботи сторінка не оновлюється і не перенаправляє користувача на інші сторінки.

Розробка передбачає наступні основні етапи:

1. Створення бекенд частини додатку написанням GraphQL API [5], використовуючи фреймворк ASP.NET [3] та бібліотеку GraphQL.NET [6], що включає реалізацію наступних сутностей:
   - «Чати»;
   - «Повідомлення»;
   - «Прочитані повідомлення»;
   - «Сесії»;
   - «Користувачі у чаті»;
   - «Користувачі»;
2. Створення інтерфейсу, з яким користувачі будуть взаємодіяти:
   - вибрати бібліотеку для створення користувацького інтерфейсу;
   - розробити відповідні сторінки з використанням обраної бібліотеки;
   - забезпечити користувачеві сповіщення про завантаження контенту за допомогою відповідного компонента.
3. Реалізувати отримання даних зі створеного API на клієнті та налагодити маршрутизацію додатку за допомогою бібліотеки React [18]:
   - налаштувати виклик API через Apollo Client [21];
   - організувати маршрутизацію додатку з використанням React Router Dom [19];

Після завершення цих кроків буде розроблена система, побудована відповідно до архітектури SPA (Single-Page Application), що забезпечить користувачеві зручний додаток для спілкування.

1.2 Аналіз аналогів програмного продукту

У сучасному світі складно вигадати щось абсолютно нове, що ще не існувало раніше. Тому під час розробки програм необхідно звертати увагу на аналогічні продукти, щоб отримати повну картину ринку і вимог користувачів до цих додатків. Для проведення аналізу було вибрано наступні програми:

1. Telegram.

2. WhatsApp.
3. Viber.

Telegram - це безпечний месенджер з широким функціоналом для обміну повідомленнями та спілкування.

Зображено додаток Telegram (рис. 1.1).

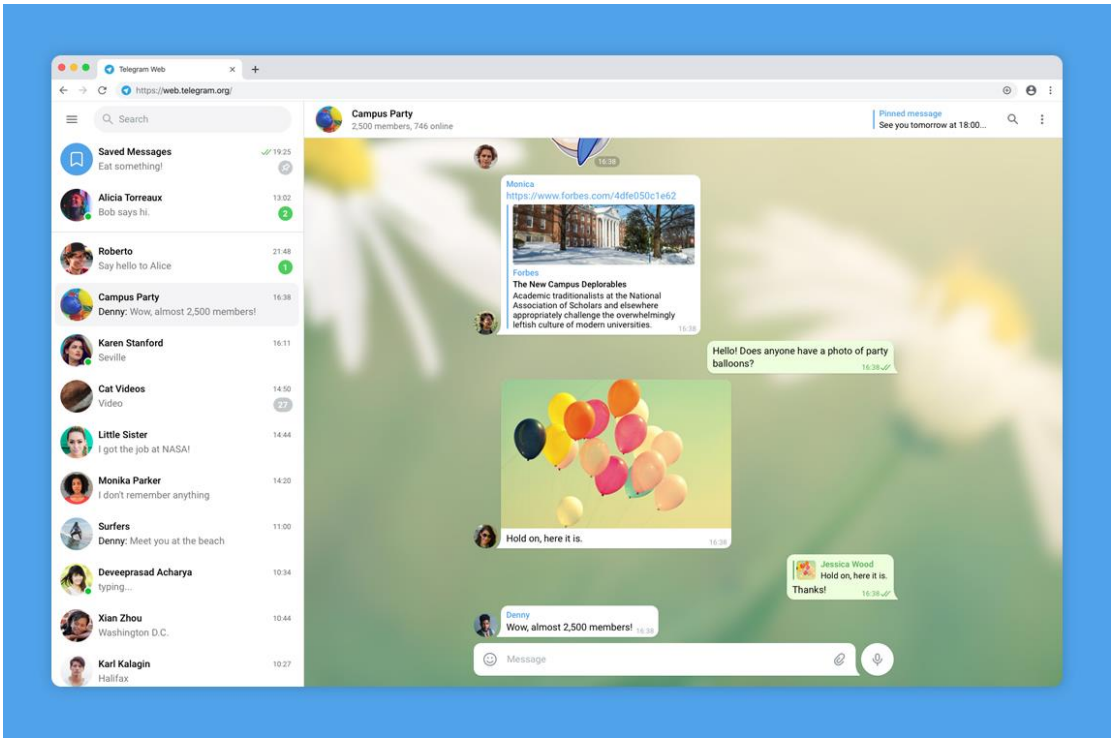

Рисунок 1.1 - Telegram

Можна підкреслити наступні переваги даного додатку:

– гарний та інтуїтивний дизайн;
– висока безпека;
– широкий функціонал;
– швидкість і ефективність;
– самознищення повідомлень;

До недоліків можна віднести наступне:

– неуніверсальність шифрування;
– несприйняття в деяких країнах.

WhatsApp - це популярний месенджер для обміну повідомленнями, фото, відео та здійснення голосових та відеодзвінків у мережі.

Зображено додаток WhatsApp (рис. 1.2).

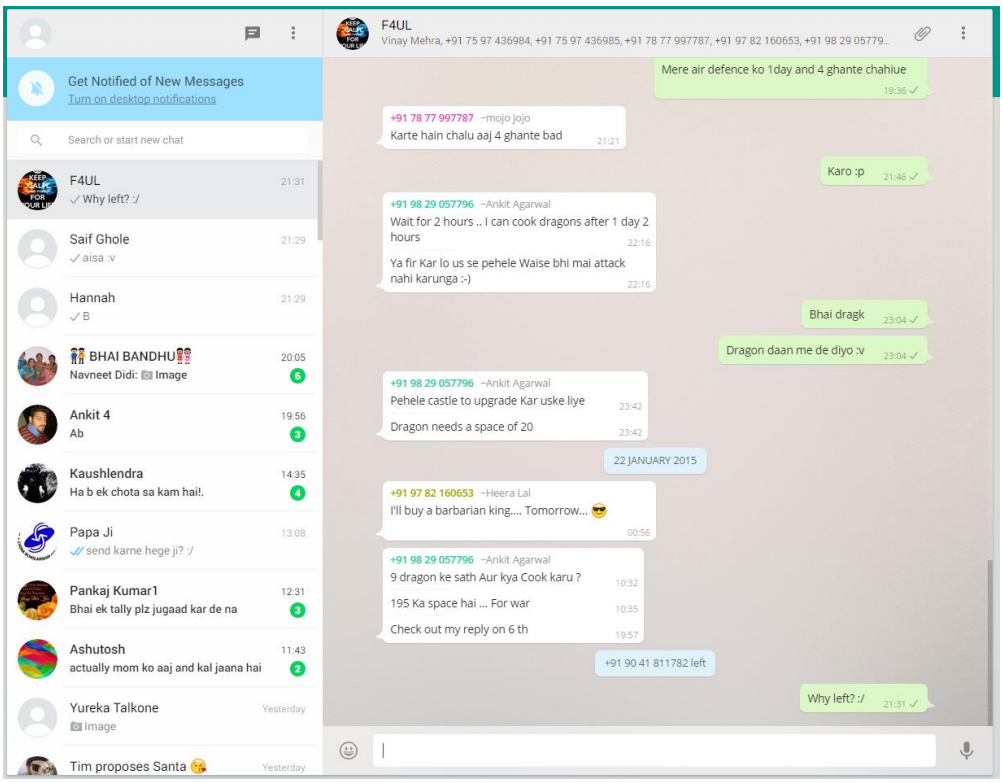

Рисунок 1.2 - WhatsApp

До переваг можна віднести:

– широкий функціонал;

– шифрування end-to-end;

– популярність.

Недоліками є наступне:

– обмеженість функцій веб-версії;

– обмеження на використання одного облікового запису на одному пристрої;

– недостатній функціонал для бізнесу.

Viber - це месенджер для обміну повідомленнями, викликів і відеодзвінків у мережі.

Зображено додаток Viber (рис. 1.3).

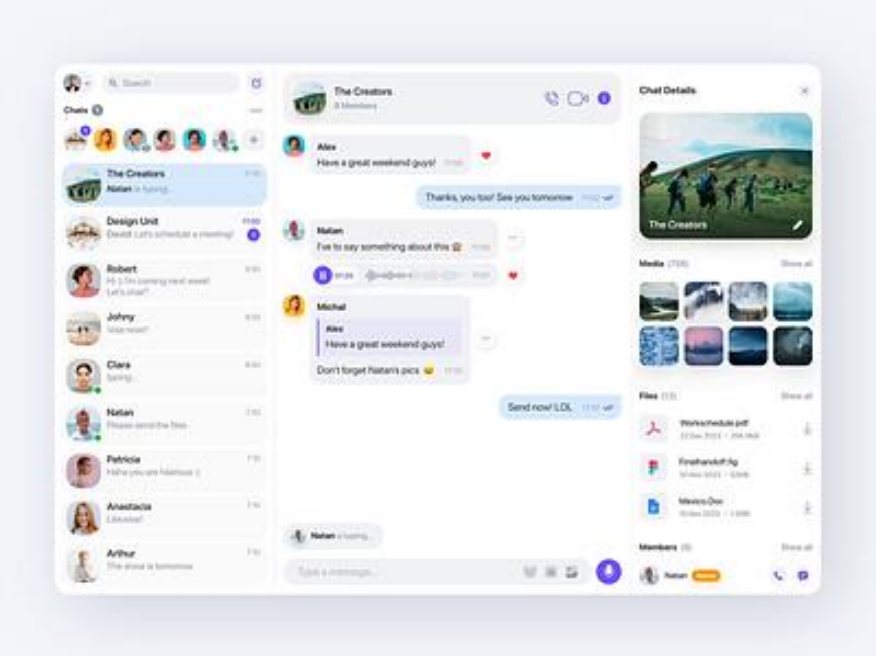

Рисунок 1.3 - Viber

Перевагами є наступне:

– шифрування end-to-end;

– велика кількість користувачів.

До недоліків можна віднести наступне:

– високе споживання ресурсів;

– можливість отримання небажаних повідомлень.

Структуруємо отримані дані за допомогою таблиці (табл. 1.1).

Таблиця 1.1

Аналіз аналогів програмного продукту

Характеристика	Telegram	Whatsup	Viber
Інтуїтивний та зрозумілий дизайн	+	+	+
Висока безпека	+	+	+
Широка інтеграція	+	-	-
Висока стабільність та швидкість	+	-	-
Мале споживання ресурсів	+	+	-

На основі отриманих результатів, визначаємо наступні критерії для розроблюваного додатку:

– інтуїтивний та зрозумілий дизайн;

– висока безпека;
– широка інтеграція;
– висока стабільність та швидкість;
– мале споживання ресурсів;

1.3 Вибір архітектури веб-додатку

У сучасному світі існує безліч архітектур для побудови веб-додатків. Розглянемо кілька з них, зокрема:

– схема MVC (Model-View-Controller);
– схема SPA (Single-Page Application).

Тепер детально розглянемо кожну з цих архітектур.

MVC - це архітектурний шаблон, що використовується для розробки програмного забезпечення, який розділяє програму на три основні компоненти: модель (Model), представлення (View) і контролер (Controller). Кожен з цих компонентів відповідає за певні аспекти функціональності програми: модель представляє дані та бізнес-логіку, представлення відображає інформацію користувачу, а контролер обробляє вхідні дані та керує взаємодією між моделлю та представленням. Цей підхід дозволяє відокремлювати різні аспекти програми та полегшує розробку, тестування та підтримку програмного забезпечення.

Зображено схему MVC (рис. 1.4).

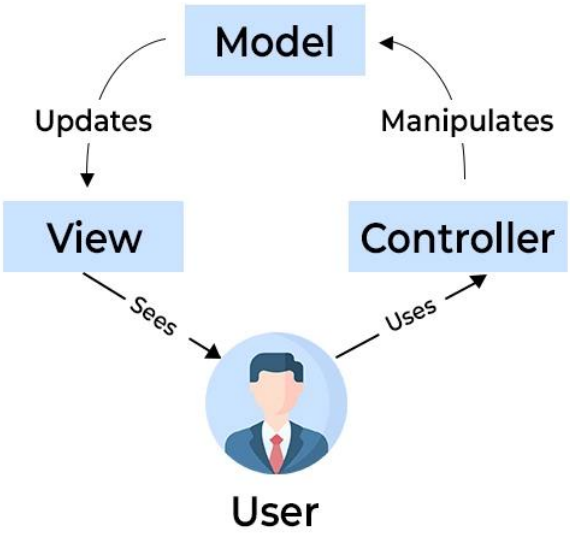

Рисунок. 1.4 - Схема MVC

До його переваг належать:

- розділення відповідальностей: MVC дозволяє розділити програму на три основні компоненти, що полегшує управління кодом та забезпечує чітку структуру;
- покращена повторне використання коду: кожен компонент (Модель, Вид, Контролер) може бути використаний повторно в інших частинах програми або в інших проектах;
- легша тестовість: кожен компонент може бути тестований окремо, що дозволяє проводити модульне тестування та забезпечує високу якість програмного забезпечення.

До недоліків можна віднести наступне:

- можливість перенаповнення контролерів: контролери можуть стати перенаповненими, якщо в них занадто багато логіки, що може знизити читабельність і підтримку коду;
- переплутаність взаємодії між компонентами: іноді взаємодія між моделлю, видом і контролером може бути неявною або неочевидною, що може призвести до проблем у розумінні і обслуговуванні коду.

SPA – це тип веб-додатка або сайту, який працює на одній HTML-сторінці. У SPA весь необхідний HTML, CSS [29] та JavaScript [14] завантажується один раз при початковому завантаженні сторінки, після чого весь інший контент та взаємодія з користувачем відбувається без повторної загрузки сторінки з сервера.

Зображено схему SPA (рис. 1.5).

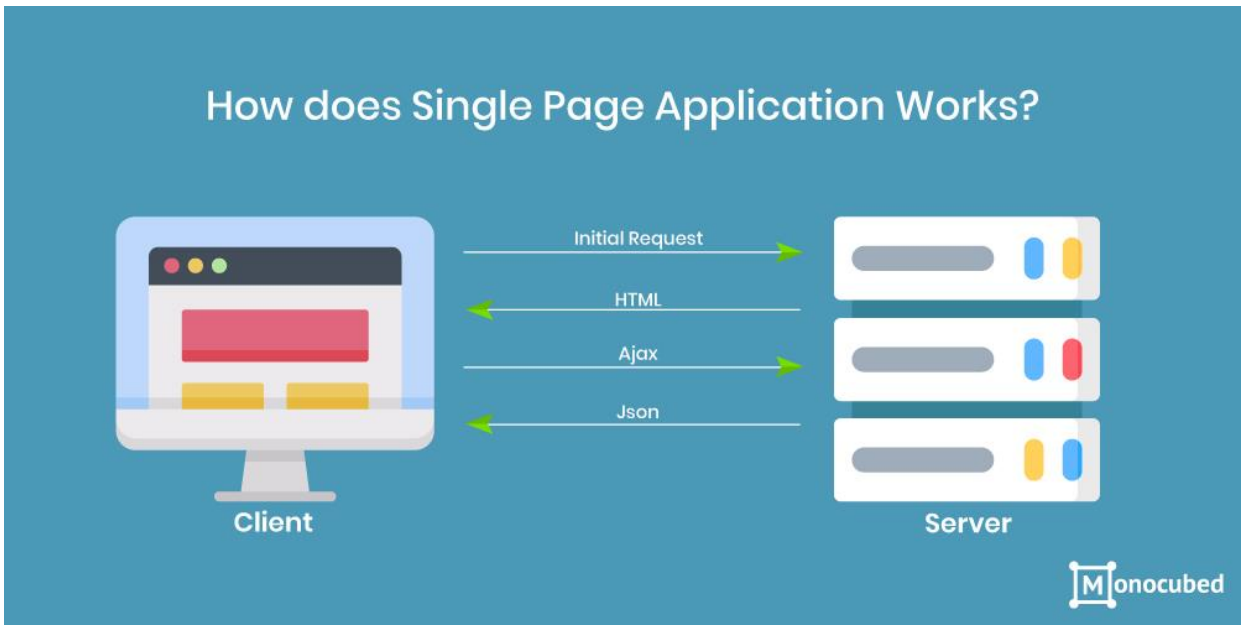

Рисунок 1.5 - Схема SPA

До його переваг належать:

– швидкість: SPA завантажує всі необхідні ресурси при запуску та працює в браузері, що забезпечує швидкість реакції на дії користувача;
– зменшення навантаження на сервер: через те, що не потрібно завантажувати повну сторінку з сервера при кожному запиті, навантаження на сервер зменшується;
– кешування: SPA можуть легко використовувати кешування, що дозволяє зберігати ресурси на боці клієнта і зменшувати час завантаження;
– можливість використання фреймворків: багато фреймворків та бібліотек спеціалізуються на розробці SPA, що дозволяє розробникам використовувати готові рішення для розробки складних клієнтських додатків.

До недоліків належать:

– збільшене використання пам'яті: завантаження всіх ресурсів одразу може призвести до збільшеного використання пам'яті в браузері, що може вплинути на продуктивність, особливо на старіших або менш потужних пристроях;

– проблеми з SEO: для пошукових систем складно індексувати та інтерпретувати вміст, що генерується динамічно через JavaScript, що може призвести до проблем з оптимізацією для пошукових систем (SEO).

Проаналізувавши вищеперелічені архітектури, було обрано використовувати підхід SPA для забезпечення кращого користувацького досвіду.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для створення додатку веб-месенджер обрано наступні технології:

- мова розмітки гіпертекстових документів: HTML5;
- препроцесор стилів: Sass [30];
- клієнтська частина: мова програмування JavaScript та бібліотека React;
- серверна частина: мова програмування C# [2] та фреймворк ASP.NET [3];
- база даних: Microsoft SQL Server [4].

Ці технології дозволять створити веб-додаток за допомогою сучасних інструментів. HTML5 забезпечить семантичну структуру веб-сторінок, Sass дозволить ефективно керувати стилями з використанням препроцесінгу, а React надасть потужні можливості для реалізації інтерактивної взаємодії з користувачем.

Серверна частина буде побудована на базі мови програмування C# та фреймворка ASP.NET, що дозволить створити потужний та масштабований серверний додаток. Використання бази даних Microsoft SQL Server забезпечить надійне зберігання та управління даними..

Використання цих технологій сприятиме створенню сучасного та функціонального веб-месенджера, забезпечуючи оптимальне використання ресурсів та гнучкість у процесі розробки.

HTML5 - використовується для створення структуризованих інтерактивних веб-сторінок і додатків.

Sass [30] – це препроцесор CSS, використовується для полегшення написання та організації CSS коду, забезпечуючи більшу гнучкість і ефективність у веб-розробці..

React [18] – використовується для створення інтерактивних та швидкодіючих користувацьких інтерфейсів у веб-додатках, спрощуючи розробку складних клієнтських додатків за допомогою компонентної архітектури.

C# [2] – використовується для створення серверної частини додатків і є потужною та об'єктно-орієнтованою мовою програмування з розширеним набором функціональних можливостей.

ASP.NET [3] - це технологічний стек для розробки веб-додатків. Він надає потужні інструменти для обробки запитів, маршрутизації та роботи з базами даних.

Microsoft SQL Server [4] – це система управління базами даних. Вона надає можливості для зберігання, керування та обробки даних в реляційних базах даних.

Ці технології допоможуть створити функціональний, зручний у використанні та масштабований веб-додаток.

Наведено порівняння максимально можливого обсягу збережених даних для кожної СУБД (табл. 1.2).

Таблиця 1.2

Порівняння максимально можливого обсягу збережених даних

	Розмір БД	Розмір таблиці	Розмір рядка
MySQL	∞	256 ТБ	64 КВ
MS SQL	524258 ТБ	524258 ТБ	∞
PostgreSQL	∞	32 ТБ	1.6 ТБ

За критерієм тригери й збережені процедури всі альтернативи ідентичні. Усі підтримують тригери, процедури й функції.

Порівняємо різні методи захисту даних, що застосовуються для запобігання несанкціонованого доступу до конфіденційної інформації в інформаційній системі організації. Потім розглянемо таблицю, в якій перераховані системи захисту даних та надано їх опис.

Наведено аналіз систем забезпечення безпеки даних в альтернативах (табл. 1.3).

Таблиця 1.3

Аналіз систем забезпечення безпеки даних в альтернативах

	Ідентифікація	Захист від brute-force	Шифрування	Сертифікація безпеки
MySQL	+	-	+	-
MS SQL	+	-	+	+
PostgreSQL	+	+	+	+

Після аналізу різних систем управління базами даних (СУБД), таких як MySQL, Microsoft SQL Server (MS SQL) [4] та PostgreSQL, було визначено, що Microsoft SQL Server [4] найбільш підходить для виконання визначених завдань. Остання версія Microsoft SQL Server [4] має розширений функціонал та можливості, такі як підтримка географічних даних, розширення для складних операцій та засоби захисту даних.

