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. | 27.02.2024 | 27.02.2024 |
| 2 | Гордієнко І. В. | 27.02.2024 | 27.02.2024 |
| 3 | Гордієнко І. В. | 27.02.2024 | 27.02.2024 |

РЕФЕРАТ

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

Пояснювальна записка до випускної роботи містить 75 сторінок, 42 ілюстрації, 17 таблиць, 4 додатки та 26 літературних джерел.

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

В роботі визначено основні завдання на розробку програми, проаналізовано аналоги розробленої додатку. Обгрунтовано використання архітектури програми. Наведено сценарії роботи програмного продукту, загальна структура програмного комплексу, опис алгоритмів взаємодії окремих модулів системи, опис об’єктної структури системи (діаграма класів). Також наводиться 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 75 pages, 42 illustrations, 17 tables, 4 appendices and 26 literary sources.

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

The paper defines the main tasks for the development of the program, and analyzes the analogues of the developed system. The use of the application architecture is justified. Scenarios of the software complex, the general structure of the software complex, a description of the interaction algorithms of individual system modules, a description of the object structure of the system (class diagram) are given. The results of system testing are also provided.

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

ЗМІСТ

[ВСТУП 6](#_Toc168306078)

[РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-МЕСЕНДЖЕРУ 8](#_Toc168306079)

[1.1. Постановка задачі 8](#_Toc168306080)

[1.2. Аналіз аналогів програмного продукту 9](#_Toc168306081)

[1.3. Вибір архітектури веб-додатку 13](#_Toc168306082)

[1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення 16](#_Toc168306083)

[Висновки до першого розділу 20](#_Toc168306084)

[РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-МЕСЕНДЖЕРУ 21](#_Toc168306085)

[2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи 21](#_Toc168306086)

[2.2. Розробка бази даних системи 25](#_Toc168306087)

[2.3. Проектування та реалізація алгоритмів роботи системи 29](#_Toc168306088)

[2.4. Реалізація веб-месенджеру 32](#_Toc168306089)

[Висновки до другого розділу 37](#_Toc168306090)

[РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-МЕСЕНДЖЕРОМ 38](#_Toc168306091)

[3.1. Порядок встановлення та налаштування параметрів системи 38](#_Toc168306092)

[3.2. Структура інтерфейсу та порядок взаємодії з онлайн-сервісом 39](#_Toc168306093)

[3.3. Тестування роботи програмного продукту 50](#_Toc168306094)

[Висновки до третього розділу 54](#_Toc168306095)

[ВИСНОВКИ 55](#_Toc168306096)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 57](#_Toc168306097)

[ДОДАТКИ 58](#_Toc168306098)

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, політичну агітацію, 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 випускної роботи** є розробка веб-месенджеру.

Дана мета передбачає виконання наступних завдань:

* проведення аналізу вимог до програми;
* аналіз аналогів програмного продукту;
* вибір архітектури системи;
* обґрунтування та вибір засобів реалізації;
* визначення варіантів використання та проектування об’єктноорієнтованої структури системи;
* проектування алгоритмів системи;
* безпосередня реалізація застосунку;
* опис структури інтерфейсу та порядку взаємодії зі системою;
* тестування розробленого додатку.

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

**Предметом дослідження** є використання сучасних веборієнтованих технологій розробки для створення вебзастосунку, що допомагає обмінюватися повідомленнями у мережі.

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

**Практичне значення одержаних результатів**. Розроблений вебзастосунок може використовуватися для особистої та професійної комунікації.

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# та фреймворку ASP.NET. Фронтенд частина має бути реалізовани за допомогою мови JavaScript та бібліотеки React.

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

Програмний продукт повинен мати наступні функціональні характеристики:

* авторизація та реєстрація з можливістю зміни та скидання пароля;
* створювати персональні чати та групи;
* пошук по чатам;
* надсилати текстові, медіа та файлові повідомлення;
* надсилати, редагувати, видаляти, пересилати, відповідати на повідомлення;
* менеджмент активними логін сесіями;

Програмний продукт повинен бути реалізованим на основі клієнт-серверної архітектури. Серверна частина додатку має базуватися на веб-сервері IIS та системі керування базами даних SQL Server. При проектуванні програмного комплексу необхідно застосувати архітектуру SPA (Single-Page Application) [5] для оновлення контенту без перезавантаження веб-сторінки.

Клієнтська частина повинна бути побудована за сучасними стандартами створення односторінкового застосунку, тобто весь необхідний код завантажується разом зі сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею.

Розробка системи передбачає наступні основні етапи:

1. Створення бекенд частини додатку написанням GraphQL API, використовуючи фреймворк ASP.NET та бібліотеку GraphQL.NET, що включає реалізацію наступних сутностей:

* «Чати»;
* «Повідомлення»;
* «Прочитані повідомлення»;
* «Сесії»;
* «Користувачі у чаті»;
* «Користувачі»;

1. Створення дизайну інтерфейсу, з яким будуть взаємодіяти користувачі в системі:

* обрати бібліотеку для реалізації користувацького інтерфейсу;
* розробити відповідні сторінки з використаннямобраної бібліотеки;
* забезпечити сповіщення користувача про завантаження контенту за допомогою відповідного елементу.

1. Налаштувати отримання даних зі створеного API на клієнті та розробити маршрутизацію додатку за допомогою бібліотеки React:

* налаштувати виклик API через Apollo Client;
* організувати маршрутизацію додатку з використанням React router dom;

В результаті виконаних етапів буде розроблена система, побудована за архітектурою SPA (Single-Page Application), яка 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.

На рис. 1.1 зображено додаток Telegram.

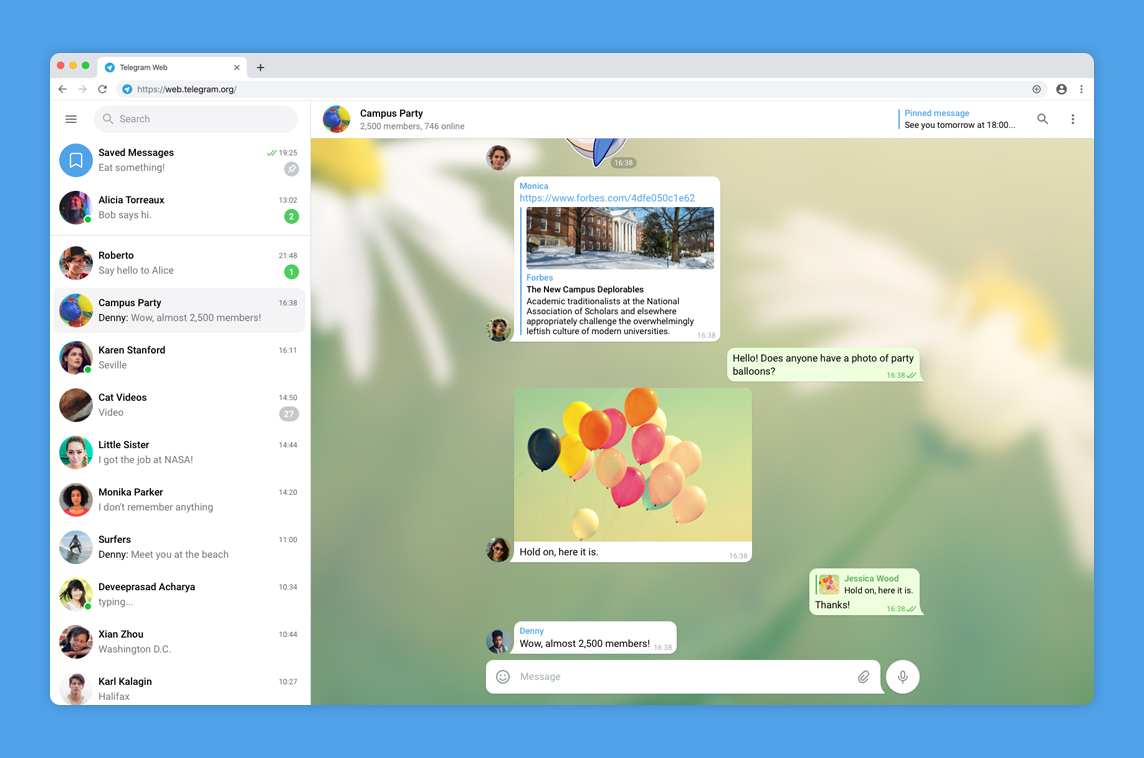


Рисунок 1.1 - Telegram

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.

На рис. 1.2 зображено додаток **WhatsApp**.