Наведено аналіз загальних показників СУБД (табл. 1.4).

Таблиця 1.4

Аналіз загальних показників СУБД

	MySQL	MS SQL	PostgreSQL
1.	*2.*	*3.*	*4.*
Логічна модель даних	Реляційна	Реляційна	Постреляційна (об’єктно-реляційна)
Фізична модель даних	Сторінкова	Сторінкова	Сторінкова
Типи даних	Всі основні	Всі основні, розширені	Всі основні, довільні, множинні значення
Індекси	Всі основні, повнотекстовий	Всі основні, повнотекстовий	Звичайні, повнотекстовий
Мови маніпулювання	SQL	SQL, QBE, XQuery, багатомірні вирази (для OLAP)	Розширений SQL

Продовження табл. 1.4

1.	*2.*	*3.*	*4.*
Вбудовані мови програмування	ANSI C, ANSI C++	MS Visual Basic, C#	C, C++, Java, Perl, PHP, .Net та ін
Генератор форм, звітів	Ні	Засоби побудови звітів	Ні
Транзакції	Так (в тому числі розподілені)	Так	Так
Тригери, процедури, що зберігаються	Так	Так	Так (підтримка декількох мов програмування)
Платформи	MS Windows, Unix, Linux, MacOS X, Novell NetWare та ін.	Тільки MS Windows	MS Windows, Linux, FreeBSD, Solaris, MacOS
Область застосування	Інформаційні системи масштабу підприємства	Інформаційні системи масштабу підприємства	Інформаційні системи масштабу підприємства
Особливості	Можливість логічного об’єднання БД. Широкий вибір платформ. Реплікація	Масштабування. Кластери. Реплікації. Розширена підтримка XML	Відкриті вихідні тексти. Механізм наслідування. Масштабування. Реплікація.

Для створення веб-додатку були обрані наступні інструменти та технології:

- база даних: Misrosoft SQL Server [4];
- мова програмування на серверній частині: C# [2];
- фреймворк на серверній частині: ASP.NET [3];
- мова програмування на клієнтській частині: JavaScript [14];
- фреймворк на клієнтській частині: React [18];
- процесор стилів: Sass [30].

Для розробки були використані наступні інтегровані середовища розробки: середовище розробки Visual Studio 2022 та Visual Studio Code.

Висновки до першого розділу

У цьому розділі було сформульовано завдання для кваліфікаційної роботи, визначено функціональні вимоги та основні етапи роботи. Також було проведено аналіз аналогів продукту, що розробляється, що дозволило встановити наступні критерії для розробки: інтуїтивний та зрозумілий дизайн, висока безпека, широка інтеграція, висока стабільність та швидкість та мале використання ресурсів.

Деякі існуючі архітектурні рішення було проаналізовано, і вибір було зроблено на користь використання схеми SPA для забезпечення кращого користувацького досвіду. Також було обгрунтовано вибір технологічного стеку для реалізації додатку: ASP.NET [3] для побудови бекенд частини, React [18] - для фронтенд частини; Miscrosoft SQL Server [4] - як систему керування базами даних.

Буде використано такі інтегровані середовища розробки: середовище розробки Visual Studio 2022 та Visual Studio Code.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-МЕСЕНДЖЕРУ

2.1 Визначення варіантів використання та об’єктно-орієнтованої структури системи

Для зображення відношення між акторами та прецедентами в системі, було побудовано діаграму варіантів використання.

Зображено діаграму варіантів використання (рис. 2.1).

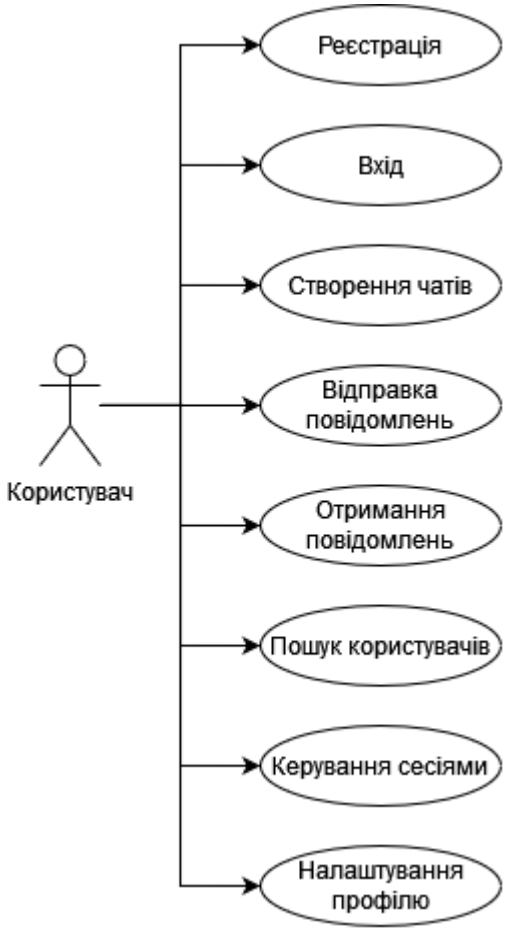

Рисунок 2.1 - Діаграма варіантів використання

Відповідно до побудованої діаграми можна виокремити наступні вимоги додатку.

Вимоги користувачів:

1. Можливість реєстрація та входу у власний аккаунт.
2. Можливість керувати чатами.
3. Відправляти та отримувати повідомлення
4. Можливість пошуку користувачів.

5. Можливість керувати власними логін сесіями.
6. Можливість налаштовувати власний профіль.

Функціональні вимоги:

1. Використання протоколу WebSocket [7] для обміну даними.
2. Створення функціоналу для обміну повідомленнями.
3. Підключення бібліотек для швидшої розробки.

Нефункціональні вимоги:

1. Сприйняття: інтерфейс повинен бути інтуїтивно зручним для користувача.
2. Продуктивність: система повинна підтримувати мінімум декілька тисяч користувачів, що одночасно обмінюються повідомленнями.

Зображено схему патерну «Репозиторій» (рис. 2.2).

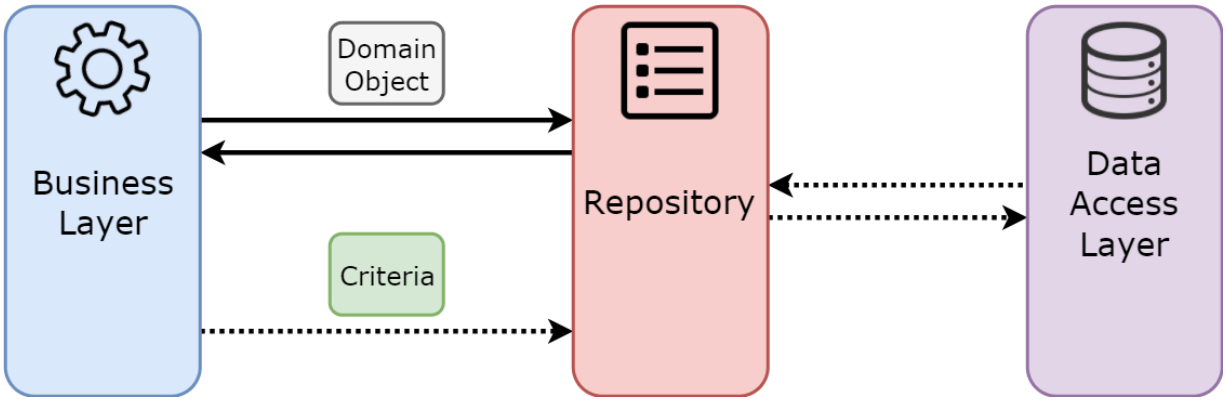

Рисунок 2.2 - Загальний вигляд патерну «Репозиторій»

Сучасна IT-сфера розвивається надзвичайно швидко, і багато часто виникаючих проблем вже вирішені завдяки «патернам проектування». Важко уявити сучасні додатки без їх використання, тому патерни проектування враховуються вже на етапі моделювання системи.

Таким чином, було вирішено застосувати патерн «Репозиторій», який є, мабуть, найпопулярнішим рішенням для застосунків.

«Репозиторії» – це класи або компоненти, які інкапсулюють логіку доступу до джерел даних. Вони централізують загальну функціональність доступу до даних, забезпечуючи кращу зручність обслуговування та відокремлюючи

інфраструктуру або технологію доступу до баз даних від рівня моделі предметної області.

Зображено діаграму класів провайдерів додатку (рис. 2.3).

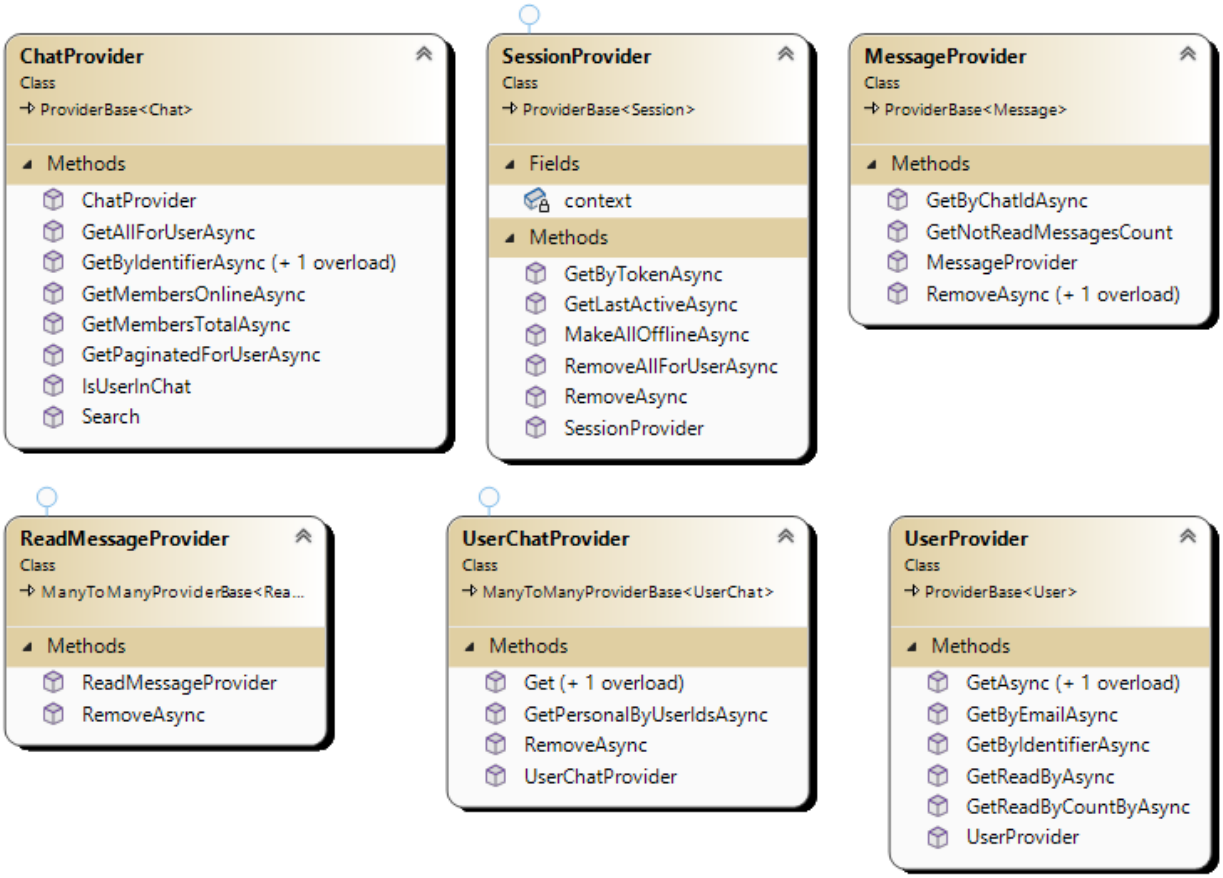

Рисунок 2.3 - Діаграма класів провайдерів додатку

На основі цього патерну спроектуємо діаграму класів провайдерів для кожної з сутностей додатку, а саме «ChatProvider», «SessionProvider», «MessageProvider», «ReadMessageProvider», «UserChatProvider», «UserProvider», які забезпечать структурований і зручний доступ до даних та функціональних можливостей.

Всі ці провайдери успадковуються від базового класу «ProviderBase», який містить загальний функціонал для управліннями провайдерами, наприклад, з’єднання з базою даних, логування, універсальні методи для отримання та модифікації даних тощо. Це дозволить спростити і стандартизувати роботу у системі.

Методи провадерів є асинхронними, що дозволяє програмі не блокувати основний потік і продовжувати обробляти запити від інших користувачів, тим самим підвищуючи продуктивність та ефективність застосунку.

2.2 Розробка бази даних системи

Для збереження даних була обрана СУБД Microsoft SQL Server з метою забезпечення надійності, ефективності та масштабованості системи, оскільки вона володіє широким функціоналом і підтримує великі обсяги даних.

Продемонстровано ER-діаграму спроєктованої бази даних (рис. 2.4).

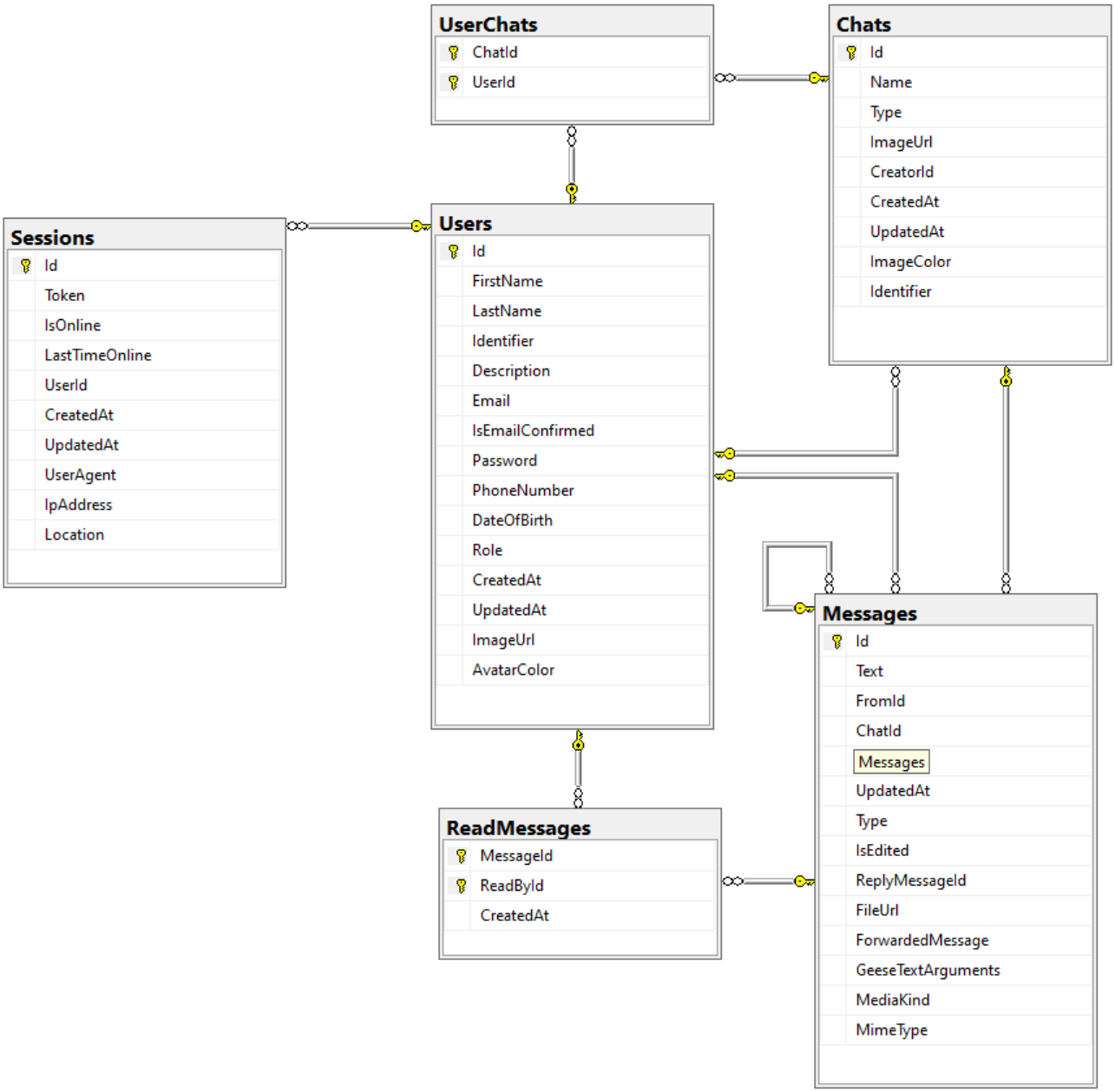

Рисунок 2.4 - ER-діаграма спроєктованої бази даних

Опис таблиць, полів та їх призначення наведені в таблицях 2.1-2.6.

Таблиця 2.1

Структура таблиці «Users»

Поле	Ключ	Тип	Призначення
1.	*2.*	*3.*	*4.*
Id	PK	uniqueidentifier	Ідентифікатор користувача
FirstName	-	varchar(450)	Ім’я користувача
LastName	-	varchar(450)	Прізвище користувача
Identifier	-	varchar(450)	Унікальне імя користувача
Description	-	varchar(max)	Опис користувача
Email	-	varchar(450)	Електронна пошта користувача
IsEmailConfirmed	-	bit	Чи підтвержєена пошта користувача
Password	-	varchar(450)	Пароль користувача
PhoneNumber	-	varchar(30)	Номер телефону користувача
DateOfBirth	-	datetime2(7)	Дата народження користувача
CreatedAt	-	datetime2(7)	Дата створення аккаунту
UpdatedAt	-	datetime2(7)	Дата редагування аккаунту
ImageUrl		varchar(450)	Посилання на картинку користувача
AvatarColor		varchar(7)	Колір користувача

Таблиця 2.2

Структура таблиці «Sessions»

Поле	Ключ	Тип	Призначення
1.	*2.*	*3.*	*4.*
Id	PK	uniqueidentifier	Ідентифікатор сесії
IsOnline	-	bit	Чи користувач онлайн
UserId	FK	uniqueidentifier	Ідентифікатор користувача для сесії (зовішній ключ на таблицю “Users”)
CreatedAt	-	datetime2(7)	Дата створення сесії
UpdatedAt	-	datetime2(7)	Дата редагування сесії
UserAgent	-	varchar(450)	User agent сесії
IpAddress	-	varchar(450)	IP-адреса сесії
Location	-	varchar(450)	Місце знаходження сесії

Таблиця 2.3

Структура таблиці «Chats»

Поле	Ключ	Тип	Призначення
1.	*2.*	*3.*	*4.*
Id	PK	uniqueidentifier	Ідентифікатор чату
Name	-	varchar(450)	Ім’я чату
Type	-	int	Тип чату
ImageUrl	-	varchar(450)	Посилання на картинку чату
CreatorId	FK	uniqueidentifier	Ідентифікатор користувача, який створив чат (зовішній ключ на таблицю “Users”)
CreatedAt	-	datetime2(7)	Дата створення чату
UpdatedAt	-	datetime2(7)	Дата редагування чату
ImageColor	-	varchar(7)	Колір чату
Identifier	-	varchar(450)	Унікальне імя чату

Таблиця 2.4

Структура таблиці «UserChats»

Поле	Ключ	Тип	Призначення
1.	*2.*	*3.*	*4.*
ChatId	PK	uniqueidentifier	Ідентифікатор чату (зовішній ключ на таблицю “Chats”)
UserId	PK	uniqueidentifier	Ідентифікатор користувача (зовішній ключ на таблицю “Users”)

Таблиця 2.5

Структура таблиці «Messages»

Поле	Ключ	Тип	Призначення
1.	*2.*	*3.*	*4.*
Id	PK	uniqueidentifier	Ідентифікатор повідомлення
Text	-	varchar(max)	Текст повідомлення
FromId	FK	uniqueidentifier	Ідентифікатор користувача, який створив повідомлення (зовішній ключ на таблицю “Users”)

Продовження табл. 2.5

1.	*2.*	*3.*	*4.*
ChatId	FK	uniqueidentifier	Ідентифікатор чату, до якого належить повідомлення (зовішній ключ на таблицю “Chats”)
CreatedAt	-	datetime2(7)	Дата створення чату
UpdatedAt	-	datetime2(7)	Дата редагування чату
Type	-	int	Тип повідомлення
IsEdited	-	bit	Чи повідомлення редагувалося
ReplyMessageId	FK	uniqueidentifier	Ідентифікатор повідомлення, на яке зробили відповідь (зовішній ключ на таблицю “Messages”)
FileUrl	-	varchar(450)	Посилання на файл повідомлення
ForwardedMessage	-	varchar(max)	Переслане повідомлення
GeeseTextArguments	-	varchar(450)	Аргументи тексту повідомлення
MediaKind	-	int	Медіа тип повідомлення
MimeType	-	varchar(50)	Mime-type повідомлення

Таблиця 2.6

Структура таблиці «ReadMessages»

Поле	Ключ	Тип	Призначення
1.	*2.*	*3.*	*4.*
MessageId	PK	uniqueidentifier	Ідентифікатор повідомлення (зовішній ключ на таблицю “Messages”)
ReadById	PK	uniqueidentifier	Ідентифікатор користувача (зовішній ключ на таблицю “Users”)
CreatedAt	-	datetime2(7)	Дата прочитання повідомлення

2.3 Проектування та реалізація алгоритмів роботи системи

Оскільки для програмного додатку обрано архітектуру SPA, веб-застосунок чітко розділяється на серверну та клієнтську частини, і передбачає використання RESTful API.