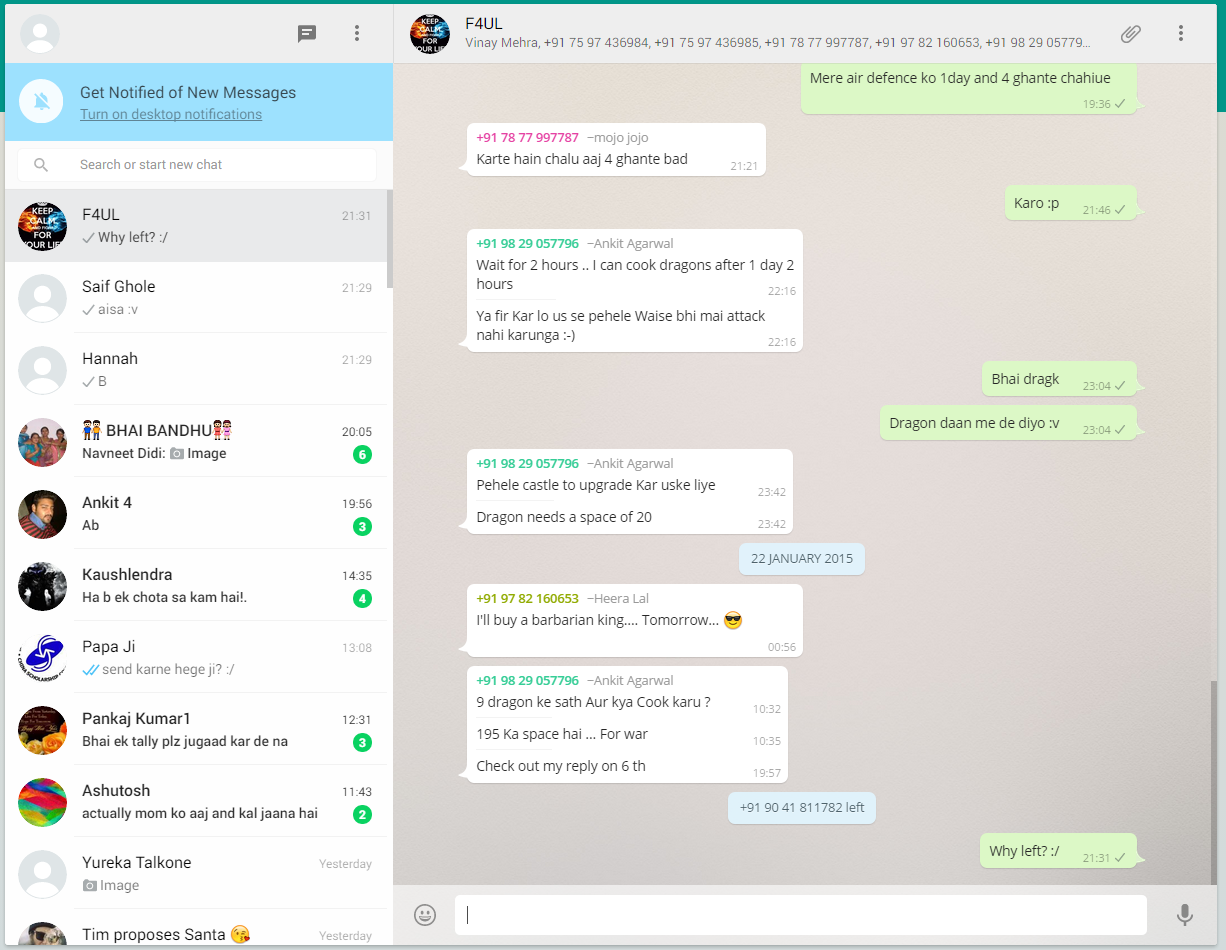


Рисунок 1.2 - **WhatsApp**

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.

На рис. 1.3 зображено додаток Viber.

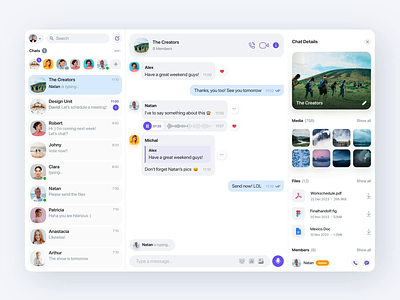


Рисунок 1.3 - Viber

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.

Таблиця 1.1

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

Ґрунтуючись на отриманих результатах, отримуємо наступні критерії для додатку, що розроблюється:

* інтуїтивний та зрозумілий дизайн;
* висока безпека;
* широка інтеграція;
* висока стабільність та швидкість;
* мале споживання ресурсів;

1.3. Вибір архітектури веб-додатку

В сучасному світі існує незліченна кількість архітектур для побудови вебдодатку. Розглянемо декілька з них, такі як:

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

Тепер докладніше розберемо кожну схему.

MVC – це архітектурний шаблон, який дозволяє розділити компоненти додатку на три основні частини: модель (дані), вид (інтерфейс користувача) і контролер (бізнес-логіка). Дозволяє забезпечити розділення відповідальностей та полегшує розробку, тестування та підтримку програмного забезпечення

На рис. 1.4 зображено схему MVC.

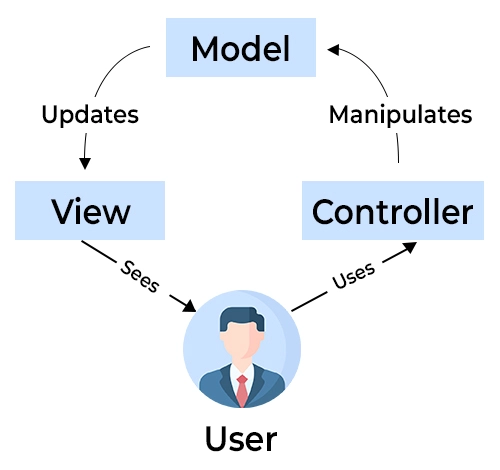


Рисунок. 1.4 - Схема MVC

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 та JavaScript завантажується один раз при початковому завантаженні сторінки, після чого весь інший контент та взаємодія з користувачем відбувається без повторної загрузки сторінки з сервера.

На рис. 1.6 зображено схему SPA.

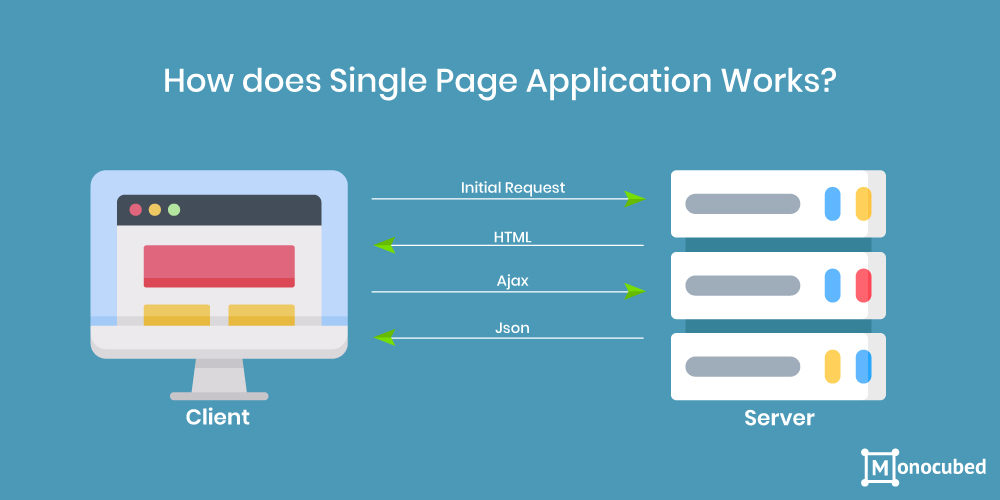


Рисунок 1.6 - Схема SPA

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 старших 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;
* мова препроцесора для стилів: Sass;
* клієнтська частина: мова програмування JavaScript та бібліотека React;
* серверна частина: мова програмування C# та фреймворк ASP.NET;
* база даних: Microsoft SQL Server.

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.NE, що дозволить створити потужний та масштабований серверний додаток. База даних Microsoft SQL Server забезпечить надійне зберігання та управління даними.

Застосування цих технологій дозволить створити сучасний та функціональний додаток веб-месенджер, забезпечивши ефективне використання ресурсів та гнучкість у розробці.

**HTML5** - використовується для створення структури та семантики веб-сторінок.

**Sass** **–** це препроцесор CSS, що дозволяє писати стилі з використанням змінних, міксинів та інших функцій, що полегшують роботу зі стилями.

**React –** це бібліотека JavaScript для розробки інтерфейсів користувача, яка дозволяє створювати ефективні та динамічні веб-додатки.

**C#** **–** використовується для розробки серверної частини. Вона є потужною та об'єктно-орієнтованою мовою з широким набором функціональних можливостей.

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

**Microsoft SQL Server –** 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.

Таблиця 1.4

Максимально можливий обсяг збережених даних для кожної СКБД

|  |  |  |  |
| --- | --- | --- | --- |
|  | Розмір БД | Розмір таблиці | Розмір рядка |
| MySQL | ∞ | 256 ТБ | 64 KB |
| 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.

Таблиця 1.5

Аналіз підтримуваних альтернативами операційних систем

|  |  |  |  |
| --- | --- | --- | --- |
|  | Windows | Linux | Android |
| MySQL | + | + | + |
| MS SQL | + | - | - |
| PostgreSQL | + | + | - |

Порівняємо різні методи захисту даних, які використовуються для уникнення несанкціонованого доступу до конфіденційної інформації в інформаційній системі організації. Далі будемо аналізувати таблицю, в якій вказані системи забезпечення безпеки даних та їх опис.

Таблиця 1.6

Аналіз систем забезпечення безпеки даних в альтернативах

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

Після аналізу систем управління базами даних (СУБД), таких як MySQL, Microsoft SQL Server (MS SQL) та PostgreSQL, встановлено, що PostgreSQL є найбільш підходящим для виконання поставлених завдань. Актуальна версія PostgreSQL має розширений функціонал та можливості, такі як підтримка географічних даних, розширення для складних операцій та механізми захисту даних.

Таблиця 1.7

Аналіз загальних показників СКБД

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 |

Продовження табл. 1.7

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| Індекси | Всі основні, повнотекстовий | Всі основні, повнотекстовий | Звичайні, повнотекстовий |
| Мови маніпулювання | SQL | SQL, QBE, XQuery, багатомірні вирази (для OLAP) | Розширений SQL |
| 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 програмного комплексу були обрані наступні інструментальні засоби та технології:

* база даних: Misrosoft SQL Server;
* мова програмування на серверній частині: C#;
* фреймворк на серверній частині: ASP.NET;
* мова програмування на клієнтській частині: JavaScript;
* фреймворк на клієнтській частині: React;
* мова процесора для стилів: Sass.