RESTful API - це інтерфейс програмування застосунків (API), який використовує архітектурний стиль REST (Representational State Transfer) для створення, оновлення, читання та видалення даних. Він базується на принципах взаємодії між клієнтом та сервером за допомогою стандартних HTTP-методів, таких як GET, POST, PUT та DELETE, і використовує ресурси (наприклад, URL) для ідентифікації та доступу до цих даних.

У випадку створення цього додатку відбувається наступне: клієнтська частина звертається до API для отримання необхідних даних. Використовуючи ASP.NET [3], на відповідний запит з клієнтської частини через бібліотеку Apollo Client [21] повертається JSON-об’єкт. На серверній стороні дані отримуються з бази даних і з ними виконуються необхідні маніпуляції. Потім ці дані використовуються для формування користувацького інтерфейсу. Якщо користувач змінює запит сторінки, здійснюється повторний запит до серверу веб-застосунку.

Зображено загальний алгоритм запитування даних додатку (рис. 2.5).

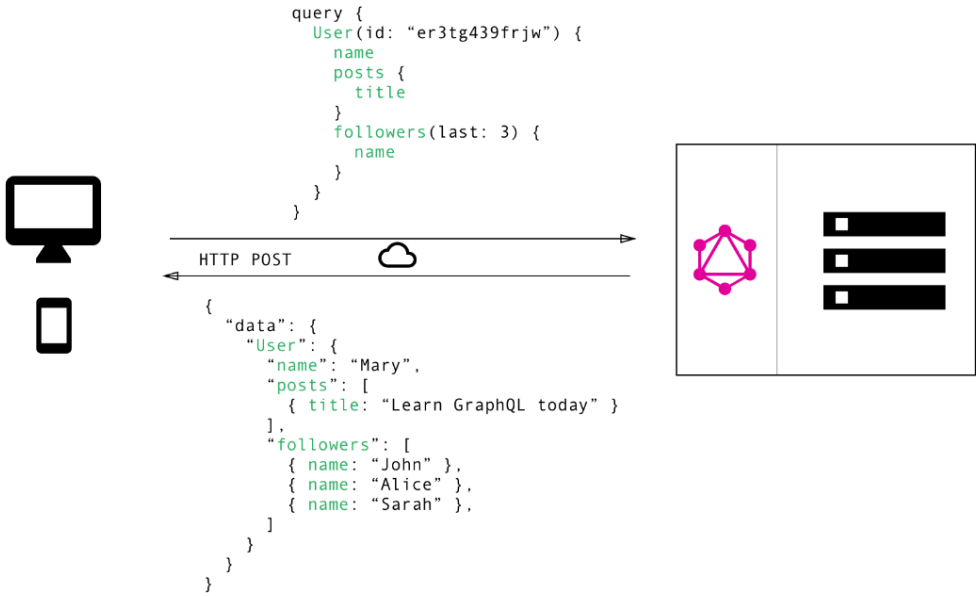

Рисунок 2.5 - Загальний алгоритм запитування даних додатку

Крім того, для ілюстрації алгоритму роботи додатку створимо діаграму активностей.

Зображено діаграму активностей (рис. 2.6).

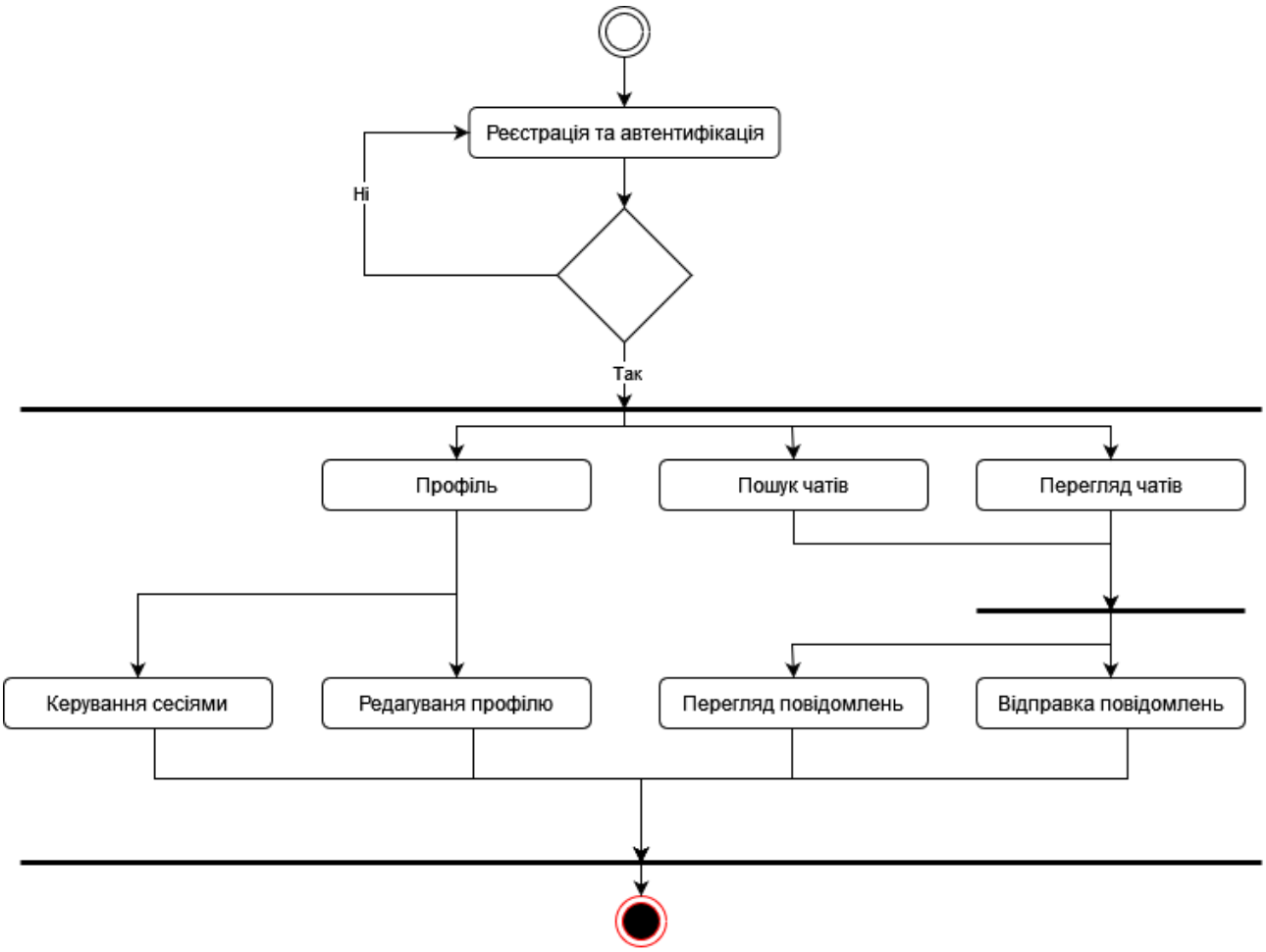

Рисунок 2.6 – Діаграма активностей системи

Опис отриманої діаграми:

1. Автентифікація (так – перехід до використання додатку, ні – перехід на сторінку входу).
2. Профіль
   - Редагування профілю
   - Керування сесіями
3. Пошук чатів, перегляд чатів:
   – перегляд повідомлень;
   – відправка повідомлень.
4. Кінець роботи веб-додатку.

Застосунок можна умовно розділити на кілька модулів. Для наочності створимо діаграму компонентів додатку.

Зображено діаграму компонентів додатку (рис. 2.7).

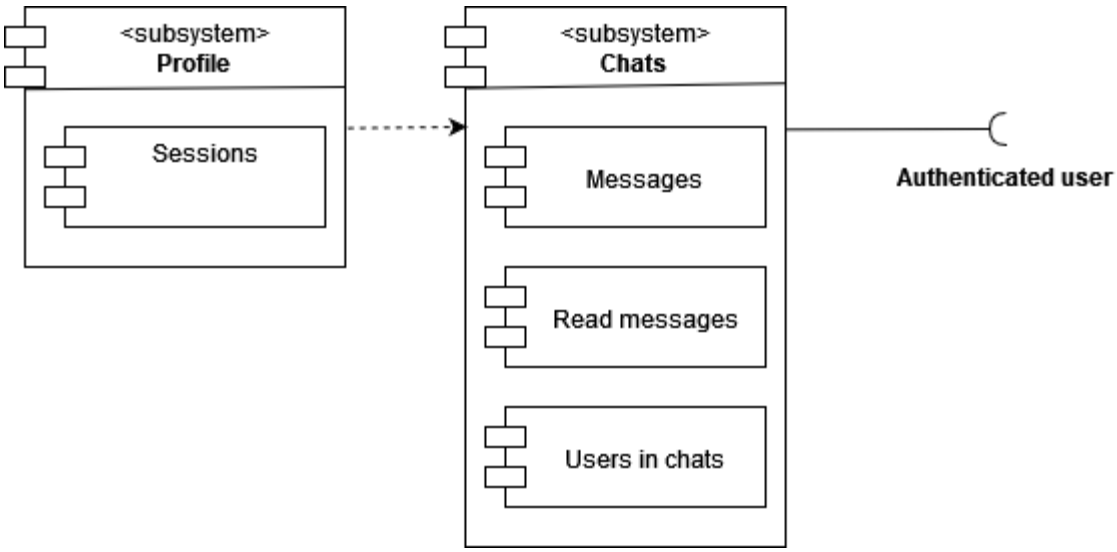

Рисунок 2.7 - Діаграма компонентів додатку

Проаналізувавши отриману діаграму, отримуємо наступне:

1. Для роботи з основними компонентами користувач повинен бути автентифікованим у додатку.
2. Систему було розділено на два підкомпоненти і один окремий компонент.
3. Підсистема Chats представляє собою керування чатами, повідомленнями, прочитаними повідомленнями та користувачами у чаті.
4. Підкомпонент Profile відповідає за профіль користувача, що включає у себе також сесії поточного користувача.

2.4 Реалізація веб-месенджеру

Початок роботи з додатком передбачає автентифікацію користувача. Розглянемо реалізацію даного функціоналу.

Лістинг GraphQL поля Login:

```
  Field<NonNullGraphType<AuthResponseType>, AuthResponse>()
      .Name("Login")
      .Argument<NonNullGraphType<AuthLoginInputType>, AuthLoginInput>("input",
"Argument to login User")
      .ResolveAsync(async context =>
      {
          var authLoginInput = context.GetArgument<AuthLoginInput>("input");
          authLoginInputValidator.ValidateAndThrow(authLoginInput);

          var user = await
userProvider.GetByIdentifierAsync(authLoginInput.Identifier);
```

```
            if (user == null)
                throw new Exception("Login or password not valid.");

            var saltedPassword = authLoginInput.Password + user.Id;
            if (user.Password != saltedPassword.CreateMD5())
                throw new Exception("Login or password not valid.");

            var session = new Session
            {
                Token = authService.GenerateAccessToken(user.Id, user.Identifier,
user.Role),
                UserId = user.Id,
            };
            session = await authService.FillSession(session, true);
            session = await sessionManager.CreateAsync(session);

            return new AuthResponse()
            {
                Token = session.Token,
                User = user,
            };
        });
```

Це поле передбачає визначення даних, які користувач вводить для входу в систему. Далі здійснюється спроба автентифікації на основі цих даних, генеруючи новий токен та логін сесію. Цей токен надалі використовуватиметься на клієнтській стороні для формування автентифікаційного заголовку. Якщо дані введено неправильно, користувачу повертається повідомлення про помилку.

Далі розглянемо оновлення профілю.

Лістинг GraphQL поля UpdateProfile:

```
 Field<NonNullGraphType<UserType>, User>()
     .Name("UpdateProfile")
     .Argument<NonNullGraphType<AuthUpdateProfileType>, AuthUpdateProfile>("Input",
"")
     .ResolveAsync(async context =>
     {
         var authUpdateProfile = context.GetArgument<AuthUpdateProfile>("Input");
         await authUpdateProfileValidator.ValidateAndThrowAsync(authUpdateProfile);

         string imageUrl;
         if (authUpdateProfile.Image != null)
             imageUrl = await
fileManagerService.UploadFileAsync(FileManagerConstants.UsersAvatarsFolder,
authUpdateProfile.Image);
         else
             imageUrl = authUpdateProfile.ImageUrl;

         var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();
         var currentUser = await userProvider.GetByIdAsync(currentUserId);
         currentUser.FirstName = authUpdateProfile.Firstname;
         currentUser.LastName = authUpdateProfile.Lastname;
         currentUser.Identifier = authUpdateProfile.Identifier;
         currentUser.ImageUrl = imageUrl;
         currentUser = await userProvider.UpdateAsync(currentUser);

         await chatActivitySubscriptionService.Notify(currentUser.Id);
```

```
            return currentUser;
        })
        .AuthorizeWith(AuthPolicies.Authenticated);
```

Поле UpdateProfle у відповідає за оновлення профілю користувача. Він перевіряє, чи користувач автентифікований, і приймає нові дані профілю, такі як ім'я, прізвище, аватар та ідентифікатор. Після валідації даних, відбувається оновлення інформації у базі даних. У разі успішного оновлення повертає підтвердження з новими даними профілю, а у випадку помилки - відповідне повідомлення про помилку.

Однією з важливих частин будь якого месенджеру це створення чату.

Лістинг GraphQL поля CreatePersonalChat:

```
        Field<NonNullGraphType<ChatType>, Chat>()
            .Name("CreatePersonal")
            .Argument<CreatePersonalChatInputType>("Input", "Chat input for creating new
chat.")
            .ResolveAsync(async context =>
            {
                var chatInp = context.GetArgument<CreatePersonalChatInput>("Input");
                var currentUserId =
httpContextAccessor.HttpContext.User.Claims.GetUserId();

                var checkChat = await
chatManager.GetByIdentifierAsync(chatInp.Identifier, currentUserId);
                if (checkChat != null)
                    throw new Exception("Personal chat already exists");

                var oppositeUser = await
userProvider.GetByIdentifierAsync(chatInp.Identifier);

                var chat = new Chat
                {
                    Type = oppositeUser.Id == currentUserId ? ChatKind.Saved :
ChatKind.Personal,
                    CreatorId = currentUserId
                };
                chat = await chatManager.CreateAsync(chat);

                var userChat = new List<UserChat>
                {
                    new UserChat { UserId = currentUserId, ChatId = chat.Id },
                };

                if (oppositeUser.Id != currentUserId)
                    userChat.Add(new UserChat { UserId = oppositeUser.Id, ChatId =
chat.Id });

                await userChatManager.CreateManyAsync(userChat);

                chat = await chat.MapForUserAsync(currentUserId, serviceProvider);
                chatActionSubscriptionService.Notify(chat, ChatActionKind.Add,
userChat.Select(uc => uc.UserId));
                return chat;
            })
            .AuthorizeWith(AuthPolicies.Authenticated);
```

Поле CreatePersonalChat створює особистий чат між двома користувачами. Він перевіряє, чи користувач автентифікований, і приймає дані, такі як ідентифікатор іншого користувача. Якщо чат між цими користувачами вже існує, повертає його дані, інакше створює новий чат у базі даних. Успішна відповідь містить інформацію про новий чат, а у разі помилки повертається відповідне повідомлення про помилку.

Ще однією з важливих частин будь якого месенджеру це відправка повідомлення. У «Додатку А» продемонстровано GraphQL поле SendMessage. Поле SendMessage перевіряє аутентифікацію користувача і приймає індентифікатор чату, текст повідомлення, масив файлів та переслані повідомлення. Якщо дані валідні, йде перевірка чи користувач належить до цього чату. Після чого зберігаються файли у файловий менеджер та відбувається зберігання повідомлення в базі даних, відправляє сповіщення отримувачам і повертає підтвердження з деталями повідомлення. У випадку помилок повертається відповідне повідомлення про помилку.

Висновки до другого розділу

У даному розділі було створено діаграму варіантів використання для зрозумілого відображення відношень між акторами та прецедентами в системі. Були визначені вимоги до системи, аргументовано та розглянуто застосування патернів при розробці додатку. Також було розроблено структуру бази даних та надано опис полів і їх призначення.

Було розглянуто використання архітектури односторінкового додатку (Single-Page Application) та обґрунтовано застосування підходу RESTful API. Загалом описано алгоритм роботи і створено наступні діаграми для більшої наочності: діаграма активностей системи, діаграма послідовності для процесу оплати та діаграма компонентів системи.

Також показано реалізацію програмного забезпечення, представлені фрагменти коду, які відповідають за процеси автентифікації, створення чату та відправлення повідомлень, з докладним поясненням їхнього принципу роботи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-МЕСЕНДЖЕРОМ

3.1 Порядок встановлення та налаштування параметрів системи

Для зображення розгортання системи існує однойменна діаграма – діаграма розгортання.

Діаграма розгортання - це тип UML-діаграми, який використовується для візуалізації архітектурної структури системи та опису розміщення її компонентів (програмного забезпечення, апаратного забезпечення, мережевих елементів тощо) на фізичних або віртуальних пристроях або серверах.

Зображено діаграму розгортання (рис. 3.1).

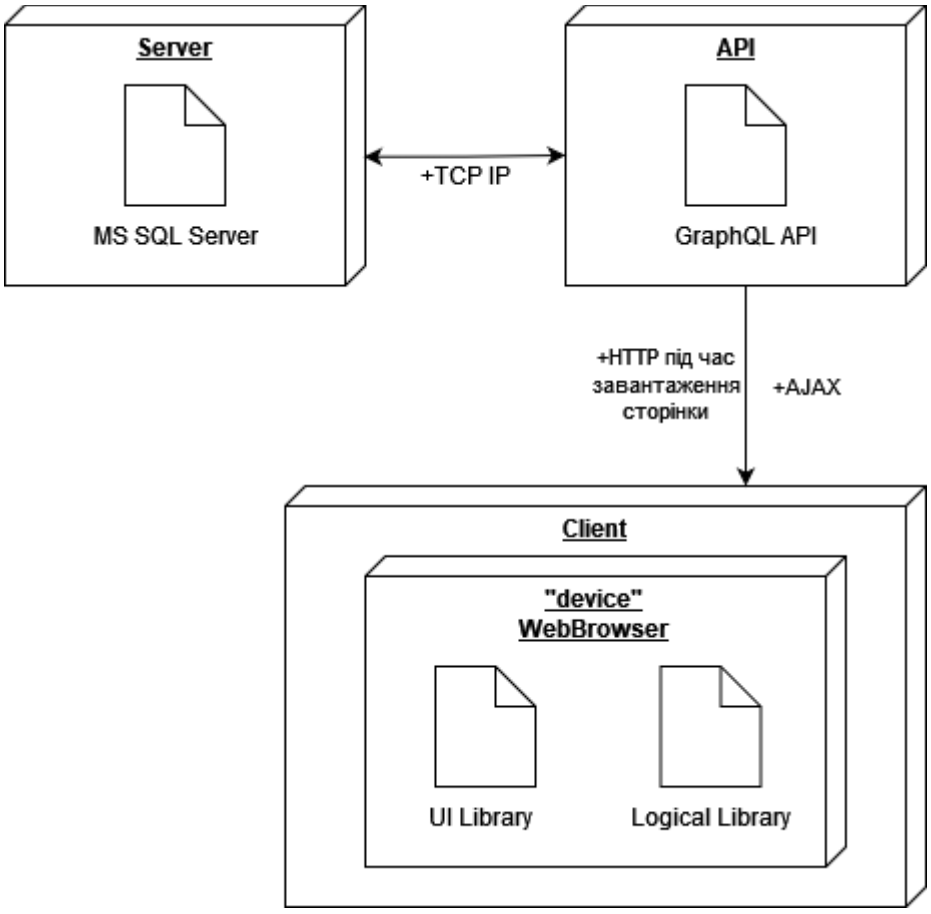

Рисунок 3.1 - Діаграма розгортання додатку

Після створення цієї діаграми можна зробити такі висновки щодо порядку встановлення та налаштування параметрів системи:

– бекенд частина звертається до сервера для отримання необхідних ресурсів;

– клієнт є додатком, що використовує бібліотеки для створення користувацького інтерфейсу та налаштування логіки оновлення даних і сторінок застосунку;

– під час запуску додатку клієнт отримує початкову HTML-розмітку, а потім оновлює її за допомогою асинхронних запитів до API.