Для розробки були використані наступні програмні продукти та 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 для побудови бекенд частини, React - для фронтенд частини; Miscrosoft SQL Server- як систему керування базами даних.

Буде використано такі програмні продукти та 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 [17].

На рис. 2.1 зображено діаграму варіантів використання.

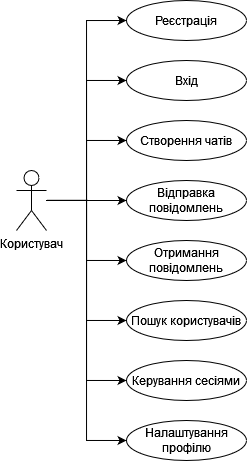


Рисунок 2.1 - Діаграма варіантів використання

Відповідно до побудованої діаграми можна виокремити наступні вимоги системи.

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 Web Socker 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.

1. Продуктивність

* система повинна підтримувати мінімум декілька тисяч користувачів, що одночасно обмінюються повідомленнями.

Зважаючи на те, що серверна частина є об’єктно-орієнтованою, побудуємо діаграму класів провайдерів веб-додатку.

На рис. 2.2 зображено діаграму класів провайдерів додатку.

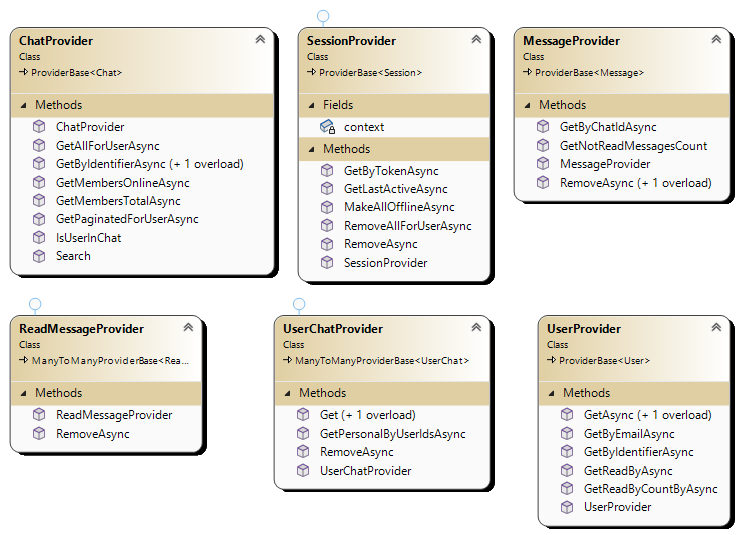


Рисунок 2.2 - Діаграма класів провайдерів додатку

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" [19], 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. Цей патерн часто використовується разом із патерном “Unit Of Work”.На рис. 2.3 зображено реалізацію патерну “Репозиторій” для модуля повідомлень.

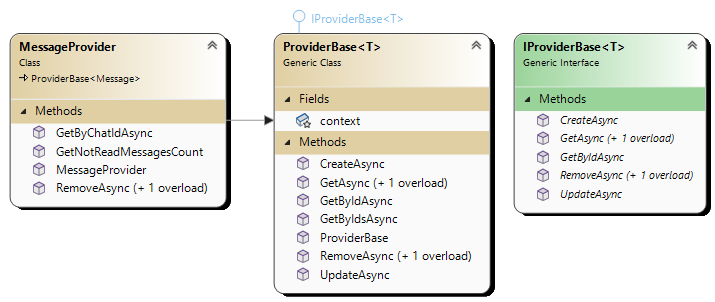


Рисунок 2.3 - Реалізація патерну “Репозиторій” для модуля повідомлень

На рис. 2.4 зображено схему патерну “Репозиторій”.

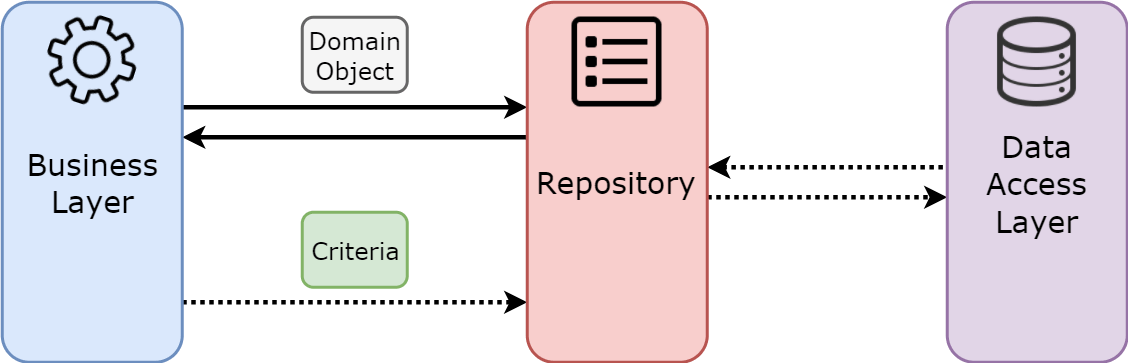


Рисунок 2.4 - Загальний вигляд патерну “Репозиторій”

Крім того, для побудови системи буде використовуватися патерн “Unit Of Work”. Однак, немає потреби реалізовувати його самостійно, оскільки більшість фреймворків надають цей патерн з коробки.

Це патерн об’єктно-реляційної поведінки, призначений для відстеження змін об’єктів під час транзакції. Під час роботи з базою даних важливо відстежувати зміни в об’єктах, інакше дані не будуть оновлені. Це також стосується операцій додавання та видалення.

Можна змінювати дані в сховищі при кожній взаємодії з об’єктом, але це

спричинить численні виклики до бази даних. Очевидно, що це вимагає підтримки транзакції у відкритому стані, що негативно впливає на продуктивність. Патерн “Unit Of Work” пропонує відстежувати всі зміни об’єктів і застосовувати їх у вигляді єдиної транзакції.

На рис. 2.5 зображено загальний вигляд патерну “Unit Of Work”.

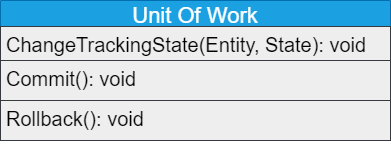


Рисунок 2.5 - Загальний вигляд патерну “Unit Of Work”

2.2. Розробка бази даних системи

Для збереження даних була обрана СУБД Microsoft SQL Server, оскільки вона є однією з найпопулярніших і найкраще підтримуваних систем керування базами даних на сьогоднішній день. Microsoft SQL Server забезпечує широкий функціонал, надійність і високу продуктивність, що робить її чудовим вибором для зберігання даних.

На рис. 2.6 продемонстровано ER-діаграму спроєктованої бази даних.

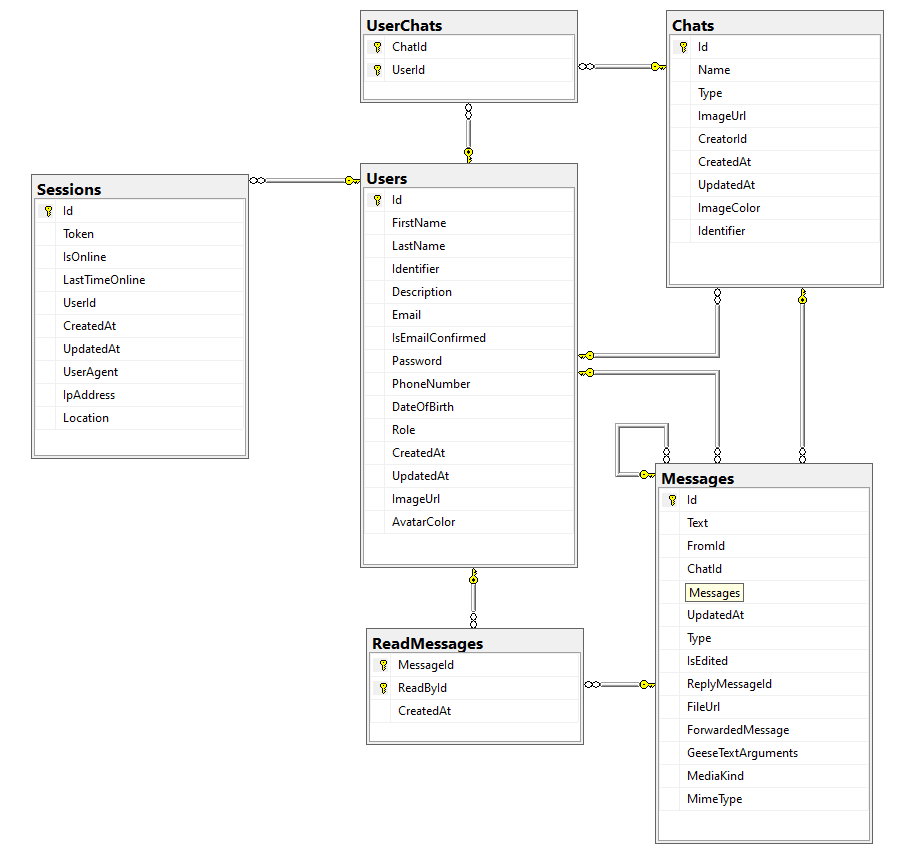


Рисунок 2.6 - ER-діаграма спроєктованої бази даних

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 |

Продовження табл. 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| 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 |

Продовження табл. 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| 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”) |
| 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 |

Продовження табл. 2.5

|  |  |  |  |
| --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* |
| 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, вебзастосунок чітко розділяється на серверну та клієнтську частини, і передбачає використання RESTful API.

RESTful API (Representational State Transfer API) – це набір правил та протоколів, що дозволяють різним програмним додаткам взаємодіяти через мережу, зазвичай через Інтернет.

Можна виділити такі основні характеристики та принципи RESTful API:

* **Ресурси:** RESTful API організований навколо ресурсів, які зазвичай представляються у вигляді URL-адрес. Кожен ресурс має унікальний URL, і клієнти взаємодіють з цими ресурсами, виконуючи HTTP-запити до відповідних URL-адрес.
* **Методи HTTP**: RESTful API використовує стандартні методи HTTP для виконання різних дій з ресурсами. Під час розробки додатку застосовуються такі методи:

1. POST: Створює новий ресурс.

У випадку створення цього додатку відбувається наступне: клієнтська частина звертається до API для отримання необхідних даних. Використовуючи ASP.NET, на відповідний запит з клієнтської частини через бібліотеку Apollo Client 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 вебзастосунку.

На рис. 2.7 зображено загальний алгоритм роботи додатку.

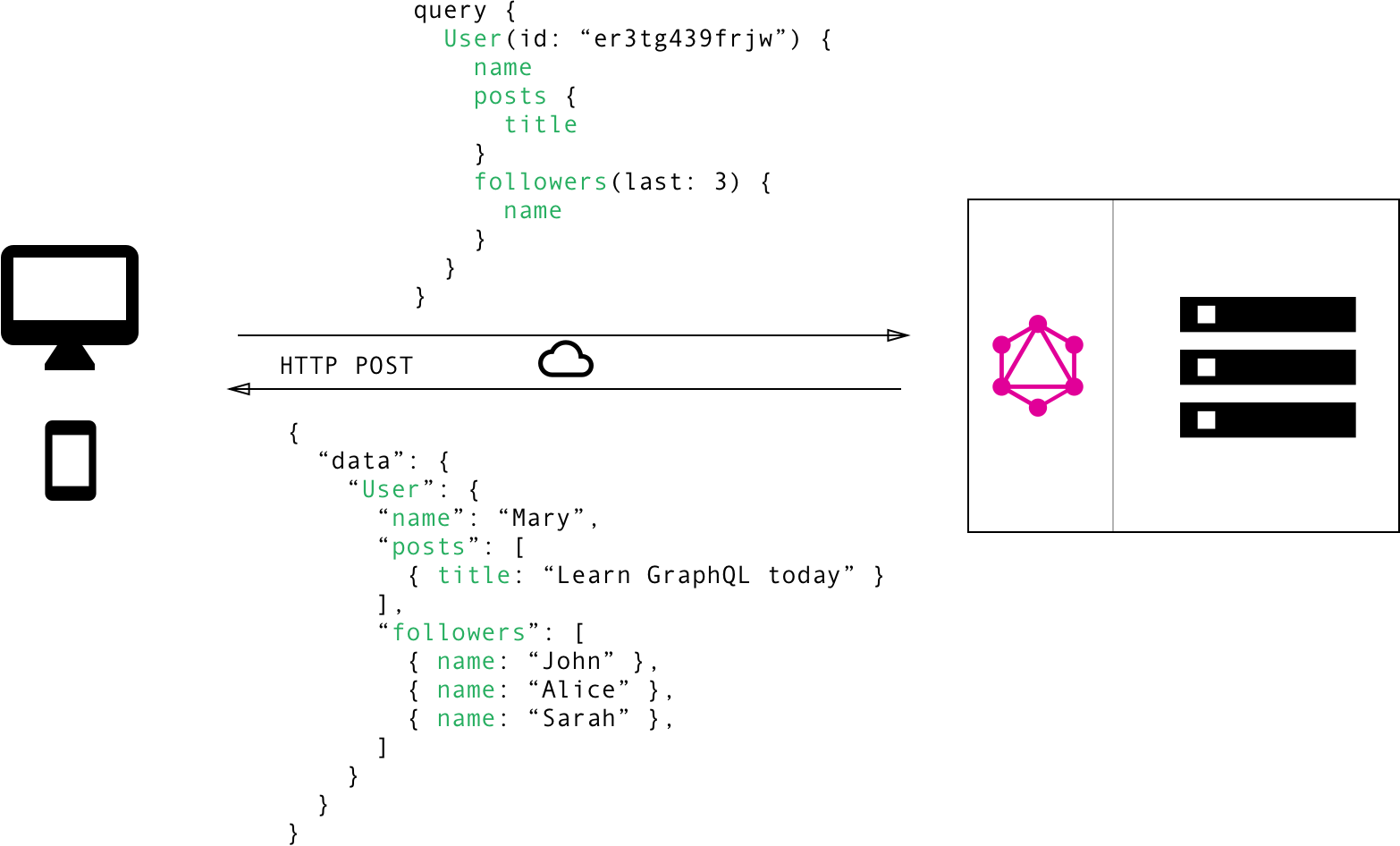


Рисунок 2.7 - Загальний алгоритм роботи додатку

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

На рис. 2.8 зображено діаграму активностей.

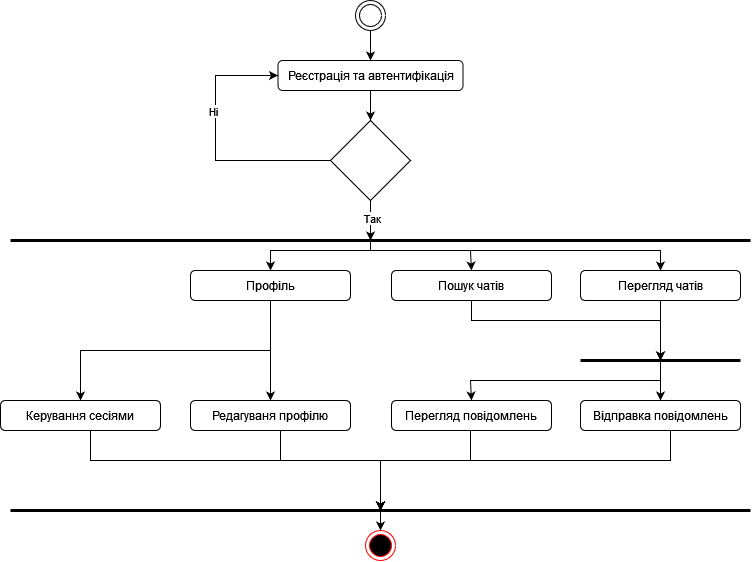


Рисунок 2.8 – Діаграма активностей системи

Опис отриманої діаграми:

1. Автентифікація (так – перехід до використання додатку, ні – перехід на сторінку входу).
2. Профіль

* Редагування профілю
* Керування сесіями

1. Пошук чатів, перегляд чатів:

* перегляд повідомлень;
* відправка повідомлень.

1. Кінець роботи веб-додатку.

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

На рис. 2.9 зображено діаграму компонентів додатку.

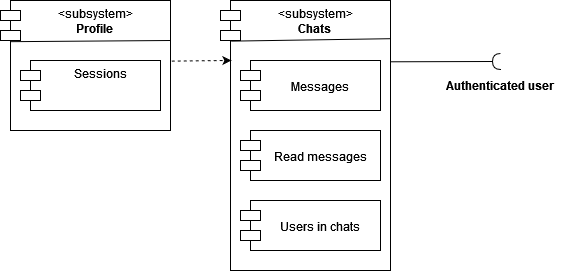


Рисунок 2.9 - Діаграма компонентів додатку

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

Будь-який початок роботи з додатком передбачає автентифікацію користувача. Тому розглянемо реалізацію даного функціоналу.

На рис. 2.11 зображено GraphQL поле Login.



Рисунок 2.11 - GraphQL поле Login

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 новолення профілю. На рис. 2.12 зображено GraphQL поле UpdateProfile.

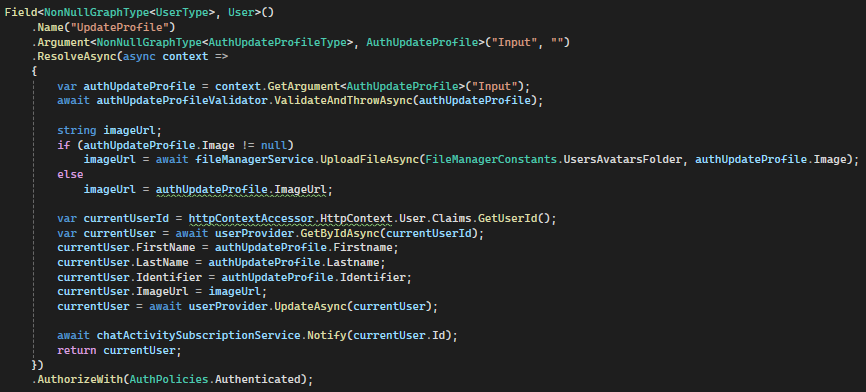


Рисунок 2.12 - GraphQL поле UpdateProfile

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. На рис. 2.13 продемонстровано GraphQL поле CreatePersonalChat.

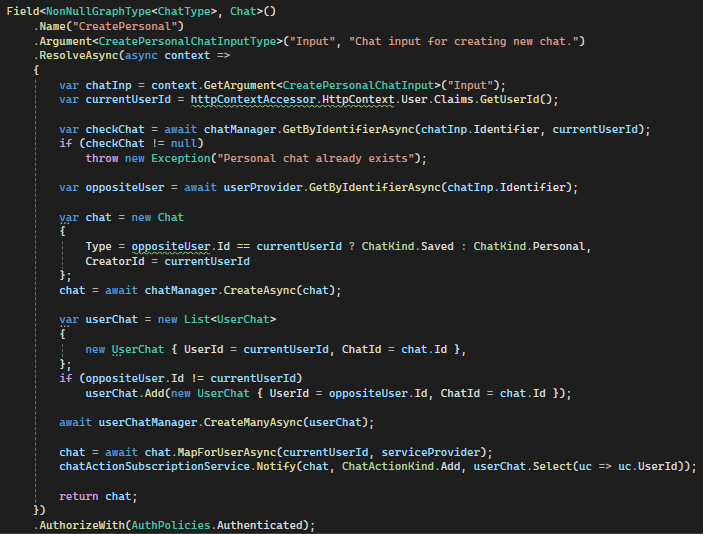


Рисунок 2.13 - GraphQL поле CreatePersonalChat

Поле СreatePersonalChat 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. На рис. 2.14 продемонстровано GraphQL поле SendMessage.