3.2 Структура інтерфейсу та порядок взаємодії з веб-додатком

Після відкриття веб-застосунку користувача зустрічає сторінка входу. Зображено сторінку входу (рис. 3.2).

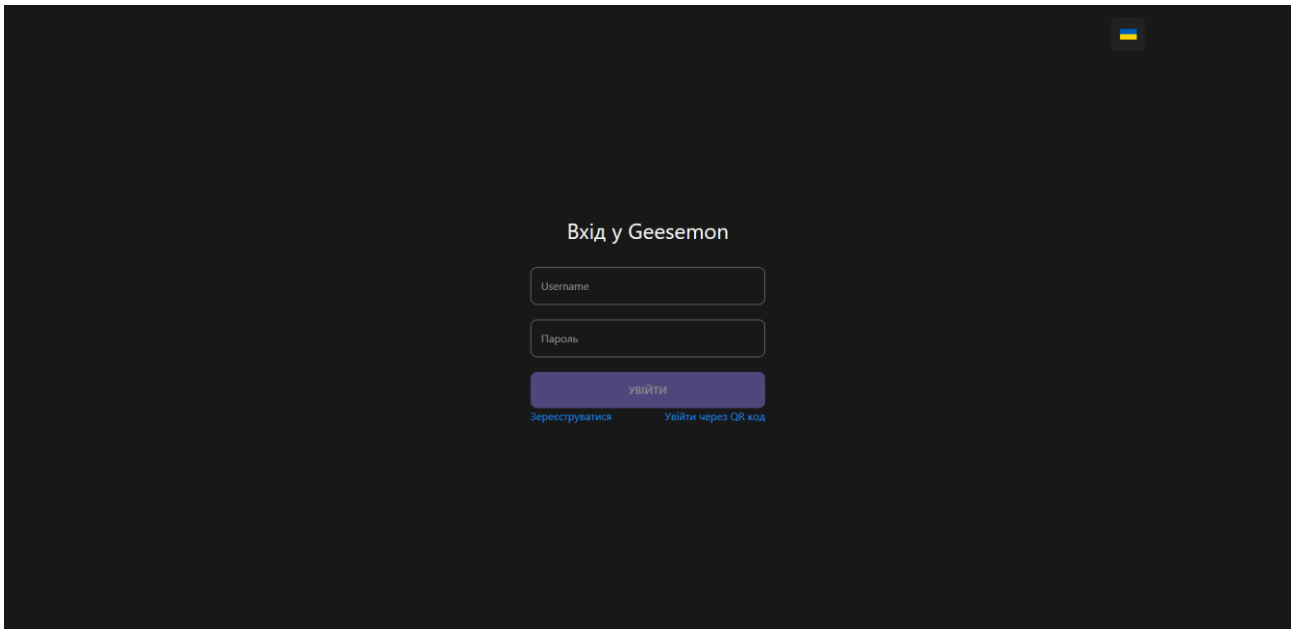

Рисунок 3.2 – Сторінку входу

Якщо користувач не має аккаунту, то він повинен перейти до сторінки реєстрації та зареєструватися, ввівши ім’я, прізвище, пошту, username та пароль.

Зображено сторінку реєстрації (рис. 3.3).

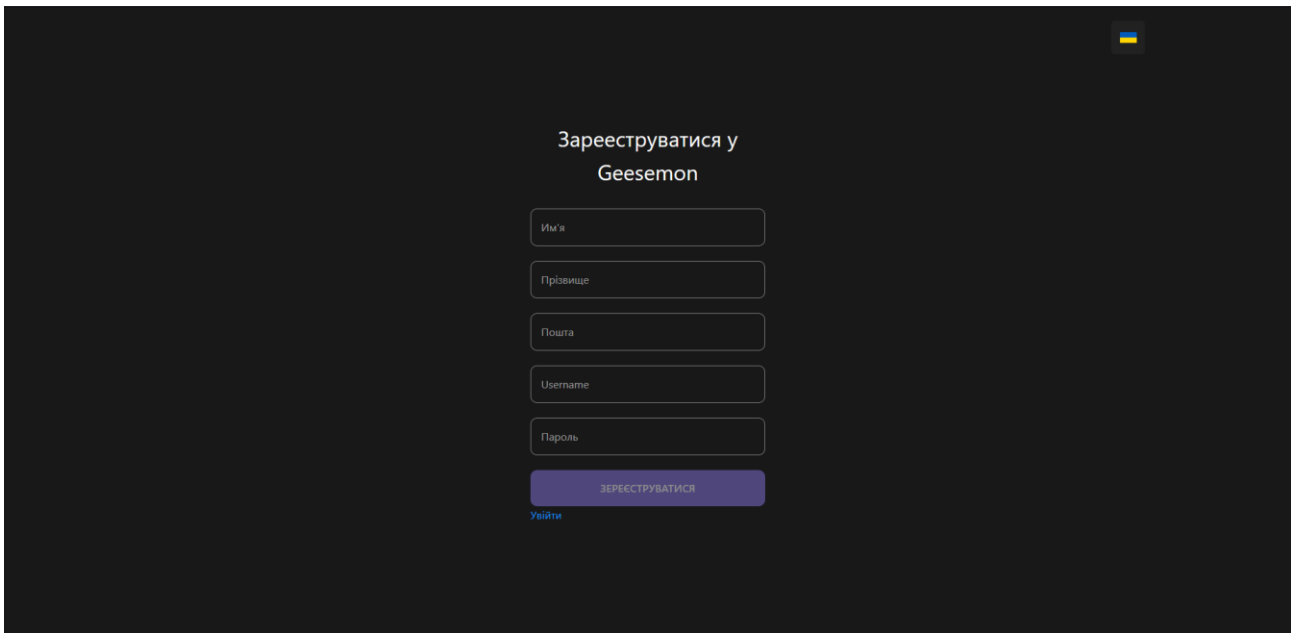

Рисунок 3.3 - Сторінка реєстрації

Після успішної рестрації користувач має можливість повноцінно користуватися додатком.

Зображено сторінку з переліком чатів (рис. 3.4).

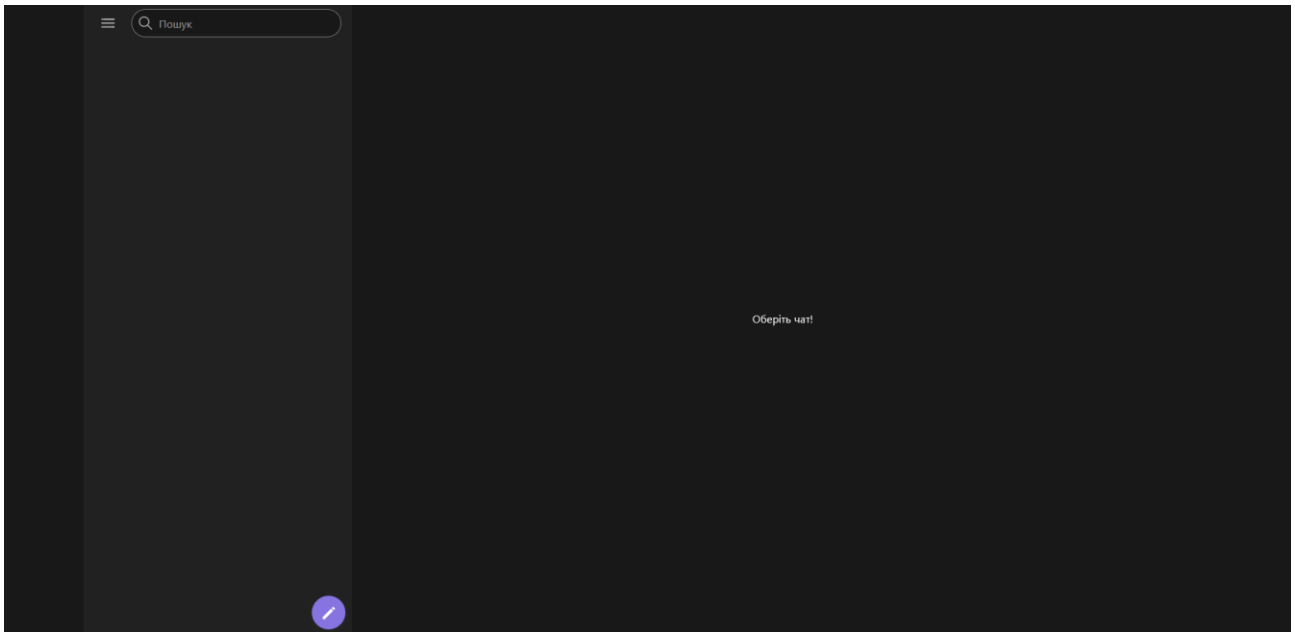

Рисунок 3.4 – Сторінка з переліком чатів

Так як користувач ще нікому не писав, то список чатів порожній і відповідно список повідомлень теж. Для створення чату потрібно натиснути кнопку «Написати» та вибрати пункт «Новий персональний чат» або «Нова група». Зображений даних процес (рис. 3.5).

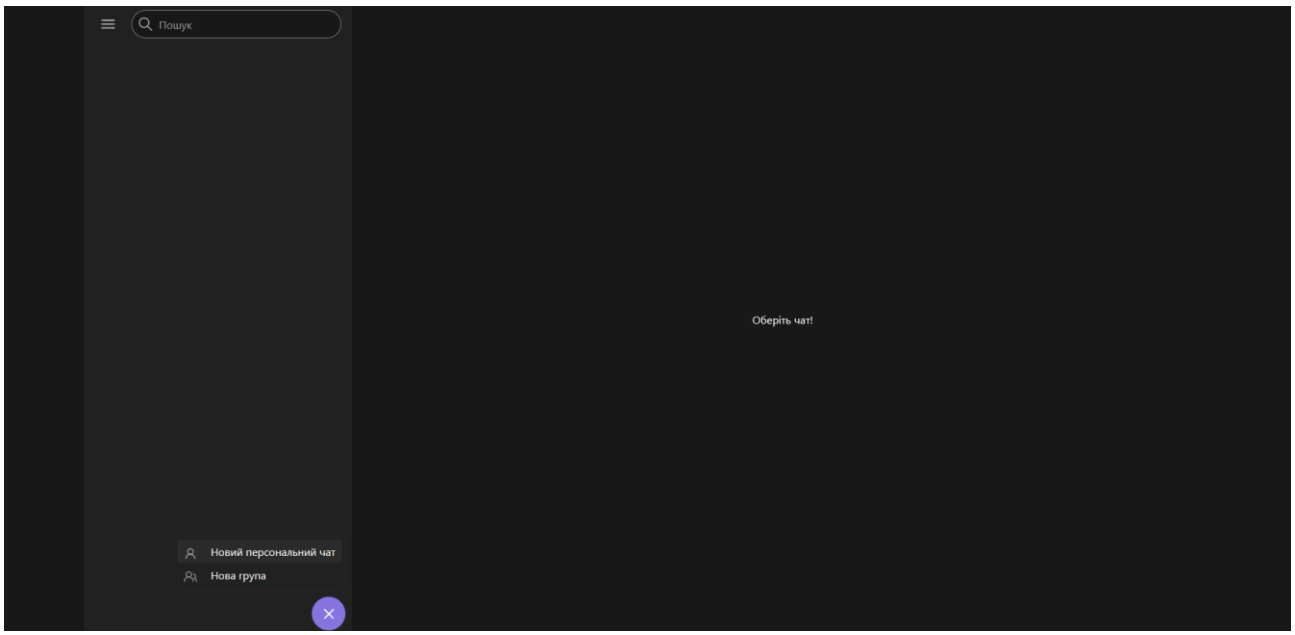

Рисунок 3.5 – Список опцій для кнопки «Написати»

Після натискання опції «Новий персональний чат» відображається список користувачів, яким ми можемо написати приватне повідомлення.

Зображено список користувачів для створення персонального чату (рис. 3.6).

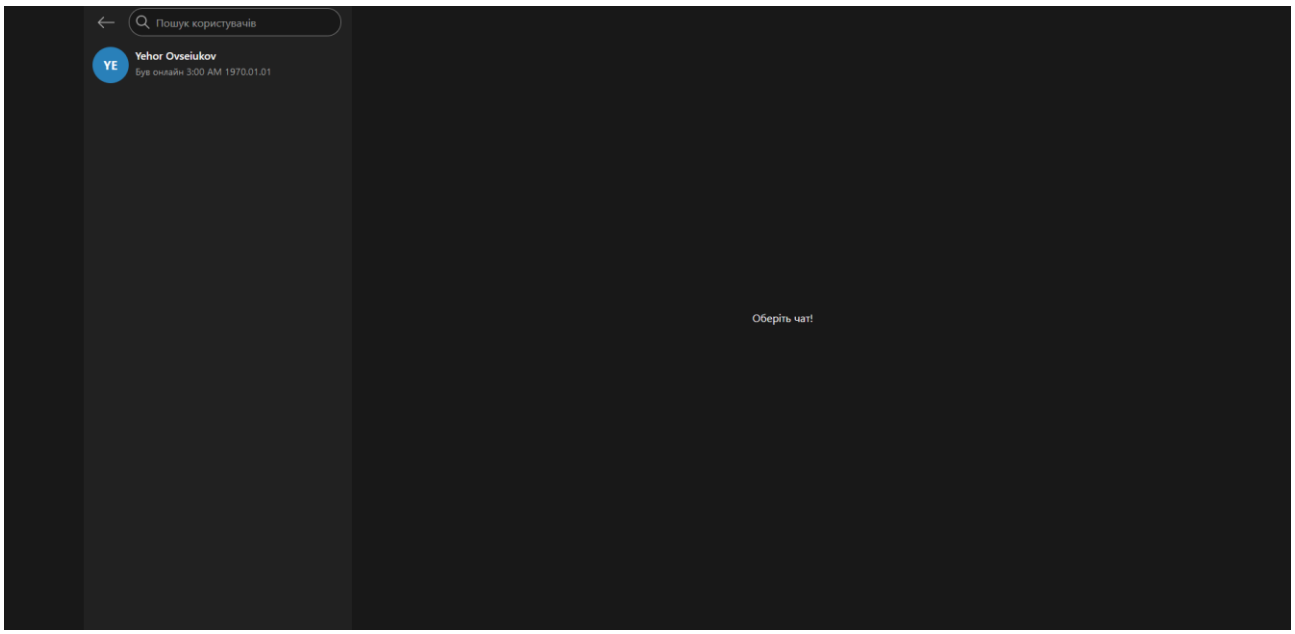

Рисунок 3.6 - Список користувачів для персонального чату

Після того, як користувач обере кому він хоче написати, буде створено тимчасовий чат, де можна буде написати повідомлення.

Зображено тимчасовий чат (рис. 3.7).

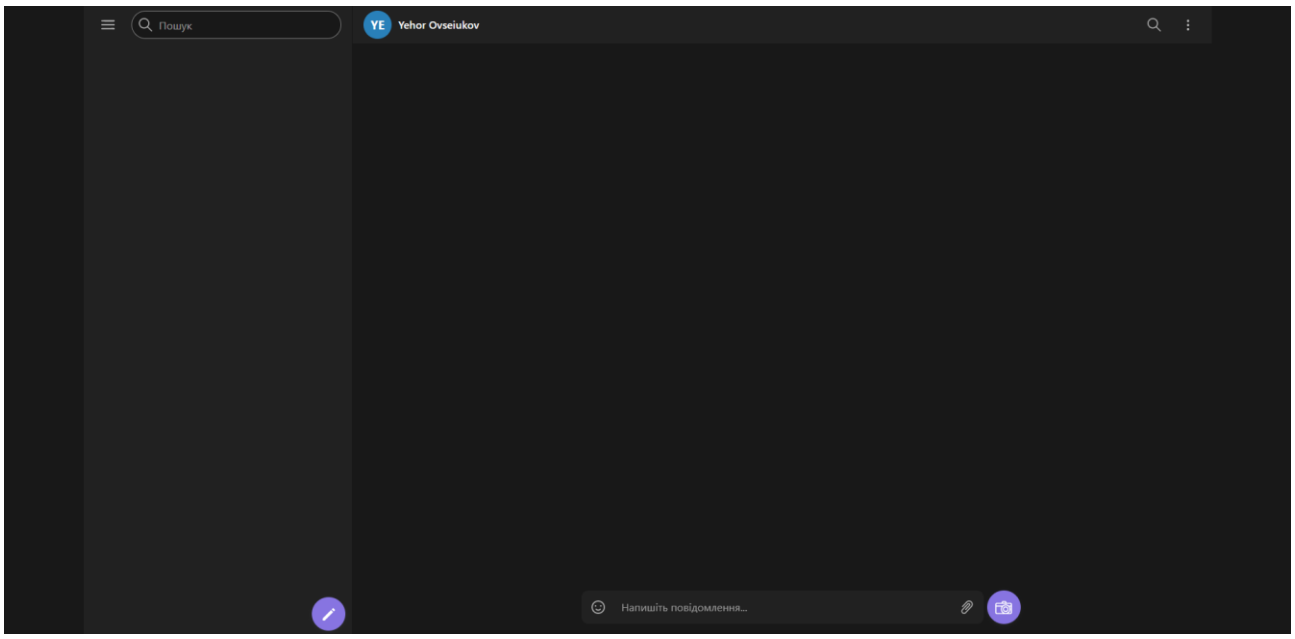

Рисунок 3.7 – Тимчасовий персональний чат

Після відправки першого повідомлення, аналогічний чат створиться для користувача, якому написали і він отримає нове повідомлення.

Зображено новий персональний створений чат і відправка першого повідомлення (рис. 3.8).

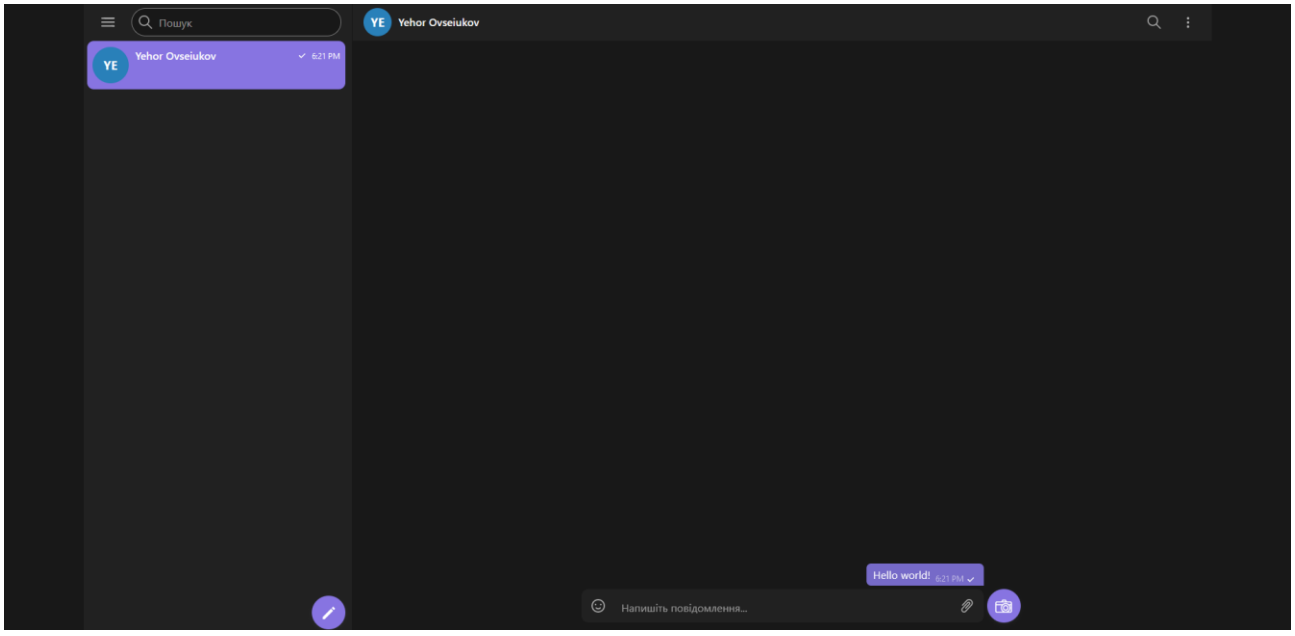

Рисунок 3.8 - Новий персональний створений чат і відправка першого повідомлення

Для створення групового чату потрібно обрати опцію «Нова група» на кнопці «Написати». Після чого буде відображено список користувачів, яких можна додати до чату.