Рисунок 2.14 - 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 цьому розділі було побудовано діаграму варіантів використання для наочного зображення відносин між акторами та прецедентами в системі. Сформовано вимоги до системи, обґрунтовано та описано використання патернів при розробці застосунку. Спроектовано базу даних та описано поля і їх призначення.

Описано застосування архітектури Single-Page Application та обґрунтовано використання підходу RESTful API. Продемонстровано загальний алгоритм роботи і створено наступні діаграми для наочності: діаграма активностей системи, діаграма послідовності для оплати та діаграма компонентів системи.

Також було продемонстровано реалізацію програмного комплексу, наведено фрагменти коду, що відповідають за авторизацію, створення чату та відправку повідомлень, з поясненням принципу їх 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 діаграма, що відображає конфігурацію вузлів виконання та компонентів, які на них розміщуються. Це різновид структурної діаграми, яка використовується для моделювання фізичних аспектів об’єктно-орієнтованої системи. Їх часто застосовують для відображення статичної структури розгортання системи, зокрема топології апаратного забезпечення.

На рис. 3.1 зображено діаграму розгортання.

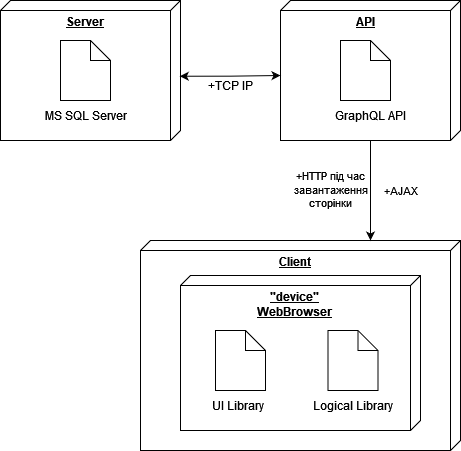


Рисунок 3.1 - Діаграма розгортання додатку

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.

На рис. 3.2 зображено сторінку входу.

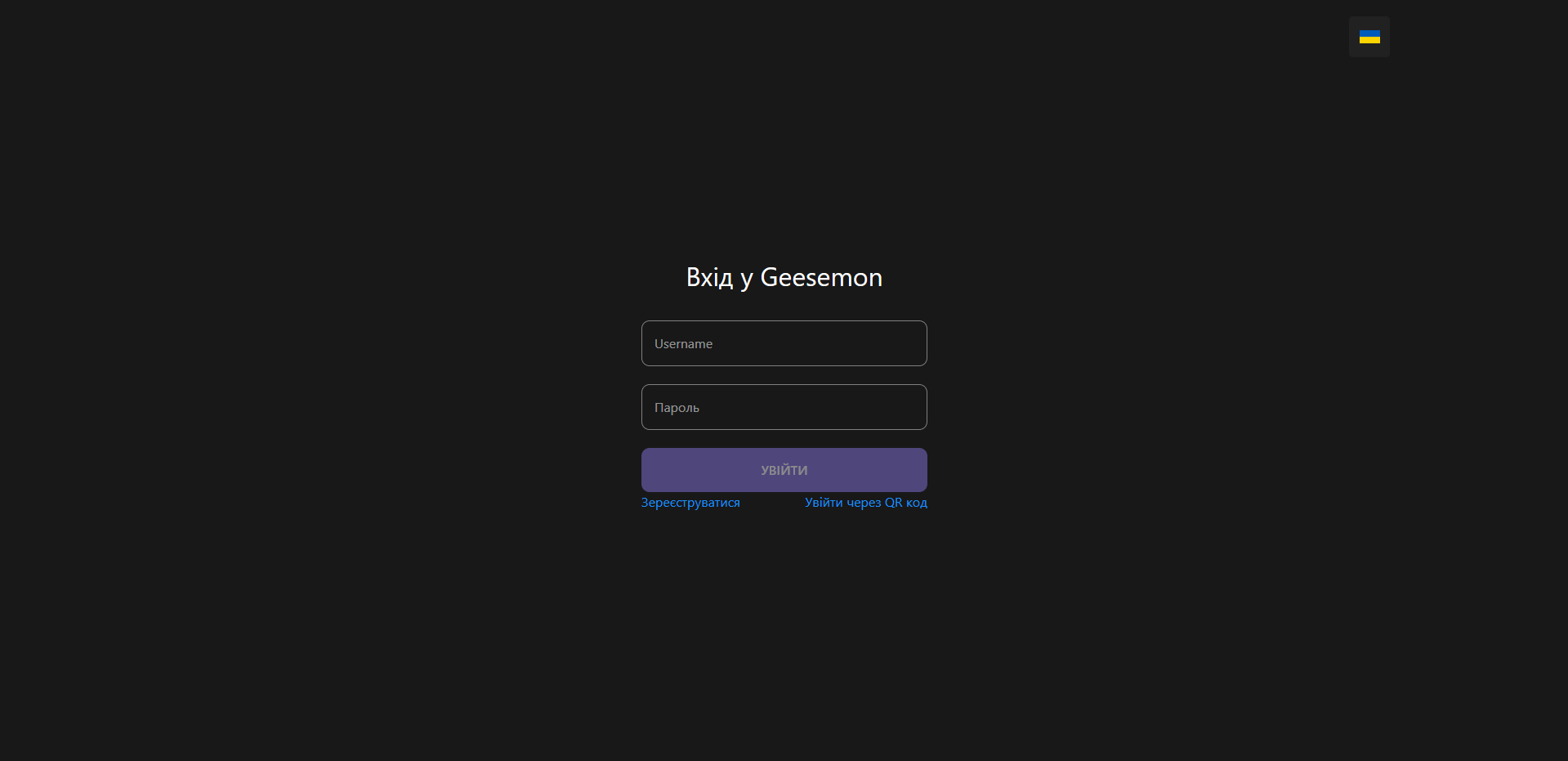


Рисунок 3.2 – Сторінку входу

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

На рис. 3.3 зображено сторінку реєстрації.

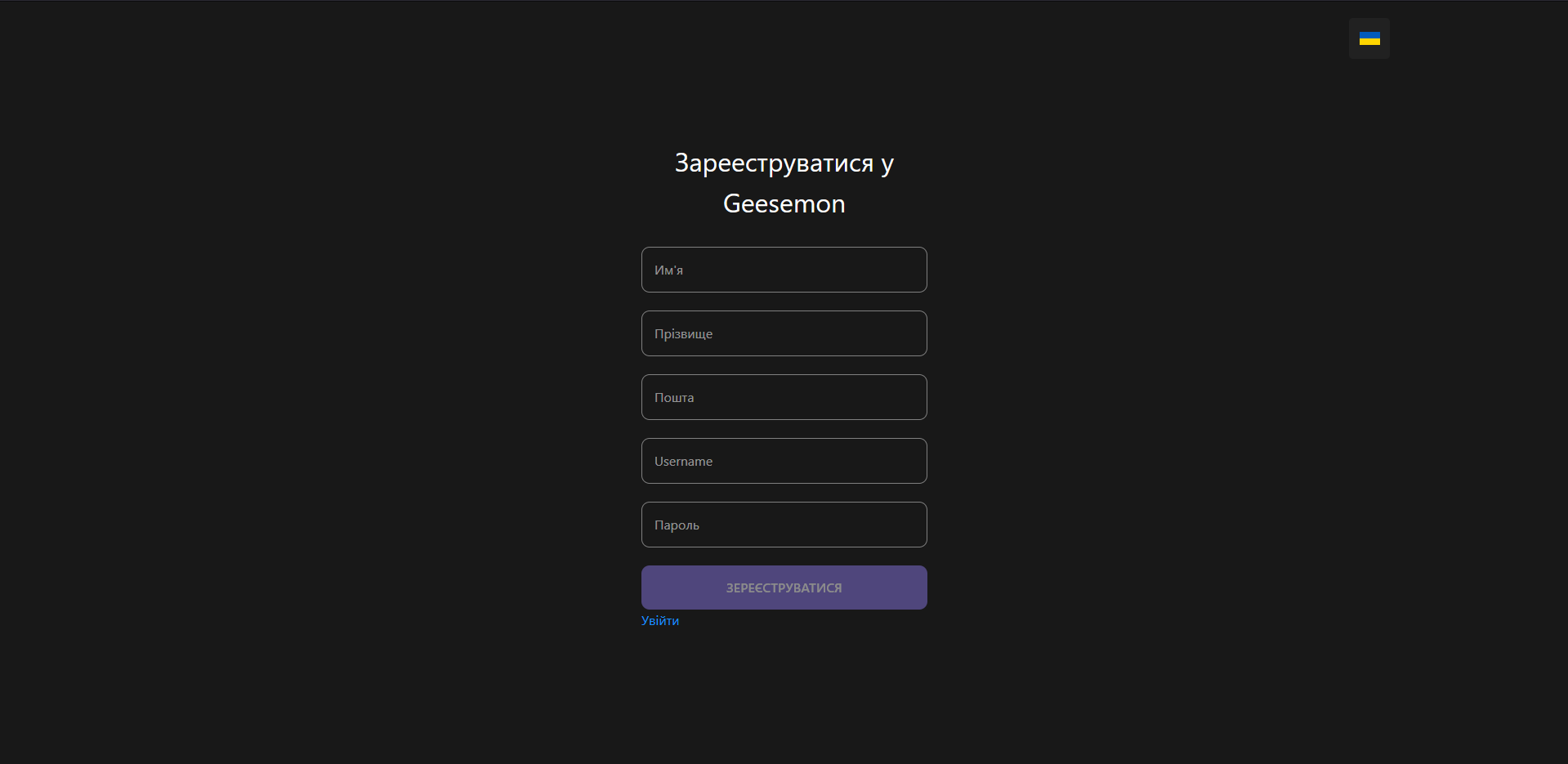


Рисунок 3.3 - Сторінка реєстрації

Після успішної рестрації користувач має можливість повноцінно користуватися додатком.

На рис. 3.4 зображено сторінку з переліком чатів.

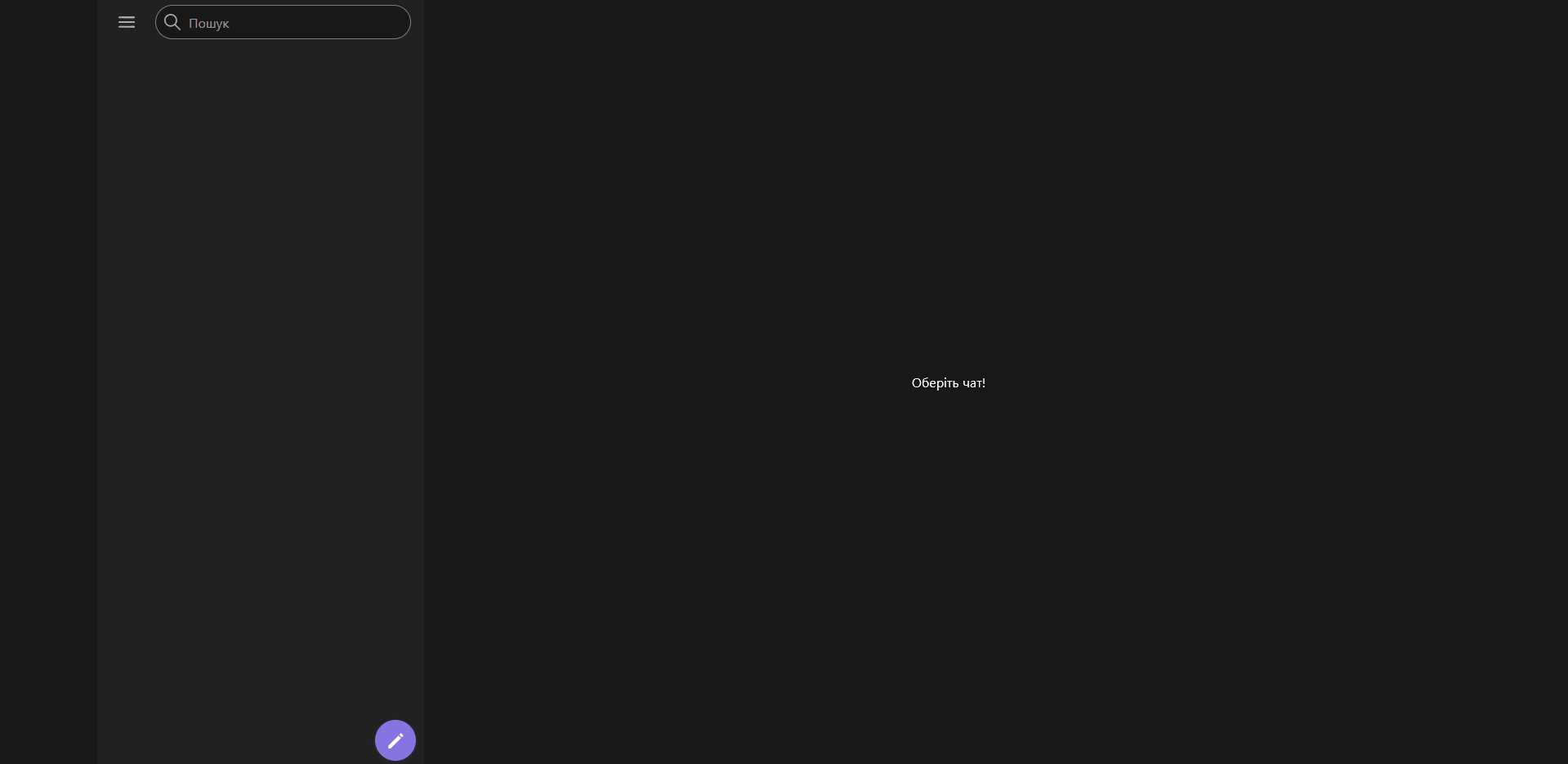


Рисунок 3.4 – Сторінка з переліком чатів

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». На рис. 3.5 зображений даних процес.

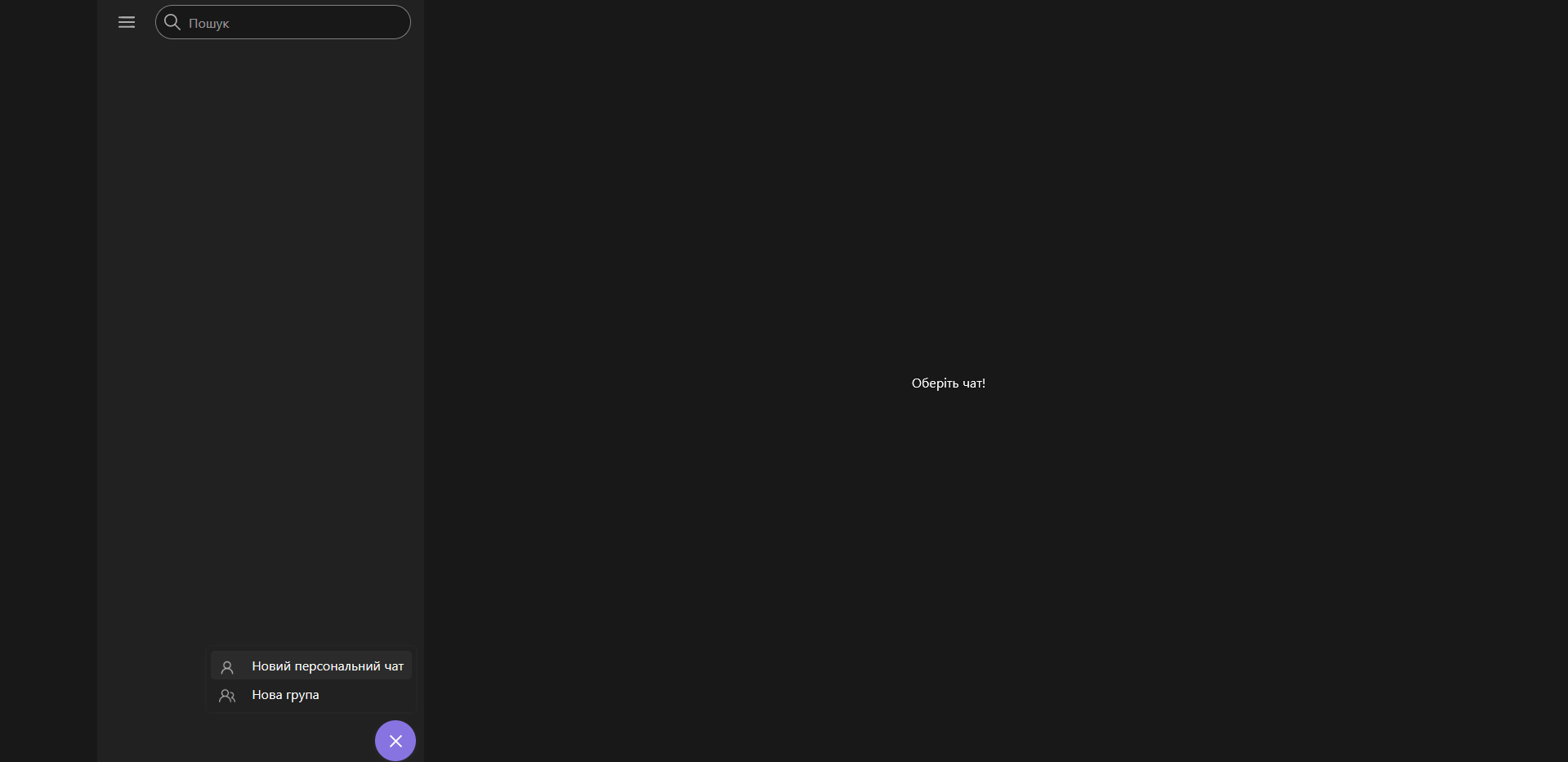


Рисунок 3.5 – Список опцій для кнопки «Написати»

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

На рис. 3.6 зображено список користувачів для створення персонального чату.