Зображено список користувачів, яких можна додати до чату (рис. 3.9).

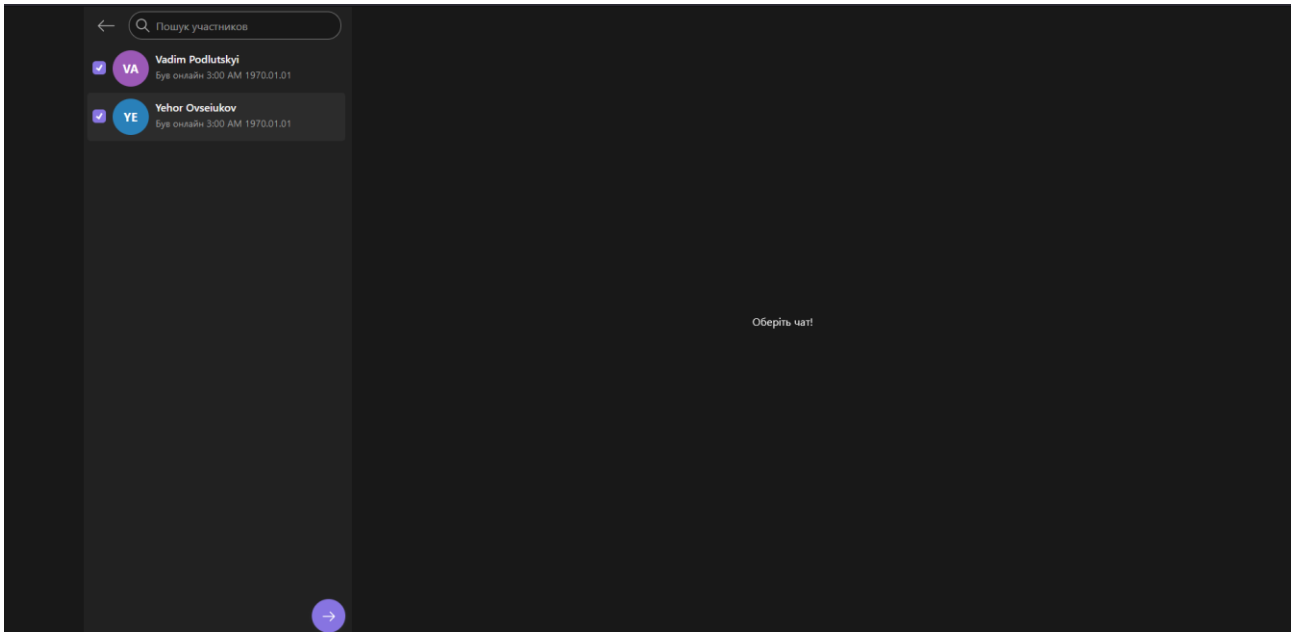

Рисунок 3.9 - Список користувачів, яких можна додати до чату

Після чого потрібно заповнити форму створення, ввівши назву, ідентифікатор та логотип групи.

Зображено форму створення чату (рис. 3.10).

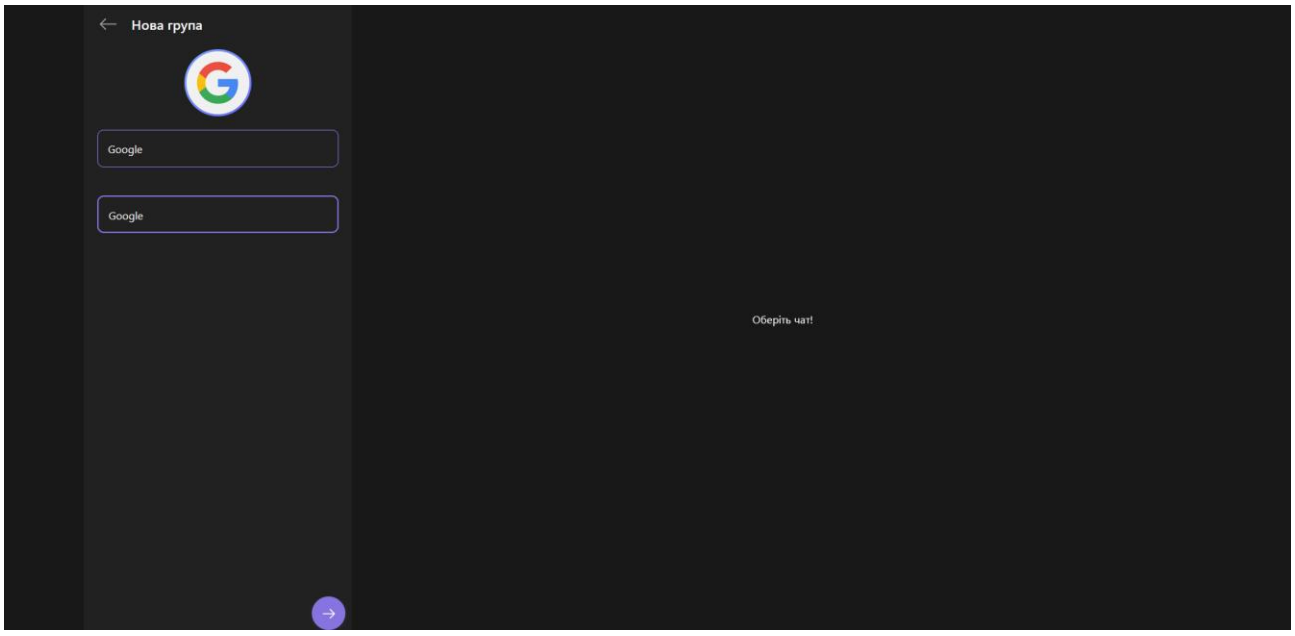

Рисунок 3.10 - Форма створення чату

Після натискання кнопки «Створити» нова група буде успішно створена для трьох учасників.

Зображено новий чат (рис. 3.11).

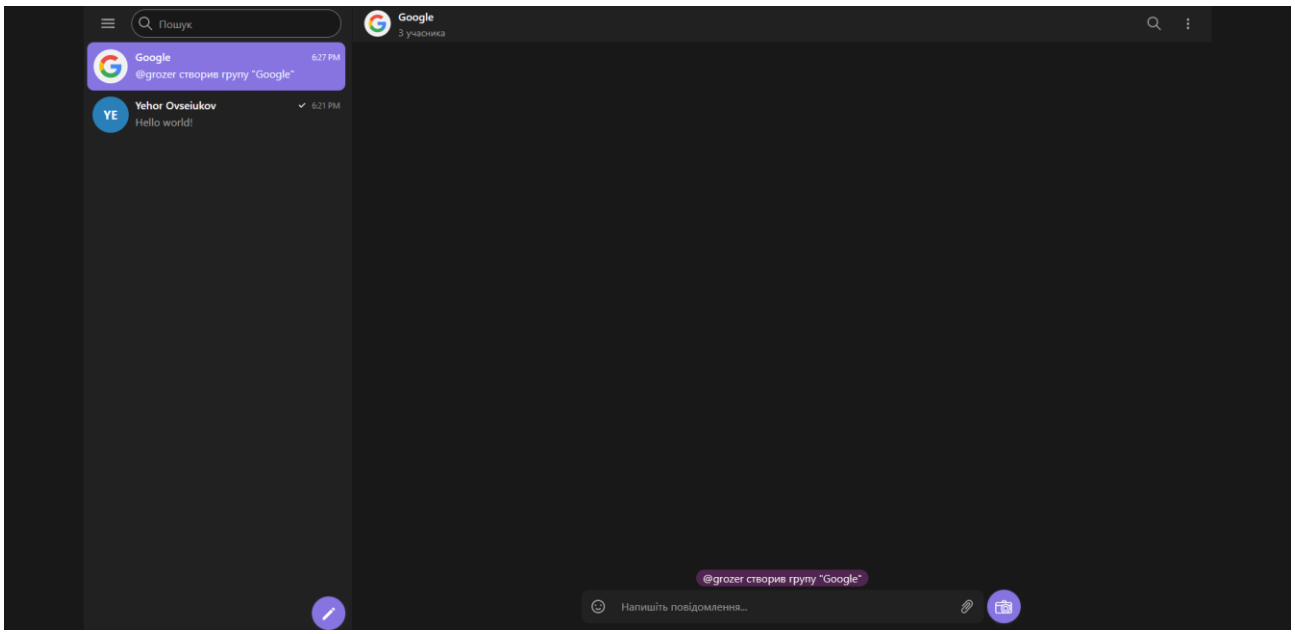

Рисунок 3.11 - Новий чат

Після відправки повідомлення всі учасники чату отримають нове повідомлення.

Далі розглянемо контекстне меню повідомлення, де є можливість відповісти, редагувати, копіювати, переслати, вибрати та видалити повідомлення. Також можна переглянути список користувачів, хто прочитав повідомлення.

Зображено контекстне меню повідомлення (рис. 3.12).

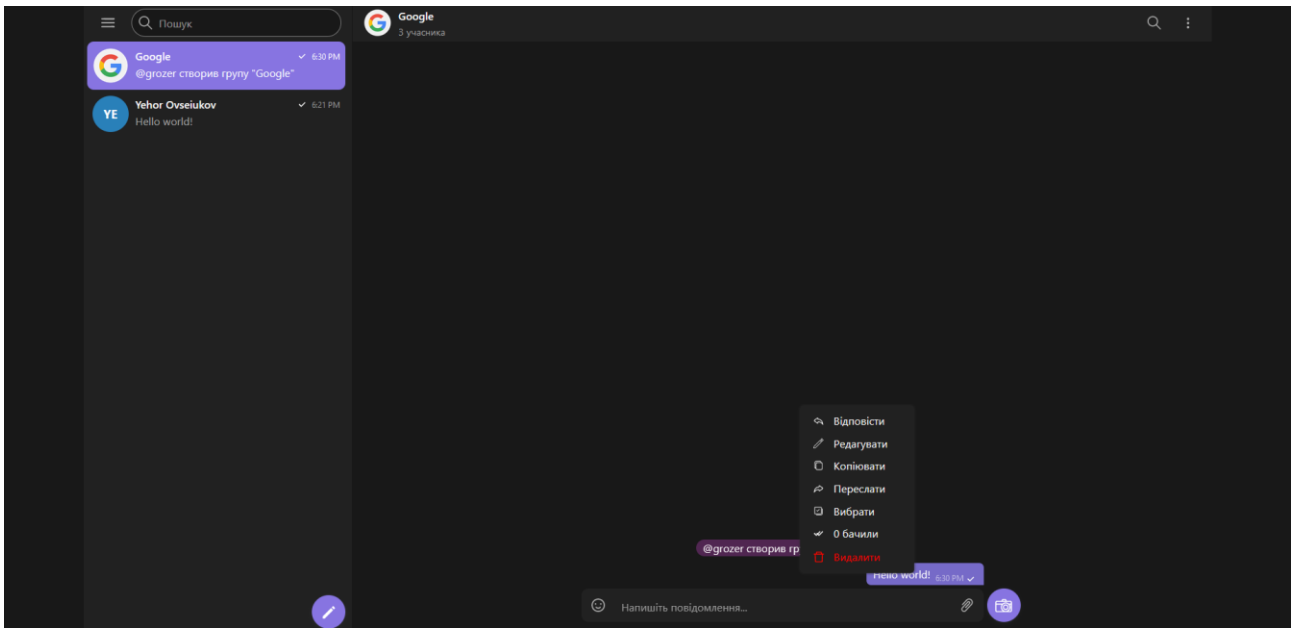

Рисунок 3.12 – Контекстне меню повідомлення

Опцією «Відповісти» користувач, буде мати можливість відповісти на своє чи від інших користувачів повідомлення.

Зображено процес «Відповісти» на повідомлення (рис. 3.13-3.14).

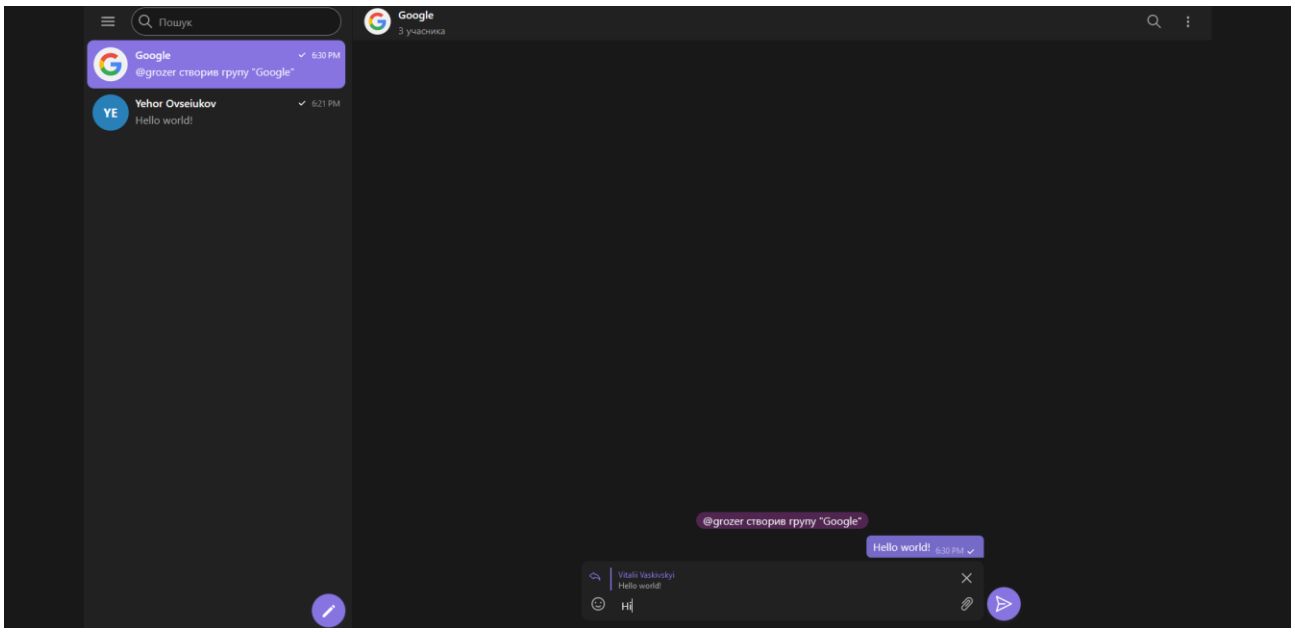

Рисунок 3.13 – Відповідь на повідомлення

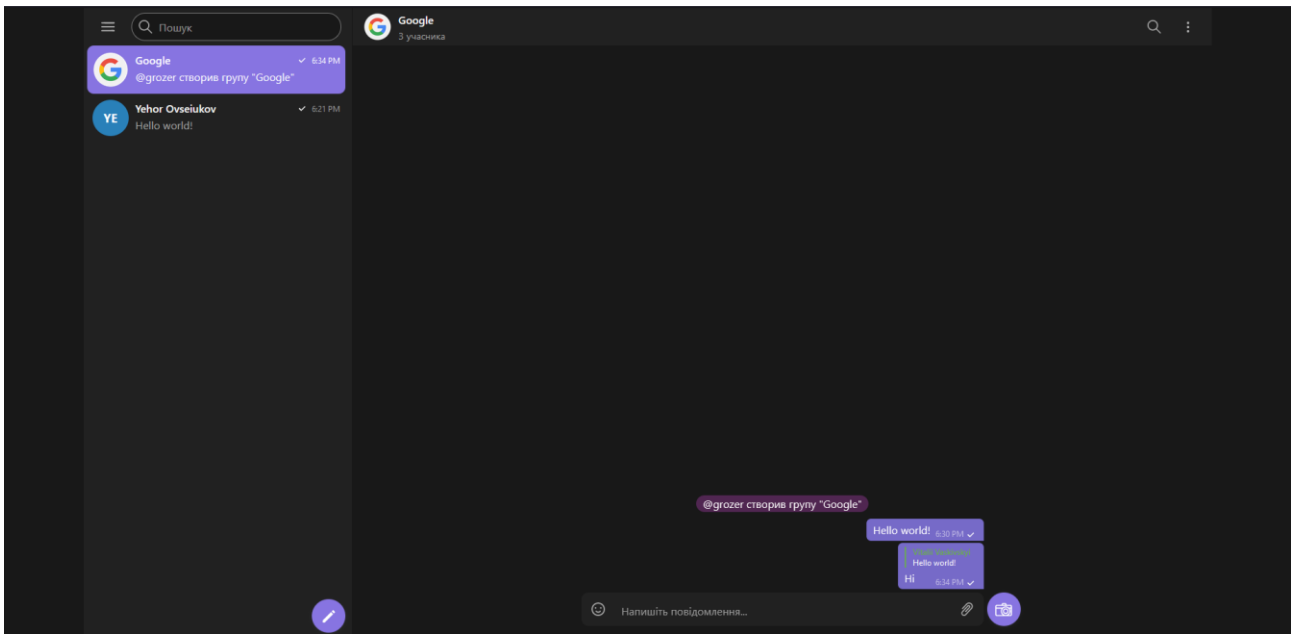

Рисунок 3.14 – Відправлене повідомлення, яке є відповідю на інше повідомлення

Для відправки файлів, користувач може написнути кнопку прикріпити файл, яка знаходиться справа від форми з відправкою повідомлення, після чого обрати потрібні файли і відправити.

Зображено відправку файлу (рис. 3.15).

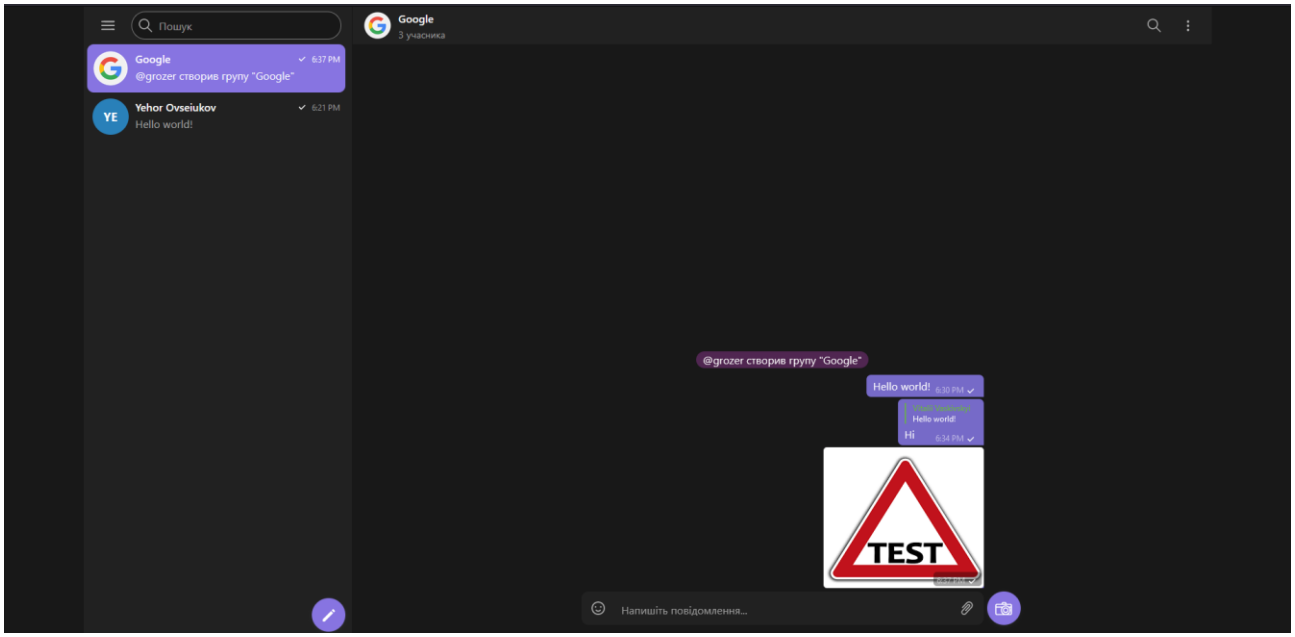

Рисунок 3.15 – Відправка файлу

Для відправки відеоповідомлень з камери користувач може натиснути кнопку з камерою. Після чого з’явиться модальне вікно де можна буде переглядати, що записує камера.

Зображено модальне вінко записування відео-повідомлення (рис. 3.16).