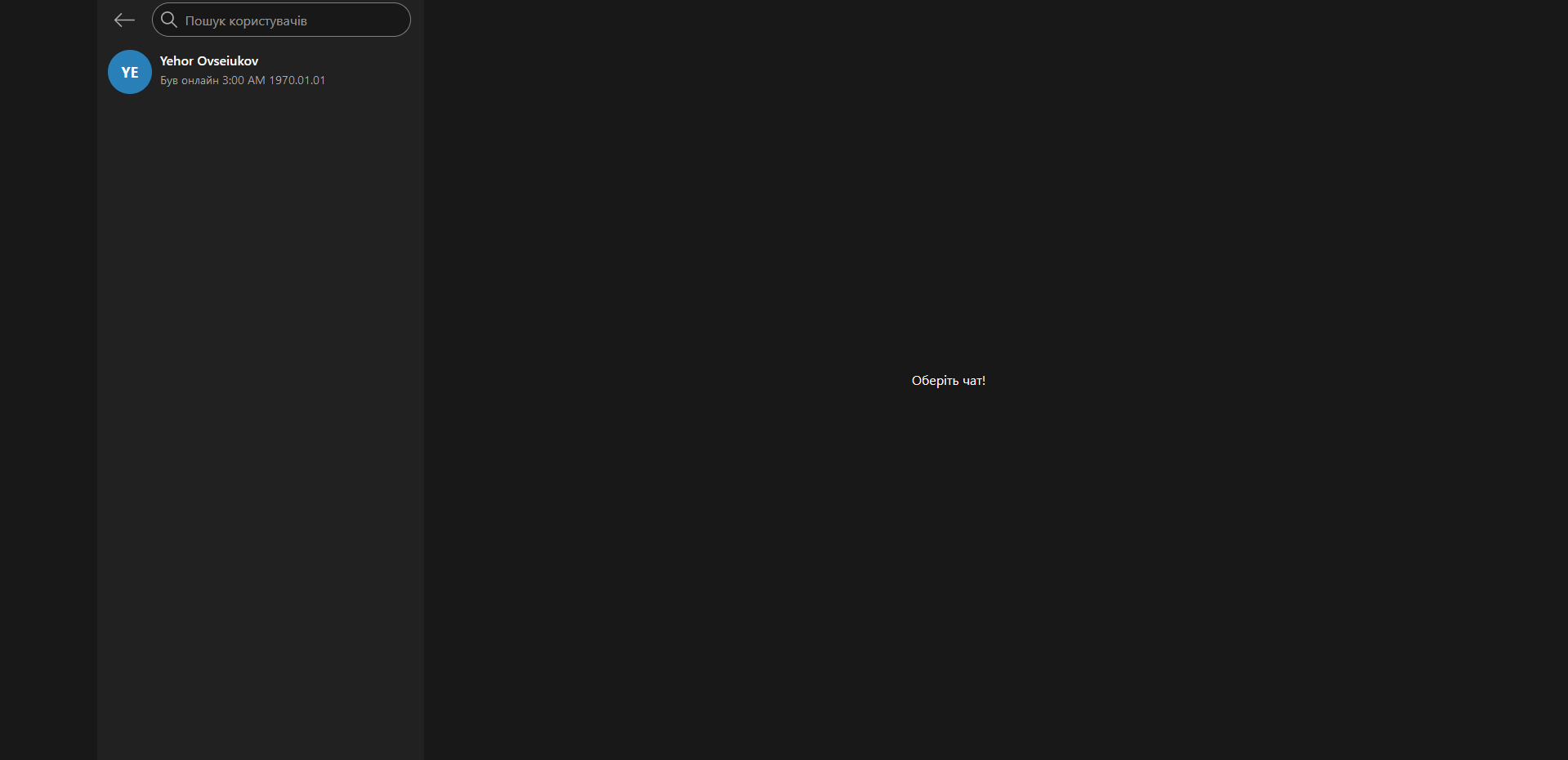


Рисунок 3.6 - Список користувачів для персонального чату

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

На рис. 3.7 зображено тимчасовий чат.

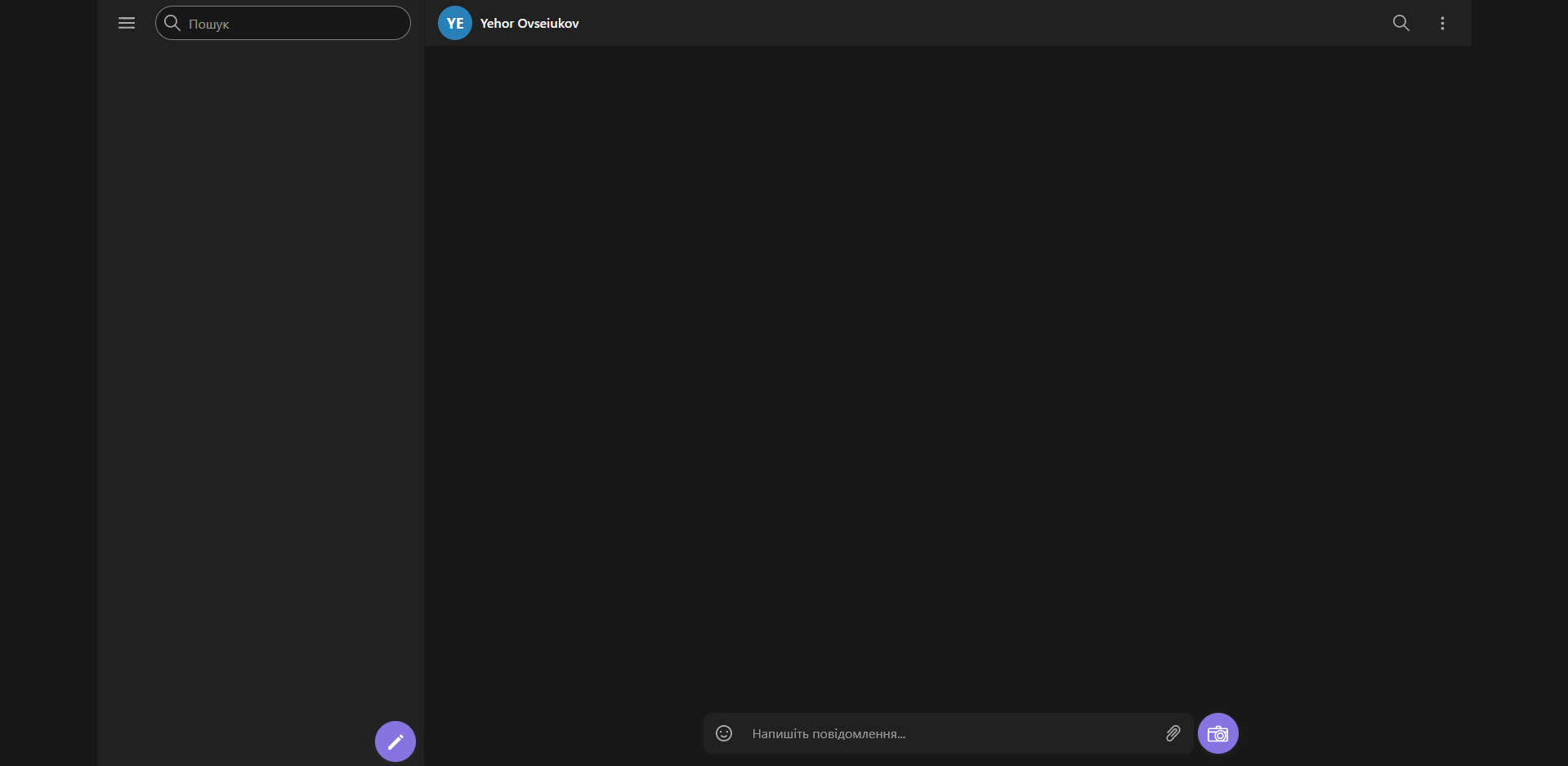


Рисунок 3.7 – Тимчасовий персональний чат

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

На рис. 3.8 зображено новий персональний створений чат і відправка першого повідомлення.