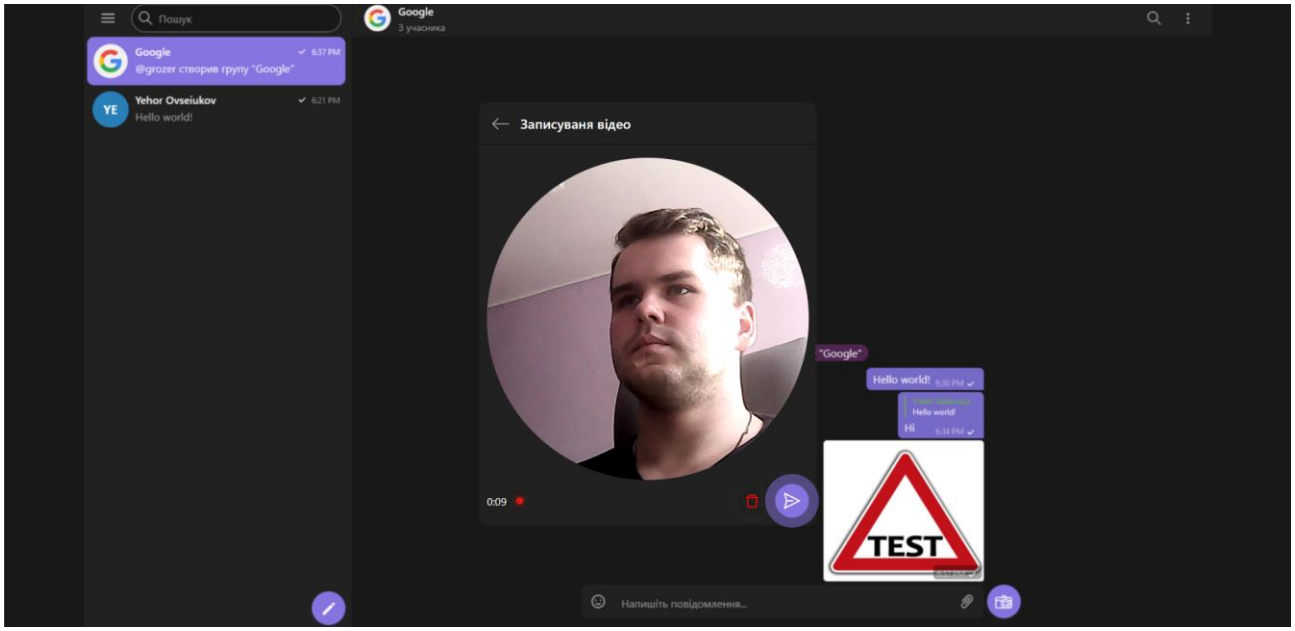

Рисунок 3.16 - Модальне вінко записування відео-повідомлення

Після записання необхідного повідомлення потрібно написнути «Надіслати» і повідомлення буде надіслано всім користувачам чату. Після чого всі користувачі, будуть мати можливість його переглянути.

Зображено перегляд надісланого відео-повідомлення (рис. 3.17).

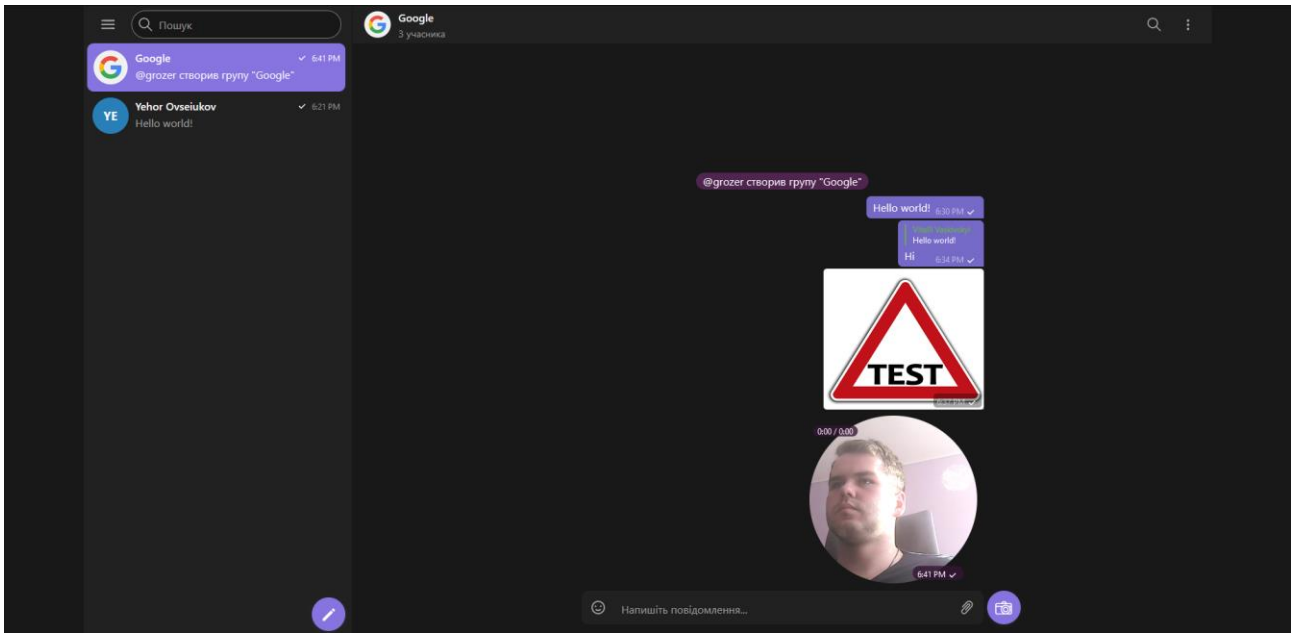

Рисунок 3.17 - Перегляд надісланого відео-повідомлення

Щоб відправити голосове повідомлення, користувач має натиснути правою кнопкою миші по кнопці з камерою, щоб перемкнути на режим відправки голосових повідомлень. Після натиснення кнопки з мікрофоном, почнеться запис звуку.

Зображено запис голосового повідомлення (рис. 3.18).

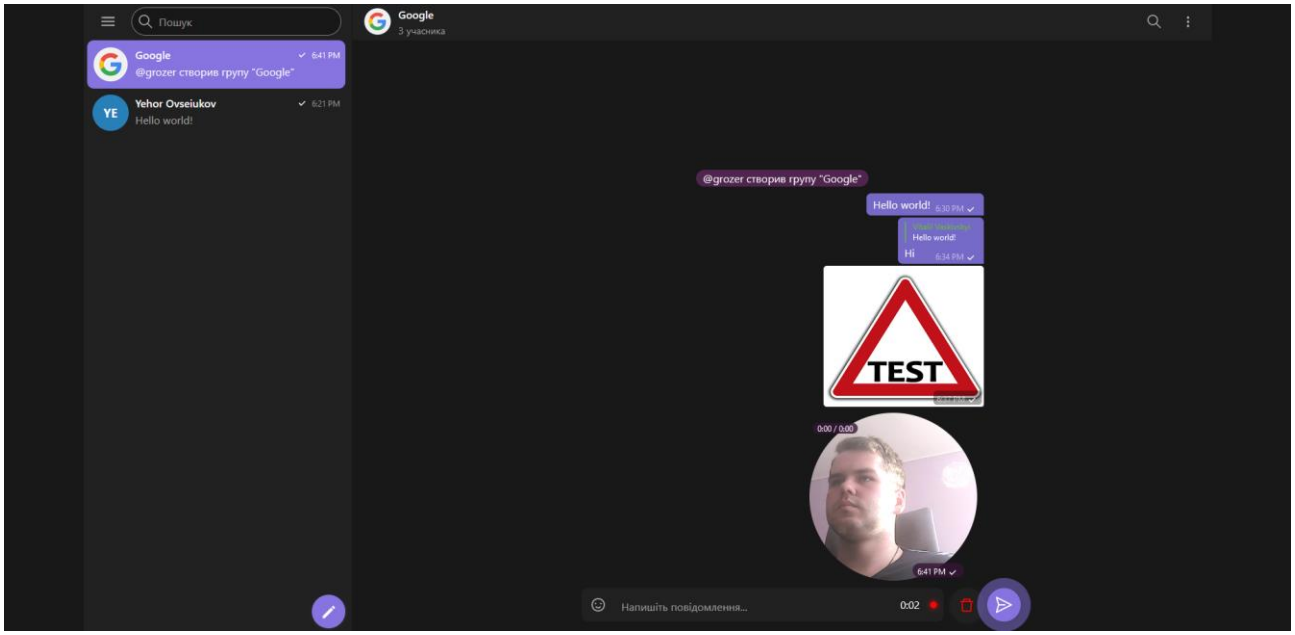

Рисунок 3.18 - Запис голосового повідомлення

Також користувач має можливіть переглядати свій профіль натиснувши «Налаштування» у головному меню.

Зображено налаштування користувача (рис. 3.19).

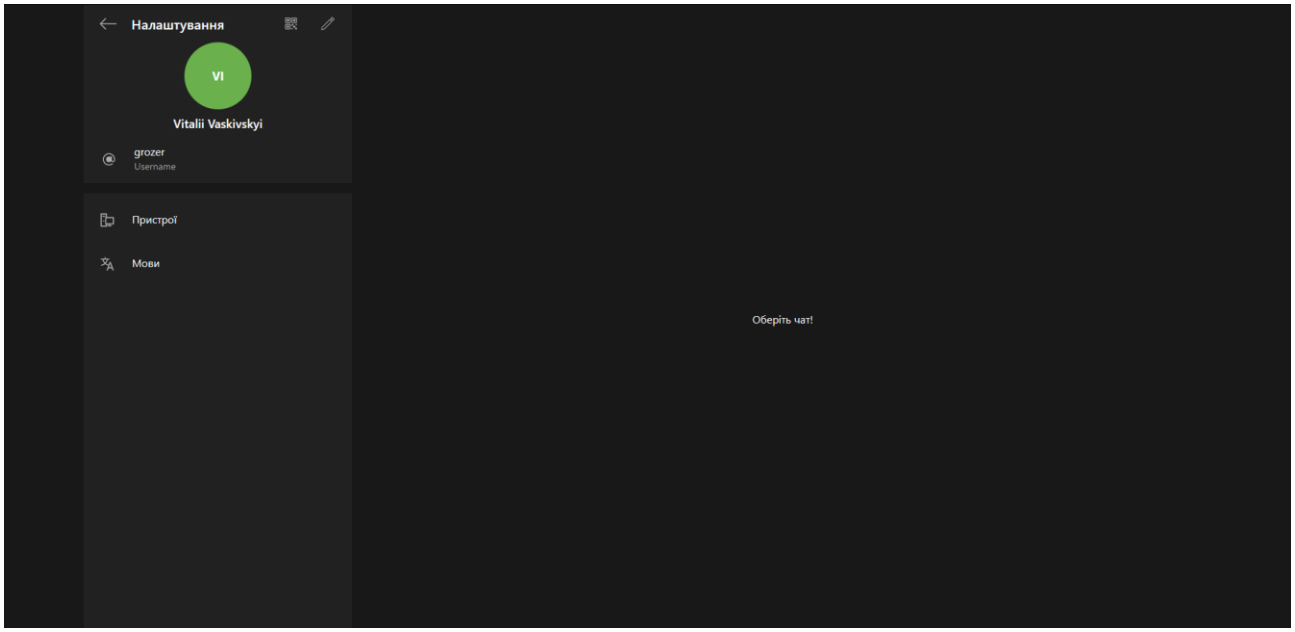

Рисунок 3.19 - Налаштування користувача

Щоб редагування профіль користувача, потрібно натиснути кнопку «Редагувати», після чого з’явиться форма редагування профілю, де можна оновити ім’я, прізвище, username та логотип користувача.

Зображено форма редагування профілю (рис. 3.20).

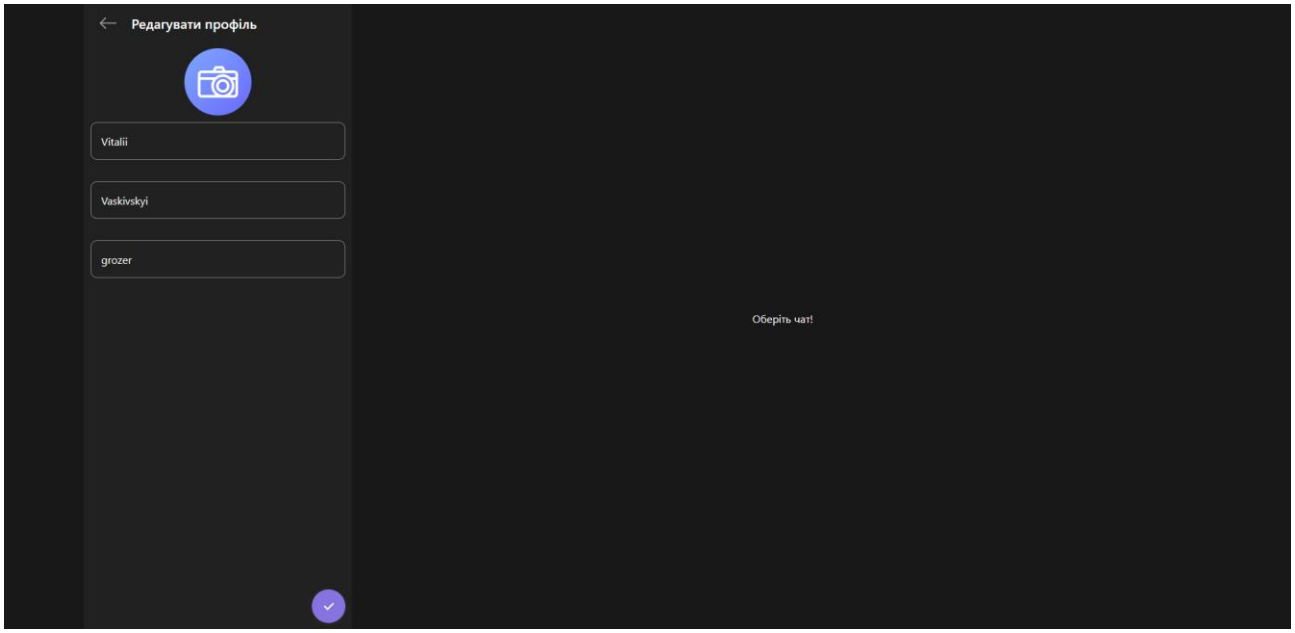

Рисунок 3.20 - Форма редагування профілю

Щоб перегляну список активних пристроїв, які використовувалися для входу у поточному аккауті, користувач має натиснути кнопку «Пристрої». На даній сторінці є можливіть видалити всі сесії, окрім поточної, або ж тільки бажані сесії, які необхідно видалити.

Зображено список пристроїв поточного користувача (рис. 3.21).

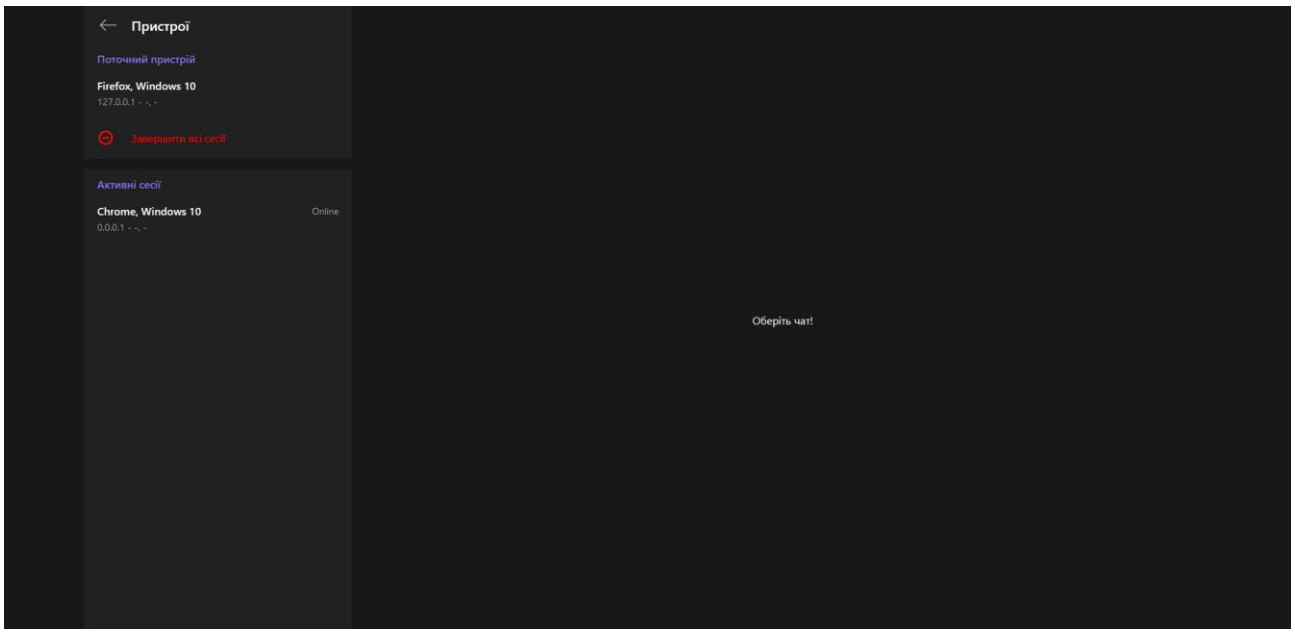

Рисунок 3.21 - Список пристроїв поточного користувача

3.3 Тестування роботи програмного продукту

Створення будь-якого програмного продукту завжди супроводжується можливістю помилок у процесі розробки, тому без тестування програмного забезпечення неможливо досягти високої якості додатку. У ході тестування перевіряються різні аспекти програми, такі як коректність вхідних та вихідних даних, правильність роботи функцій та алгоритмів, обробка помилок і відновлення після збоїв. Метою тестування є виявлення помилок і дефектів програмного продукту, що дозволяє їх виправити до випуску на ринок або введення в експлуатацію.

Спочатку створимо так звані Test Cases. Test Case – це документована процедура або сценарій, який використовується для виконання тестування програмного продукту. Кожен тест-кейс містить вхідні дані, дії, що повинні бути виконані, та очікуваний результат. Вони використовуються для перевірки правильності роботи програмного забезпечення та визначення відхилень від очікуваного результату.

Проаналізувавши таблицю, можемо зробити висновок, що всі описані вище тестові випадки відповідають поставленим очікуванням (табл. 3.1).

Таблиця 3.1

Тестові випадки

Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass /Fail
1.	*2.*	*3.*	*4.*	*5.*	*6.*
Перевірка автентифікації з дійсними даними	1. Перейти на сторінку https://localhost:7195/auth/login 2. Введіть username 3. Введіть пароль 4. Натисніть "Увійти"	Username = grozer Пароль = 12345	Користувач повинен увійти у додаток	Як і очікувалося	Pass
Перевірка автентифікації з недійсними даними	1. Перейти на сторінку https://localhost:7195/auth/login 2. Введіть username 3. Введіть пароль 4. Натисніть "Увійти"	Username = grozer Пароль = 123	Користувач не повинен увійти у додаток	Як і очікувалося	Pass
Перевірка реєстрації з дійсними даними	1. Перейти на сторінку https://localhost:7195/auth/register 2. Введіть ім’я 3. Введіть прізвище 4. Введіть пошту 5. Введіть username 6. Введіть пароль 6. Натисніть "Зареєструватися"	Ім’я = Yehor Прізвище = Ovseiukov Пошта = yehor@gmail.com Username = yehor Пароль = 12345	Повинен створитися акаунт і користувач повинен увійти у додаток	Як і очікувалося	Pass
Перевірка реєстрації з зареєстрованим usename	1. Перейти на сторінку https://localhost:7195/auth/register 2. Введіть ім’я 3. Введіть прізвище 4. Введіть пошту 5. Введіть username 6. Введіть пароль 6. Натисніть "Зареєструватися"	Ім’я = Yehor Прізвище = Ovseiukov Пошта = yehor@gmail.com Username = yehor Пароль = 12345	Користувач повинен отримати повідомлення про помилку, що аккаут з таким username уже існує	Як і очікувалося	Pass
Перевірка надсилання повідомлення всім учасникам чату	1. Увійти у додаток 2. Перейти у чат 3. Написати повідомлення 4. Натиснути «Надіслати»	Текст повідомлення = Hello world!	Повідомлення повинно відправитися усім учасникам чату	Як і очікувалося	Pass
Перевірка редагування повідомлення і відображення всім учасникам чату	1. Увійти у додаток 2. Перейти у чат 3. Натиснути ПКМ по повідомленню 4. Написати опцію «Редагувати» 5. Відредагувати тектс повідомлення 4. Натиснути «Надіслати»	Новий текст повідомлення = Hello to the whole world!	Повідомлення повинно відредагуватися для усіх учасників чату	Як і очікувалося	Pass

Також, необхідно перевірити обмеження деякого функціоналу для користувачів. Для цього використаємо API-платформу Altair. Вона дозволяє розробникам тестувати, створювати та розробляти GraphQL API [5].

Зображено спробу надіслати повідомлення до чату, до якого користувач не належить (рис. 3.22).

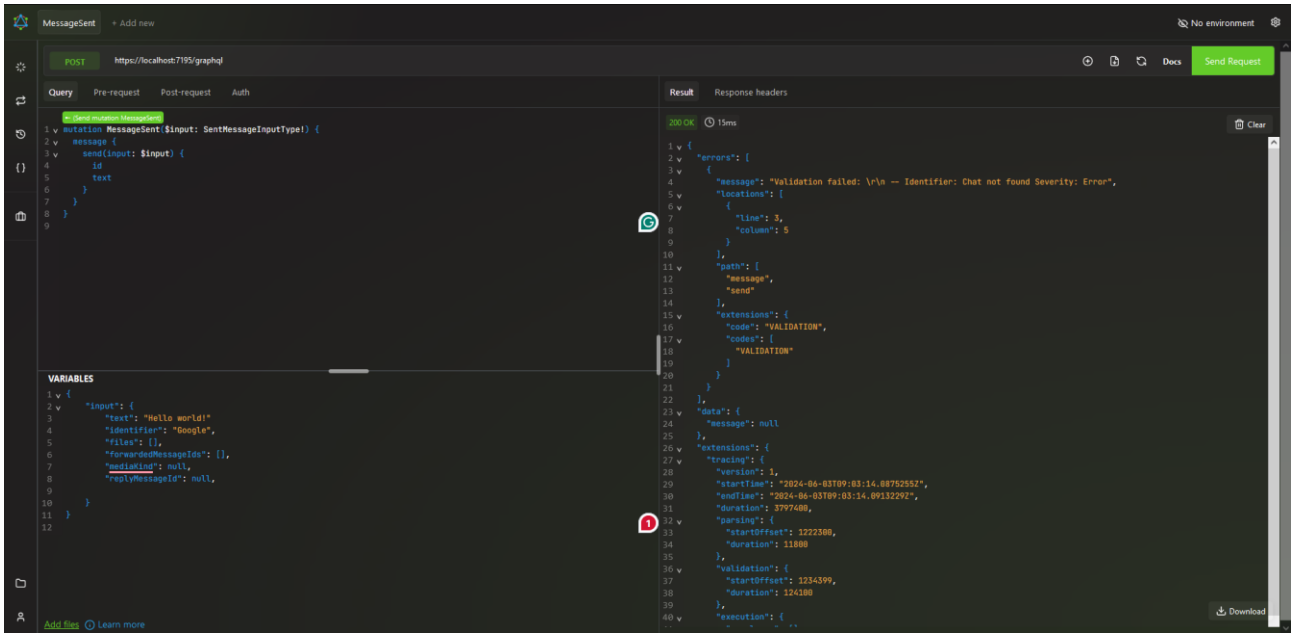

Рисунок 3.22 - Спроба надіслати повідомлення до якого користувач не належить

Виконання цієї дії не вдалося так, як очікувалося, і було отримано відповідне повідомлення.

Перевіримо спробу відредагувати повідомлення іншого користувача.

Зображено спробу редагування повідомлення іншого користувача (рис. 3.23).

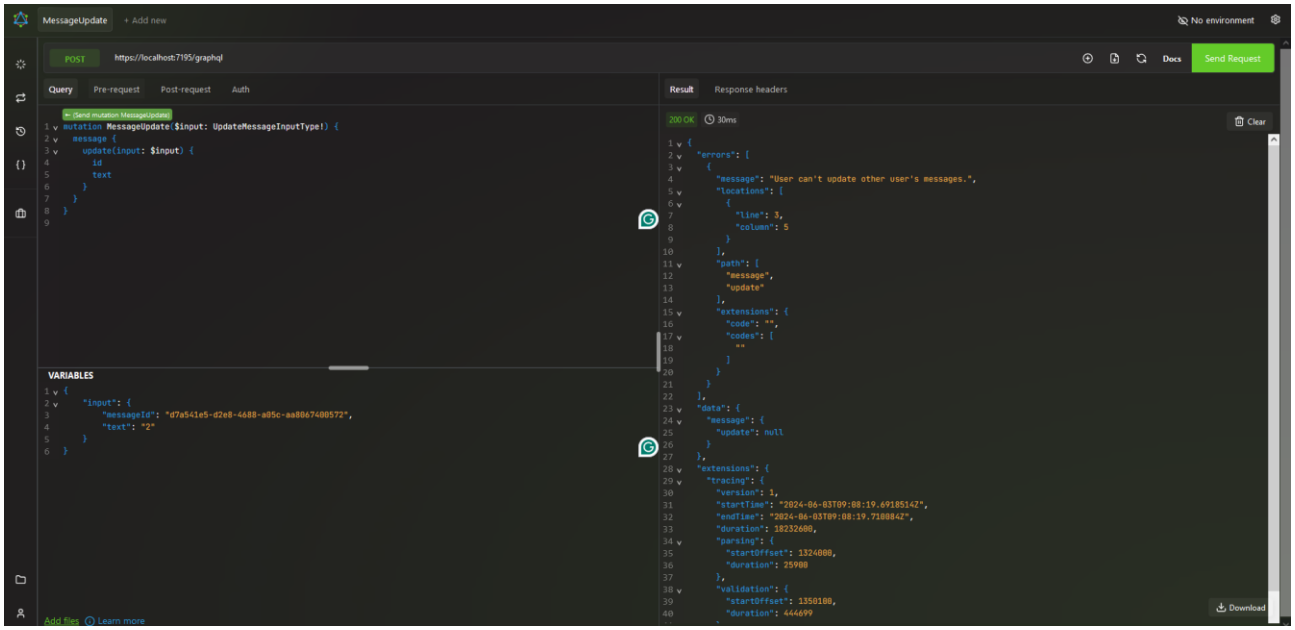

Рисунок 3.23 - Спроба редагування повідомлення іншого користувача

Виконання цієї дії не вдалося так, як очікувалося, і було отримано відповідне повідомлення.

Під час тестування було оглянуто програму та усунуто виявлені помилки, що призвело до створення додатку високої якості.

Висновки до третього розділу

У цьому розділі була побудована діаграма розгортання і отримані дані щодо порядку встановлення та налаштування параметрів системи.

Також була продемонстрована структура інтерфейсу і порядок взаємодії з веб-додатком. Було детально описано функціональні можливості додатку.

Проведено тестування веб-застосунку за допомогою Test Cases та API-платформи Altair.

Під час тестування були перевірені додаток і виправлені знайдені помилки, в результаті чого отримано застосунок належної якості.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано додаток веб-месенджер. Виконання поставленої задачі було розділено на 3 розділи.

У першому розділі було сформульовано постановку завдання для кваліфікаційної роботи. Описано функціональні вимоги та визначено основні етапи виконання роботи.

Був проведений аналіз аналогів продукту, який розробляється, що дозволив визначити наступні критерії розробки: інтуїтивний та зрозумілий дизайн; висока безпека; широка інтеграція; висока стабільність та швидкість; ефективне використання ресурсів.

Після аналізу наявних архітектур було вирішено використовувати підхід SPA для створення додатка, щоб забезпечити користувачам кращий користувацький досвід.

Крім того, було доведено обґрунтування вибору наступного набору технологій для створення додатка: використання ASP.NET [3] та GraphQL.NET [6] для розробки бекенду, React [18] для фронтенду, і Microsoft SQL Server [4] як системи керування базами даних.

У другому розділі була створена діаграма варіантів використання для візуального представлення взаємозв'язків між акторами та прецедентами у системі. Також були визначені вимоги до системи, розглянуто та описано використання патернів у процесі розробки додатку. Крім того, було проведено проектування бази даних та надано опис полів та їх призначення.

Було описано, як використовувалася архітектура Single-Page Application та обґрунтовано використання підходу RESTful API. Продемонстровано загальний алгоритм роботи і побудовані наступні діаграми для наочності: діаграма активностей системи та діаграма компонентів системи.

Реалізація додатку була показана, представлені фрагменти коду, що відповідають за процеси автентифікації, оновлення профілю, створення чату та відправлення повідомлень, з детальним поясненням їхньої роботи.

У третьому розділі була побудована діаграма розгортання і отримані дані про порядок встановлення та налаштування параметрів застосунку.

Крім того, було показано структуру інтерфейсу та процес взаємодії з веб-додатком. Детально описано функціональні можливості додатку.

Отриманий веб-застосунок було протестовано за допомогою тестових сценаріїв та API-платформи Altair.

Під час тестування було перевірено додаток, і знайдені помилки були виправлені. Як результат, отримано застосунок високої якості.

Розроблений програмний продукт готовий до експлуатації.. Веб-месенджер може бути використаний для комунікації, колаборації, організації робочих процесів, надання підтримки клієнтам, навчання та консультацій, а також для розваг та спілкування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Васьківський В. Ю., Овсеюков Є. Ю., Окунькова О. О., Розробка ВЕБ-меседжера. Тези V Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення» 01-02 грудня 2022 року. Житомир: «Житомирська політехніка», 2022. С.10-11 [Електронний ресурс]. Режим доступу до ресурсу: https://conf.ztu.edu.ua/wp-content/uploads/2023/02/10.pdf.
2. Документація C# [Електронний ресурс]. Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/dotnet/csharp.
3. Документація ASP.NET [Електронний ресурс]. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0.
4. Документація MS SQL Server [Електронний ресурс]. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16.
5. Документація GraphQL [Електронний ресурс]. Режим доступу до ресурсу: https://graphql.org/learn.
6. Документація GraphQL.NET [Електронний ресурс]. Режим доступу до ресурсу: https://graphql-dotnet.github.io.
7. Документація WebSockets [Електронний ресурс]. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
8. Документація Entity Framework [Електронний ресурс]. Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/ef.
9. Документація Cloudinary [Електронний ресурс]. Режим доступу до ресурсу: https://cloudinary.com/documentation/dotnet_integration.
10. Документація Fluent Validation [Електронний ресурс]. Режим доступу до ресурсу: https://docs.fluentvalidation.net/en/latest.
11. Документація Azure Blob Storage [Електронний ресурс]. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-dotnet.

12. Документація Dapper [Електронний ресурс]. Режим доступу до ресурсу: https://www.learndapper.com.
13. Jeffrey Richter. CLR via C#, 2012. - 894 с.
14. Документація JavaScript [Електронний ресурс]. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
15. Документація TypeScript [Електронний ресурс]. Режим доступу до ресурсу: https://www.typescriptlang.org/docs.
16. Документація WebPack [Електронний ресурс]. Режим доступу до ресурсу: https://webpack.js.org.
17. Документація Create React App [Електронний ресурс]. Режим доступу до ресурсу: https://create-react-app.dev/docs/getting-started.
18. Документація React [Електронний ресурс]. Режим доступу до ресурсу: https://reactjs.org.
19. Документація React Router Dom [Електронний ресурс]. Режим доступу до ресурсу: https://reactrouter.com/en/main.
20. Документація Redux [Електронний ресурс]. Режим доступу до ресурсу: https://redux.js.org.
21. Документація Apollo Client [Електронний ресурс]. Режим доступу до ресурсу: https://www.apollographql.com/docs/react.
22. Документація Formik [Електронний ресурс]. Режим доступу до ресурсу: https://formik.org/docs/overview.
23. Документація Framer Motion [Електронний ресурс]. Режим доступу до ресурсу: https://www.framer.com/motion.
24. Документація Yup [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/jquense/yup.
25. Документація RxJS [Електронний ресурс]. Режим доступу до ресурсу: https://rxjs.dev/guide/overview.
26. Документація Redux Observable [Електронний ресурс]. Режим доступу до ресурсу: https://redux-observable.js.org.

27. Документація Moment [Електронний ресурс]. Режим доступу до ресурсу: https://momentjs.com/docs.
28. Документація React Responsive [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/yocontra/react-responsive.
29. Документація CSS [Електронний ресурс]. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/CSS.
30. Документація SASS [Електронний ресурс]. Режим доступу до ресурсу: https://sass-lang.com/documentation.
31. Документація CSS Modules [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/css-modules/css-modules.

ДОДАТКИ

GraphQL поле, що відповідає за відправку повідомлення

```
Field<NonNullGraphType<ListGraphType<MessageType>>, IEnumerable<Message>>()
     .Name("Send")
     .Argument<NonNullGraphType<SentMessageInputType>, SentMessageInput>("Input", "")
     .ResolveAsync(async context =>
     {
         var sentMessageInput = context.GetArgument<SentMessageInput>("Input");
         var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();
         await sentMessageInputValidator.ValidateAndThrowAsync(sentMessageInput);

         var chat = await chatManager.GetByIdentifierAsync(sentMessageInput.Identifier,
currentUserId);
         if (!chat.UserChats.Any(uc => uc.UserId == currentUserId))
             throw new ExecutionError("User can sent messages only to chats that he
participate.");

         Message? replyMessage = null;

         var replyMessageId = sentMessageInput.ReplyMessageId;
         if (replyMessageId != null)
         {
             replyMessage = await messageProvider.GetByIdAsync(replyMessageId.Value);
             if (replyMessage.ChatId != chat.Id)
                 throw new ExecutionError("User can reply messages only from one chat");
         }

         var newMessages = new List<Message>();

         if (sentMessageInput.Files != null && sentMessageInput.Files.Count() > 0)
         {
             foreach (var file in sentMessageInput.Files)
             {
                 var fileUrl = await
fileManagerService.UploadFileAsync(FileManagerConstants.FilesFolder, file);
                 var newMessage = new Message()
                 {
                     ChatId = chat.Id,
                     Text = sentMessageInput.Files.Count() == newMessages.Count - 1 ?
sentMessageInput.Text : null,
                     FromId = currentUserId,
                     ReplyMessageId = replyMessage?.Id,
                     FileUrl = fileUrl,
                     MediaKind = sentMessageInput.MediaKind,
                     MimeType = file.ContentType,
                 };
                 newMessages.Add(newMessage);
             }
         }

         if (sentMessageInput.ForwardedMessageIds != null &&
sentMessageInput.ForwardedMessageIds.Count() > 0)
         {
             if (newMessages.Count == 0 && !string.IsNullOrEmpty(sentMessageInput.Text))
             {
                 var newMessage = new Message()
                 {
                     ChatId = chat.Id,
                     Text = sentMessageInput.Text,
                     FromId = currentUserId,
                     ReplyMessageId = replyMessage?.Id,
                 };
                 newMessages.Add(newMessage);
             }

             foreach (var forwardedMessageId in sentMessageInput.ForwardedMessageIds)
```

```
            {
                var message = await messageProvider.GetByIdAsync(forwardedMessageId);
                var newMessage = new Message()
                {
                    ChatId = chat.Id,
                    FromId = currentUserId,
                    ReplyMessageId = replyMessage?.Id,
                    ForwardedMessage = ForwardedMessage.GetForwardedMessage(message),
                };
                newMessages.Add(newMessage);
            }
        }

        if (newMessages.Count == 0)
        {
            if (string.IsNullOrEmpty(sentMessageInput.Text))
                throw new ExecutionError("Message text can not be empty");

            Message newMessage = new Message()
            {
                ChatId = chat.Id,
                Text = sentMessageInput.Text,
                FromId = currentUserId,
                ReplyMessageId = replyMessage?.Id,
            };
            newMessages.Add(newMessage);
        }

        var createdMessages = new List<Message>();
        foreach (var newMessage in newMessages)
        {
            var createdMessage = await messageProvider.CreateAsync(newMessage);
            messageActionSubscriptionService.Notify(createdMessage, MessageActionKind.Create);
            createdMessages.Add(createdMessage);
        }

        return createdMessages;

    })
    .AuthorizeWith(AuthPolicies.Authenticated);
```

Клас, що відповідає за автентифікацію

```
using Geesemon.Model.Enums;
using Geesemon.Model.Models;
using Geesemon.Web.GraphQL.Auth;
using Geesemon.Web.Utils.SettingsAccess;

using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;

using MyCSharp.HttpUserAgentParser;
using Newtonsoft.Json;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace Geesemon.Web.Services;

public class AuthService
{
    private readonly ISettingsProvider settingsProvider;
    private readonly IHttpContextAccessor httpContextAccessor;

    public AuthService(ISettingsProvider settingsProvider, IHttpContextAccessor
httpContextAccessor)
    {
        this.settingsProvider = settingsProvider;
        this.httpContextAccessor = httpContextAccessor;
    }

    public const string Bearer = "Bearer";

    public string GenerateAccessToken(Guid userId, string userIdentifier, UserRole userRole)
    {
        SymmetricSecurityKey key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(settingsProvider.GetAuthIssuerSigningKey()));
        SigningCredentials signingCredentials = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);

        List<Claim> claims = new List<Claim>
        {
            new Claim(AuthClaimsIdentity.DefaultIdClaimType, userId.ToString()),
            new Claim(AuthClaimsIdentity.DefaultIdentifierClaimType, userIdentifier),
            new Claim(ClaimsIdentity.DefaultRoleClaimType, userRole.ToString()),
        };
        JwtSecurityToken token = new JwtSecurityToken(
                claims: claims,
                expires: DateTime.UtcNow.AddDays(30),
                signingCredentials: signingCredentials);
        return Bearer + " " + new JwtSecurityTokenHandler().WriteToken(token);
    }

    public ClaimsPrincipal ValidateAccessToken(string token)
    {
        try
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(settingsProvider.GetAuthIssuerSigningKey());
            tokenHandler.ValidateToken(CleanBearerInToken(token), new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new SymmetricSecurityKey(key),
                ValidateIssuer = false,
                ValidateAudience = false,
            }, out var validatedToken);
```

```
            var jwtToken = (JwtSecurityToken)validatedToken;

            ClaimsIdentity claimsIdentity = new ClaimsIdentity(jwtToken.Claims,
JwtBearerDefaults.AuthenticationScheme);

            return new ClaimsPrincipal(claimsIdentity);
        }
        catch
        {
            return null;
        }
    }

    public string GenerateLoginToken()
    {
        SymmetricSecurityKey key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(settingsProvider.GetAuthIssuerSigningKey()));
        SigningCredentials signingCredentials = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);

        JwtSecurityToken token = new JwtSecurityToken(
                expires: DateTime.UtcNow.AddMinutes(30),
                signingCredentials: signingCredentials);
        return new JwtSecurityTokenHandler().WriteToken(token);
    }

    public ClaimsPrincipal ValidateLoginToken(string token)
    {
        try
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(settingsProvider.GetAuthIssuerSigningKey());
            tokenHandler.ValidateToken(token, new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new SymmetricSecurityKey(key),
                ValidateIssuer = false,
                ValidateAudience = false,
            }, out var validatedToken);

            var jwtToken = (JwtSecurityToken)validatedToken;

            ClaimsIdentity claimsIdentity = new ClaimsIdentity(jwtToken.Claims,
JwtBearerDefaults.AuthenticationScheme);

            return new ClaimsPrincipal(claimsIdentity);
        }
        catch
        {
            return null;
        }
    }

    public string? CleanBearerInToken(string token)
    {
        return token?.Replace(Bearer + " ", string.Empty);
    }

    public async Task<Session> FillSession(Session session, bool isOnline)
    {
        var ipAddress =
httpContextAccessor.HttpContext.Connection.RemoteIpAddress.MapToIPv4().ToString();
        string? location;
        if (ipAddress == "127.0.0.1" || ipAddress == "0.0.0.1")
        {
            location = "-, -";
        }
        else
        {
```

```
            try
            {
                using var client = new HttpClient() { Timeout = TimeSpan.FromSeconds(5) };
                client.DefaultRequestHeaders.Add("apikey",
Environment.GetEnvironmentVariable("APILAYER_KEY"));
                var result = await
client.GetAsync($"https://api.apilayer.com/ip_to_location/{ipAddress}");
                dynamic response = JsonConvert.DeserializeObject(await
result.Content.ReadAsStringAsync());
                location = $"{response.region_name}, {response.country_name}";
            }
            catch
            {
                location = "-, -";
            }
        }
        var userAgentString = httpContextAccessor.HttpContext.Request.Headers["User-
Agent"].ToString();
        var userAgent = HttpUserAgentParser.Parse(userAgentString);

        session.LastTimeOnline = DateTime.UtcNow;
        session.IsOnline = isOnline;
        session.IpAddress = ipAddress;
        session.UserAgent = $"{userAgent.Name}, {userAgent.Platform.Value.Name}";
        session.Location = location;
        return session;
    }
}
```

Клас, що відповідає за валідування сесії

```
using Microsoft.AspNetCore.Authentication;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using Microsoft.Net.Http.Headers;
using System.IdentityModel.Tokens.Jwt;
using System.Text.Encodings.Web;
using System.Text;
using Geesemon.Web.Utils.SettingsAccess;
using Geesemon.DataAccess.Managers;
using Geesemon.Web.Services;

namespace Geesemon.Web.Middlewares;

public class BasicAuthenticationOptions : AuthenticationSchemeOptions
{
}

public class BasicAuthenticationHandler : AuthenticationHandler<BasicAuthenticationOptions>
{
    public const string SchemeName = "GeesemonSchemeName";
    private readonly SessionManager sessionManager;
    private readonly ISettingsProvider settingsProvider;
    private readonly AuthService authService;

    public BasicAuthenticationHandler(
        IOptionsMonitor<BasicAuthenticationOptions> options,
        ILoggerFactory logger,
        UrlEncoder encoder,
        ISystemClock clock,
        SessionManager sessionManager,
        ISettingsProvider settingsProvider,
        AuthService authService) : base(options, logger, encoder, clock)
    {
        this.sessionManager = sessionManager;
        this.settingsProvider = settingsProvider;
        this.authService = authService;
    }

    protected override async Task<AuthenticateResult> HandleAuthenticateAsync()
    {
        string token = Request.Headers[HeaderNames.Authorization];
        var handler = new JwtSecurityTokenHandler();
        var validations = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(settingsProvider.GetAuthIssuerSigningKey())),
            ValidateIssuer = false,
            ValidateAudience = false
        };
        try
        {
            var claimsPrincipal = handler.ValidateToken(authService.CleanBearerInToken(token),
validations, out var tokenSecure);
            var userId = claimsPrincipal.Claims.GetUserId();
            var sessions = await sessionManager.GetAsync(t => t.UserId == userId);
            if (!sessions.Any(t => t.Token == token))
                throw new Exception("Bad token");

            var ticket = new AuthenticationTicket(claimsPrincipal, new AuthenticationProperties {
IsPersistent = false }, SchemeName);
            return AuthenticateResult.Success(ticket);
        }
        catch (Exception ex)
```

```
            {
                Console.WriteLine(ex.Message);
                return AuthenticateResult.Fail(ex);
            }
        }
    }
```

Додаток Г

Клас, що відповідає за операції з чатами

```
using FluentValidation;

using Geesemon.DataAccess.Dapper.Providers;
using Geesemon.DataAccess.Managers;
using Geesemon.Model.Enums;
using Geesemon.Model.Models;
using Geesemon.Web.Extensions;
using Geesemon.Web.GraphQL.Auth;
using Geesemon.Web.GraphQL.Types;
using Geesemon.Web.Services.ChatActionsSubscription;
using Geesemon.Web.Services.FileManagers;
using Geesemon.Web.Services.MessageSubscription;

using GraphQL;
using GraphQL.Types;

namespace Geesemon.Web.GraphQL.Mutations
{
    public class ChatMutation : ObjectGraphType
    {
        private readonly IHttpContextAccessor httpContextAccessor;
        private readonly IFileManagerService fileManagerService;
        private readonly IChatActionSubscriptionService chatActionSubscriptionService;
        private readonly IMessageActionSubscriptionService messageSubscriptionService;
        private readonly IChatMembersSubscriptionService chatMembersSubscriptionService;
        private readonly ChatManager chatManager;
        private readonly UserChatManager userChatManager;
        private readonly UserProvider userProvider;
        private readonly IServiceProvider serviceProvider;
        private readonly IValidator<CreateGroupChatInput> createGroupChatInputValidator;
        private readonly IValidator<UpdateChatInput> updateChatInputValidator;
        private readonly MessageProvider messageProvider;

        public ChatMutation(
            IHttpContextAccessor httpContextAccessor,
            IFileManagerService fileManagerService,
            IChatActionSubscriptionService chatActionSubscriptionService,
            IMessageActionSubscriptionService messageSubscriptionService,
            IChatMembersSubscriptionService chatMembersSubscriptionService,
            ChatManager chatManager,
            UserChatManager userChatManager,
            UserProvider userProvider,
            IServiceProvider serviceProvider,
            IValidator<CreateGroupChatInput> createGroupChatInputValidator,
            IValidator<UpdateChatInput> updateChatInputValidator,
            MessageProvider messageProvider
            )
        {
            Field<NonNullGraphType<ChatType>, Chat>()
                .Name("CreatePersonal")
                .Argument<CreatePersonalChatInputType>("Input", "Chat input for creating new
chat.")
                .ResolveAsync(ResolveCreatePersonal)
                .AuthorizeWith(AuthPolicies.Authenticated);

            Field<NonNullGraphType<ChatType>, Chat>()
                .Name("CreateGroup")
                .Argument<CreateGroupChatInputType>("Input", "Chat input for creating new chat.")
                .ResolveAsync(ResolveCreateGroup)
                .AuthorizeWith(AuthPolicies.Authenticated);

            Field<NonNullGraphType<BooleanGraphType>, bool>()
                .Name("Delete")
                .Argument<NonNullGraphType<GuidGraphType>>("Input", "Chat id for delete chat.")
```

```
                .ResolveAsync(ResolveDelete)
                .AuthorizeWith(AuthPolicies.Authenticated);

            Field<NonNullGraphType<ChatType>, Chat>()
                .Name("Update")
                .Argument<UpdateChatInputType>("Input", "Chat input for updating chat.")
                .ResolveAsync(ResolveUpdate)
                .AuthorizeWith(AuthPolicies.Authenticated);

            Field<NonNullGraphType<ListGraphType<UserType>>, IEnumerable<User>>()
                .Name("AddMembers")
                .Argument<NonNullGraphType<ChatsAddMembersInputType>,
ChatsAddMembersInput>("Input", "")
                .ResolveAsync(ResolveAddMembers)
                .AuthorizeWith(AuthPolicies.Authenticated);

            Field<NonNullGraphType<ListGraphType<UserType>>, IEnumerable<User>>()
                .Name("RemoveMembers")
                .Argument<NonNullGraphType<ChatsAddMembersInputType>,
ChatsAddMembersInput>("Input", "")
                .ResolveAsync(ResolveRemoveMembers)
                .AuthorizeWith(AuthPolicies.Authenticated);

            Field<NonNullGraphType<BooleanGraphType>, bool>()
                .Name("LeaveChat")
                .Argument<NonNullGraphType<GuidGraphType>, Guid>("ChatId", "")
                .ResolveAsync(ResolveLeaveChat)
                .AuthorizeWith(AuthPolicies.Authenticated);

            this.httpContextAccessor = httpContextAccessor;
            this.fileManagerService = fileManagerService;
            this.chatActionSubscriptionService = chatActionSubscriptionService;
            this.messageSubscriptionService = messageSubscriptionService;
            this.chatMembersSubscriptionService = chatMembersSubscriptionService;
            this.chatManager = chatManager;
            this.userChatManager = userChatManager;
            this.userProvider = userProvider;
            this.serviceProvider = serviceProvider;
            this.createGroupChatInputValidator = createGroupChatInputValidator;
            this.updateChatInputValidator = updateChatInputValidator;
            this.messageProvider = messageProvider;
        }

        private async Task<Chat?> ResolveCreatePersonal(IResolveFieldContext context)
        {
            var chatInp = context.GetArgument<CreatePersonalChatInput>("Input");
            var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();

            var checkChat = await chatManager.GetByIdentifierAsync(chatInp.Identifier,
currentUserId);
            if (checkChat != null)
                throw new Exception("Personal chat already exists");

            var oppositeUser = await userProvider.GetByIdentifierAsync(chatInp.Identifier);

            var chat = new Chat
            {
                Type = oppositeUser.Id == currentUserId ? ChatKind.Saved : ChatKind.Personal,
                CreatorId = currentUserId
            };
            chat = await chatManager.CreateAsync(chat);

            var userChat = new List<UserChat>
            {
                new UserChat { UserId = currentUserId, ChatId = chat.Id },
            };
            if (oppositeUser.Id != currentUserId)
                userChat.Add(new UserChat { UserId = oppositeUser.Id, ChatId = chat.Id });
            await userChatManager.CreateManyAsync(userChat);
```

```
            chat = await chat.MapForUserAsync(currentUserId, serviceProvider);
            chatActionSubscriptionService.Notify(chat, ChatActionKind.Add, userChat.Select(uc =>
uc.UserId));
            return chat;
        }

        private async Task<Chat?> ResolveCreateGroup(IResolveFieldContext context)
        {
            var chatManager = context.RequestServices.GetRequiredService<ChatManager>();
            var userChatManager = context.RequestServices.GetRequiredService<UserChatManager>();
            var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();
            var currentUserIdentifier =
httpContextAccessor.HttpContext.User.Claims.GetIdentifier();
            var chatInput = context.GetArgument<CreateGroupChatInput>("Input");
            await createGroupChatInputValidator.ValidateAndThrowAsync(chatInput);

            string imageUrl = null;
            if (chatInput.Image != null)
                imageUrl = await
fileManagerService.UploadFileAsync(FileManagerConstants.GroupImagesFolder, chatInput.Image);

            var chat = new Chat
            {
                Type = ChatKind.Group,
                CreatorId = currentUserId,
                Name = chatInput.Name,
                Identifier = chatInput.Identifier,
                ImageUrl = imageUrl,
            };
            chat = await chatManager.CreateAsync(chat);

            var userChat = new List<UserChat>
            {
                new UserChat { UserId = currentUserId, ChatId = chat.Id }
            };
            foreach (var userId in chatInput.UsersId)
                userChat.Add(new UserChat { UserId = userId, ChatId = chat.Id });
            await userChatManager.CreateManyAsync(userChat);

            chatActionSubscriptionService.Notify(chat, ChatActionKind.Add, userChat.Select(uc =>
uc.UserId));

            await messageSubscriptionService.SentSystemGeeseMessageAsync("ChatCreatedMessage",
chat.Id, new string[] { "@" + currentUserIdentifier, chat.Name });
            return chat;
        }

        private async Task<bool> ResolveDelete(IResolveFieldContext context)
        {
            var chatInput = context.GetArgument<Guid>("Input");
            var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();
            var chat = await chatManager.GetByIdAsync(chatInput);

            Exception exception = new Exception("User can only delete personal chats or chats he
own.");

            if (chat == null)
                throw new Exception($"Chat with id {chatInput} doesn't exist.");

            if (!await chatManager.IsUserInChat(currentUserId, chatInput))
                throw exception;

            if (chat.Type != ChatKind.Personal && chat.CreatorId != currentUserId)
                throw exception;

            //NOTE: Because functionality in ChatActionSubscriptionService depends on chat being
in database we need to run notify before deletion
            var userChats = await userChatManager.Get(chat.Id);
```

```
            await chatManager.RemoveAsync(chat.Id);
            chatActionSubscriptionService.Notify(chat, ChatActionKind.Delete, userChats.Select(uc
=> uc.UserId));
            return true;
        }

        private async Task<Chat?> ResolveUpdate(IResolveFieldContext context)
        {
            var chatUpdateInput = context.GetArgument<UpdateChatInput>("Input");
            await updateChatInputValidator.ValidateAndThrowAsync(chatUpdateInput);
            var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();

            var chat = await chatManager.GetByIdAsync(chatUpdateInput.Id);

            Exception exception = new Exception("User can update only group chats he own.");

            if (!await chatManager.IsUserInChat(currentUserId, chat.Id))
                throw exception;

            if (chat.Type != ChatKind.Group && chat.CreatorId != currentUserId)
                throw exception;

            if (chatUpdateInput.Image != null)
                chat.ImageUrl = await
fileManagerService.UploadFileAsync(FileManagerConstants.GroupImagesFolder,
chatUpdateInput.Image);

            chat.Name = chatUpdateInput.Name;
            chat.Identifier = chatUpdateInput.Identifier;
            await chatManager.UpdateAsync(chat);

            var userChats = await userChatManager.Get(chat.Id);
            chatActionSubscriptionService.Notify(chat, ChatActionKind.Update, userChats.Select(uc
=> uc.UserId));
            return chat;
        }

        private async Task<IEnumerable<User>> ResolveAddMembers(IResolveFieldContext context)
        {
            var chatsAddMembersInput = context.GetArgument<ChatsAddMembersInput>("Input");
            var chat = await chatManager.GetByIdAsync(chatsAddMembersInput.ChatId);
            var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();
            var currentIdentifier = httpContextAccessor.HttpContext.User.Claims.GetIdentifier();

            Exception exception = new Exception("User can update only group chats he own.");
            if (chat == null || !await chatManager.IsUserInChat(currentUserId, chat.Id))
                throw exception;

            if (chat.Type != ChatKind.Group || chat.CreatorId != currentUserId)
                throw exception;

            var newUserChats = new List<UserChat>();
            foreach (var userId in chatsAddMembersInput.UserIds)
            {
                var user = await userProvider.GetByIdAsync(userId);
                if (user == null)
                    throw new ExecutionError($"User with id {user.Id} not found");

                var userChat = await userChatManager.Get(chatsAddMembersInput.ChatId, userId);
                if (userChat == null)
                    newUserChats.Add(new UserChat
                    {
                        ChatId = chatsAddMembersInput.ChatId,
                        UserId = user.Id,
                        User = user,
                    });
            }

            await userChatManager.CreateManyAsync(newUserChats);
```

```
            chatActionSubscriptionService.Notify(chat, ChatActionKind.Add, newUserChats.Select(uc
=> uc.UserId));

            foreach (var userChat in newUserChats)
            {
                var newMessage = new Message
                {
                    ChatId = chatsAddMembersInput.ChatId,
                    Text = "AddedToChatMessage",
                    Type = MessageKind.SystemGeeseText,
                    GeeseTextArguments = new[] { "@" + currentIdentifier, "@" +
userChat.User.Identifier }
                };
                newMessage = await messageProvider.CreateAsync(newMessage);
                messageSubscriptionService.Notify(newMessage, MessageActionKind.Create);
                chatMembersSubscriptionService.Notify(userChat.User, ChatMembersKind.Add,
chat.Id);
            }
            return newUserChats.Select(uc => uc.User);
        }

        private async Task<IEnumerable<User>> ResolveRemoveMembers(IResolveFieldContext context)
        {
            var chatsAddMembersInput = context.GetArgument<ChatsAddMembersInput>("Input");
            var chat = await chatManager.GetByIdAsync(chatsAddMembersInput.ChatId);
            var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();
            var currentIdentifier = httpContextAccessor.HttpContext.User.Claims.GetIdentifier();

            if (chat == null || !await chatManager.IsUserInChat(currentUserId, chat.Id) ||
chat.Type != ChatKind.Group || chat.CreatorId != currentUserId)
                throw new Exception("User can update only group chats he own.");

            var removeUserChats = new List<UserChat>();
            foreach (var userId in chatsAddMembersInput.UserIds)
            {
                var user = await userProvider.GetByIdAsync(userId);
                if (user == null)
                    throw new ExecutionError($"User with id {user.Id} not found");

                var userChat = await userChatManager.Get(chatsAddMembersInput.ChatId, userId);
                if (userChat == null)
                    throw new ExecutionError($"User {user.Identifier} not in chat");

                removeUserChats.Add(userChat);
            }

            chatActionSubscriptionService.Notify(chat, ChatActionKind.Delete,
removeUserChats.Select(uc => uc.UserId));

            foreach (var userChat in removeUserChats)
            {
                await userChatManager.RemoveAsync(userChat);

                var newMessage = new Message
                {
                    ChatId = chatsAddMembersInput.ChatId,
                    Text = "RemovedFromChatMessage",
                    Type = MessageKind.SystemGeeseText,
                    GeeseTextArguments = new[] { "@" + currentIdentifier, "@" +
userChat.User.Identifier }
                };
                newMessage = await messageProvider.CreateAsync(newMessage);
                messageSubscriptionService.Notify(newMessage, MessageActionKind.Create);
                chatMembersSubscriptionService.Notify(userChat.User, ChatMembersKind.Delete,
chat.Id);
            }
            return removeUserChats.Select(uc => uc.User);
        }
```

```
        private async Task<bool> ResolveLeaveChat(IResolveFieldContext context)
        {
            var chatId = context.GetArgument<Guid>("ChatId");
            var chat = await chatManager.GetByIdAsync(chatId);
            var currentUserId = httpContextAccessor.HttpContext.User.Claims.GetUserId();
            var currentIdentifier = httpContextAccessor.HttpContext.User.Claims.GetIdentifier();

            if (chat == null || !await chatManager.IsUserInChat(currentUserId, chat.Id) ||
chat.Type != ChatKind.Group)
                throw new Exception("User can leave only chats that he is in.");

            chatActionSubscriptionService.Notify(chat, ChatActionKind.Delete, new List<Guid> {
currentUserId });

            var userChat = new UserChat() { ChatId = chat.Id, UserId = currentUserId };

            var uc = await userChatManager.RemoveAsync(userChat);

            var newMessage = new Message
            {
                ChatId = chatId,
                Text = "UserLeftFromChatMessage",
                Type = MessageKind.SystemGeeseText,
                GeeseTextArguments = new[] { "@" + currentIdentifier }
            };
            newMessage = await messageProvider.CreateAsync(newMessage);
            messageSubscriptionService.Notify(newMessage, MessageActionKind.Create);
            chatMembersSubscriptionService.Notify(userChat.User, ChatMembersKind.Delete,
chat.Id);

            return true;
        }
    }
}
```

Основний компонент клієнтської частини

```
import { useSubscription } from '@apollo/client';
import { FC, useEffect } from 'react';
import { Navigate, Route, Routes } from 'react-router-dom';
import { authActions } from './behavior/features/auth/slice';
import { chatActions } from './behavior/features/chats';
import {
  ChatActionsData, ChatActionsVars, CHAT_ACTIONS_SUBSCRIPTIONS, MessageActionsData,
  MessageActionsVars, MESSAGE_ACTIONS_SUBSCRIPTIONS,
} from './behavior/features/chats/subscriptions';
import {
  ChatActionKind,
  MessageActionKind,
} from './behavior/features/chats/types';
import { useAppDispatch, useAppSelector } from './behavior/store';
import { ContentBar } from './components/common/ContentBar/ContentBar';
import { LeftSidebar } from './components/common/LeftSidebar/LeftSidebar';
import { RightSidebar } from './components/common/RightSidebar/RightSidebar';
import { useIsMobile } from './hooks/useIsMobile';
import { localStorageGetItem } from './utils/localStorageUtils';

export const AuthedApp: FC = () => {
  const isMobile = useIsMobile();
  const dispatch = useAppDispatch();
  const isRightSidebarVisible = useAppSelector(s => s.app.isRightSidebarVisible);
  const messageActionSubscription = useSubscription<MessageActionsData,
MessageActionsVars>(MESSAGE_ACTIONS_SUBSCRIPTIONS);
  const chatActionSubscription = useSubscription<ChatActionsData,
ChatActionsVars>(CHAT_ACTIONS_SUBSCRIPTIONS, {
    variables: { token: localStorageGetItem('AuthToken') || '' },
  });

  const makeOfflineAsync = () => {
    dispatch(authActions.toggleOnlineAsync(false));
  };

  useEffect(() => {
    dispatch(authActions.toggleOnlineAsync(true));

    window.addEventListener('beforeunload', makeOfflineAsync);

    return () => {
      window.removeEventListener('beforeunload', makeOfflineAsync);
    };
  }, []);

  useEffect(() => {
    const data = messageActionSubscription.data;
    if (data) {
      switch (data?.messageActions.type) {
      case MessageActionKind.Create:
        dispatch(chatActions.addInStartMessages({
          chatId: data.messageActions.message.chatId,
          messages: [data.messageActions.message],
        }));
        break;
      case MessageActionKind.Update:
        dispatch(chatActions.updateMessage(data.messageActions.message));
        break;
      case MessageActionKind.Delete:
        dispatch(chatActions.deleteMessage(data.messageActions.message));
        break;
      }
    }
  }, [messageActionSubscription.data, dispatch]);
```

```
  useEffect(() => {
    const data = chatActionSubscription.data;
    if (data) {
      switch (data?.chatActions.type) {
      case ChatActionKind.Add:
        dispatch(chatActions.addChats([data.chatActions.chat]));
        break;
      case ChatActionKind.Update:
        dispatch(chatActions.shallowUpdateChat(data.chatActions.chat));
        break;
      case ChatActionKind.Delete:
        dispatch(chatActions.deleteChat(data.chatActions.chat.id));
        break;
      }
    }
  }, [chatActionSubscription.data, dispatch]);

  return (
    <div className={'authedRoutes'}>
      {isMobile
        ? (
          <Routes>
            <Route path={'/'} element={<LeftSidebar />} />
            <Route
              path={'/:chatIdentifier'}
              element={isRightSidebarVisible ? <RightSidebar /> : <ContentBar />}
            />
            <Route path={'/auth/*'} element={<Navigate to={'/'} />} />
          </Routes>
        )
        : (
          <>
            <Routes>
              <Route path={'/'} element={<LeftSidebar />} />
              <Route path={'/:chatIdentifier'} element={<LeftSidebar />} />
              <Route path={'/auth'} element={<Navigate to={'/'} />} />
              <Route path={'/auth/*'} element={<Navigate to={'/'} />} />
            </Routes>
            <Routes>
              <Route path={'/'} element={<ContentBar />} />
              <Route path={'/:chatIdentifier'} element={<ContentBar />} />
            </Routes>
            <Routes>
              <Route path={'/:chatIdentifier'} element={<RightSidebar />} />
              <Route path={'*'} element={<RightSidebar />} />
            </Routes>
          </>
        )}
    </div>
  );
};
```