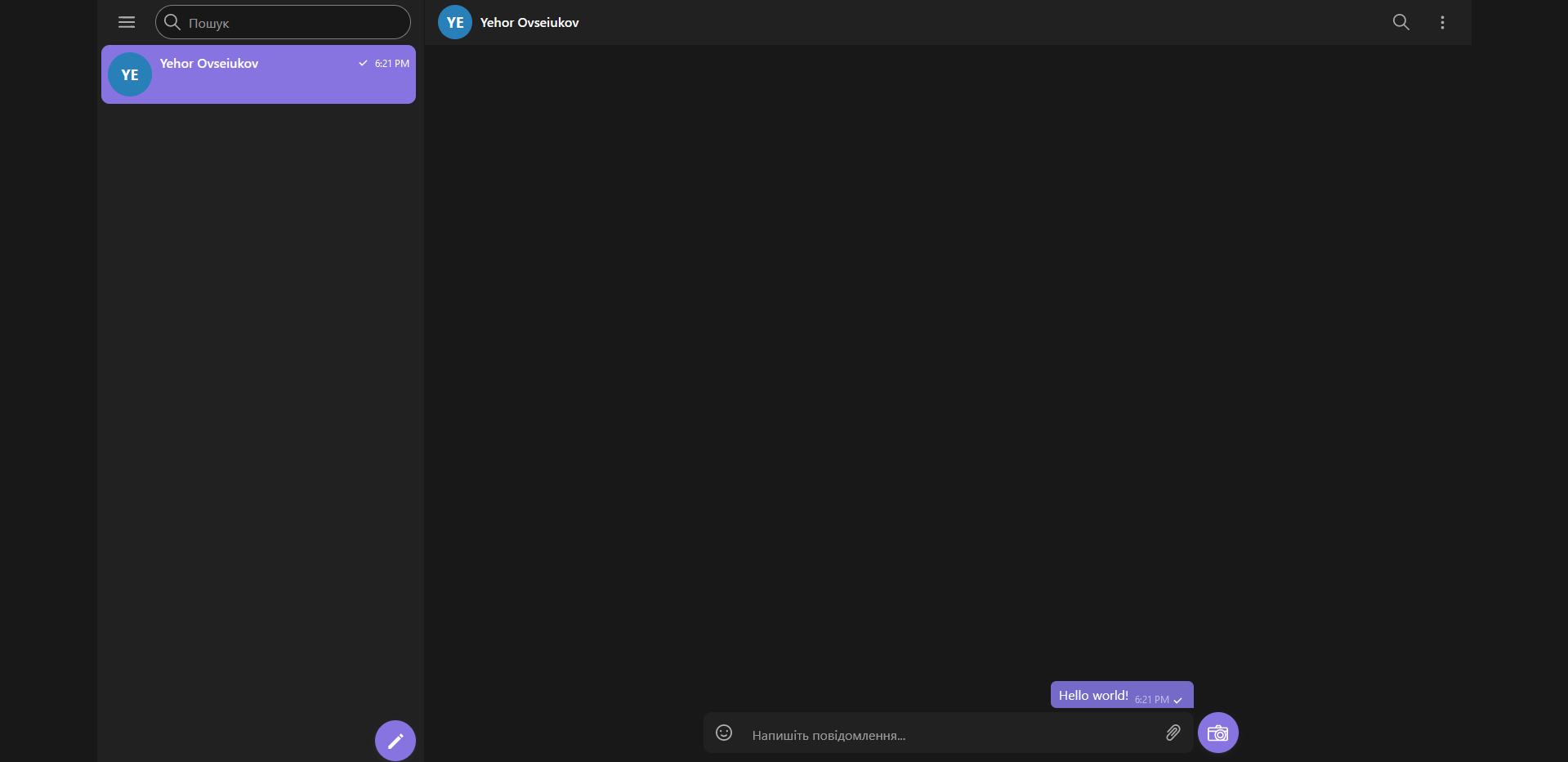


Рисунок 3.8 - Новий персональний створений чат і відправка першого повідомлення

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

На рис. 3.9 зображено список користувачів, яких можна додати до чату.

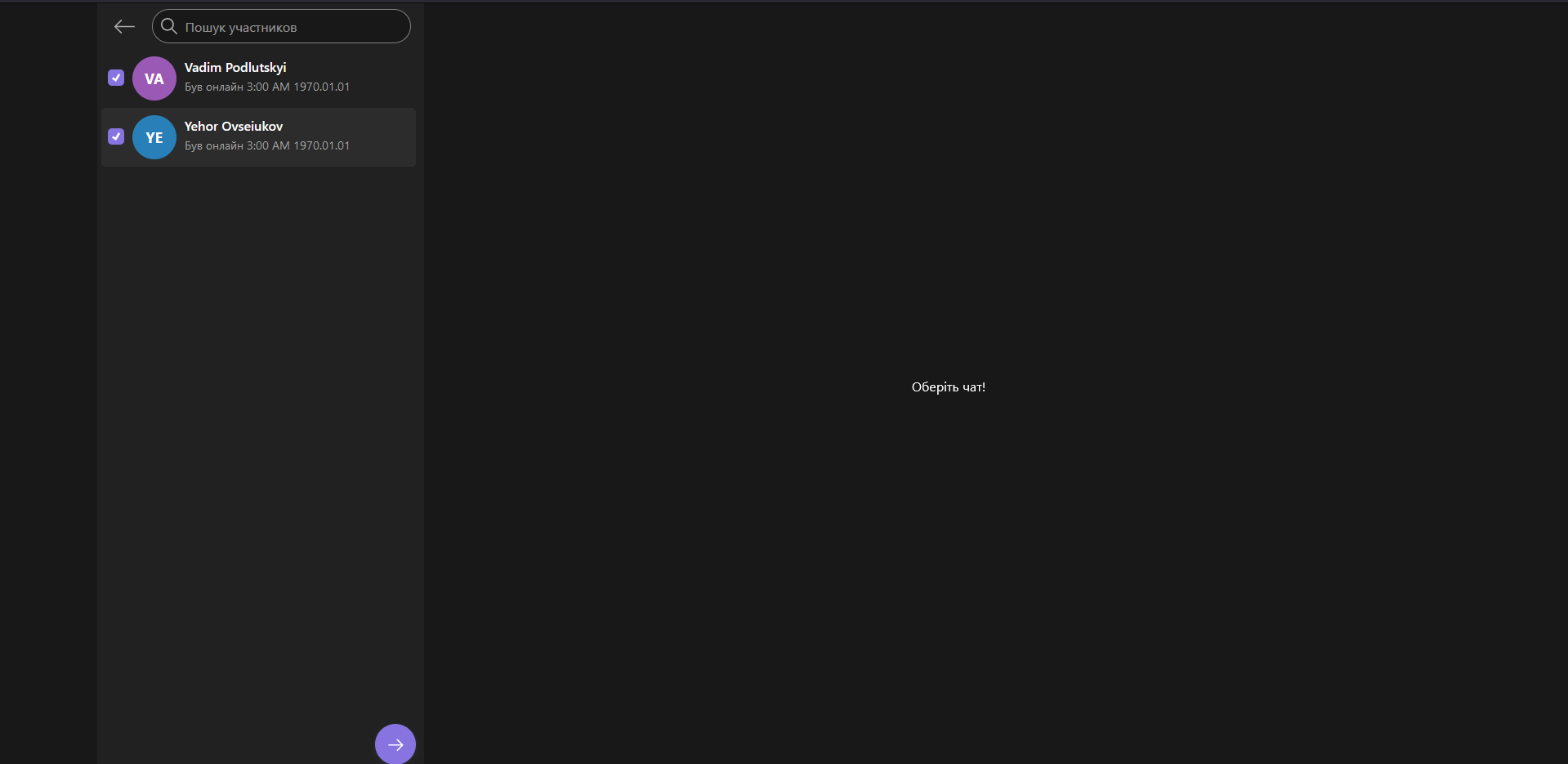


Рисунок 3.9 - Список користувачів, яких можна додати до чату

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

На рис. 3.10 зображено форму створення чату.

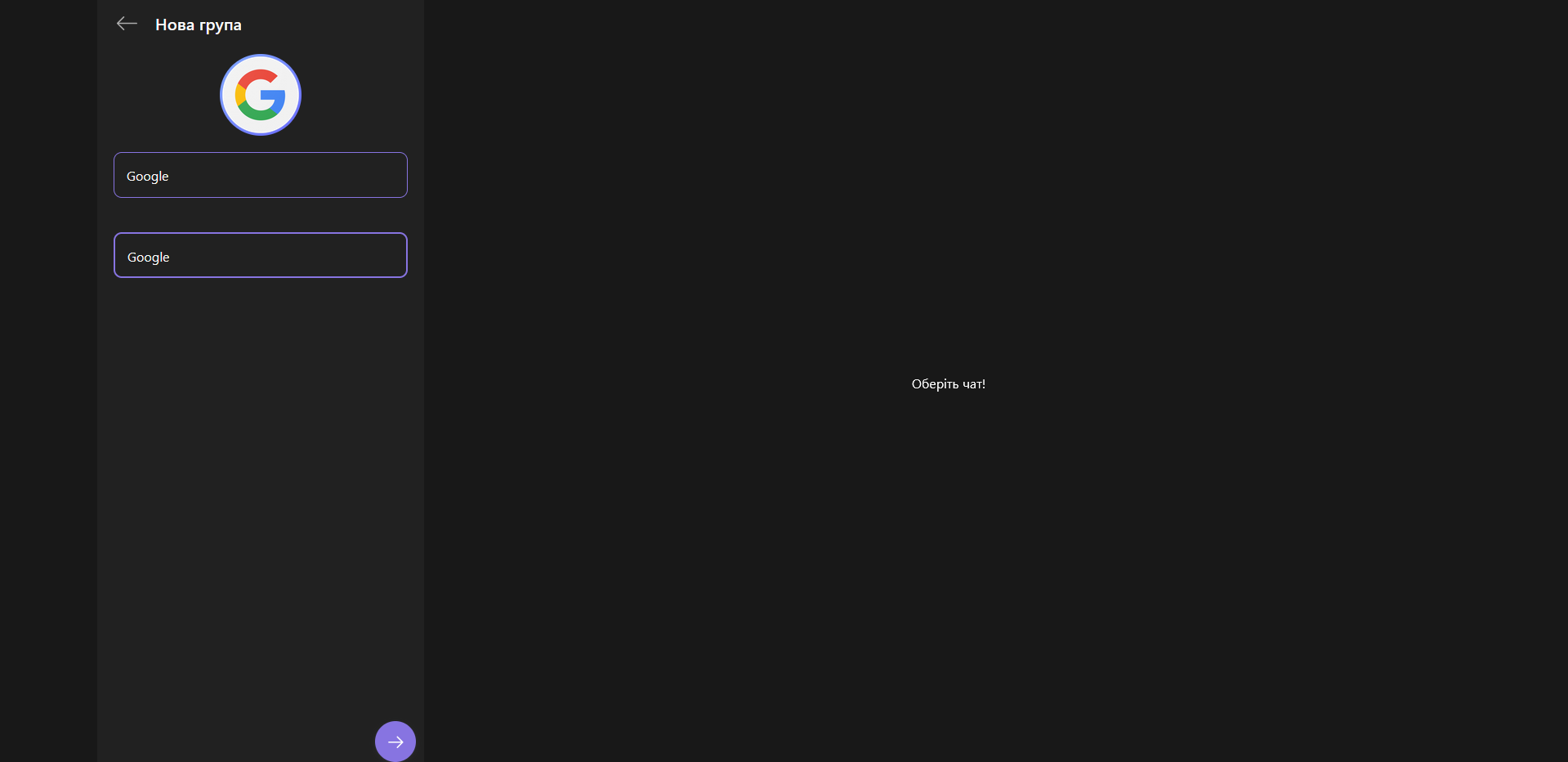


Рисунок 3.10 - Форма створення чату

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

На рис. 3.11 зображено новий чат.

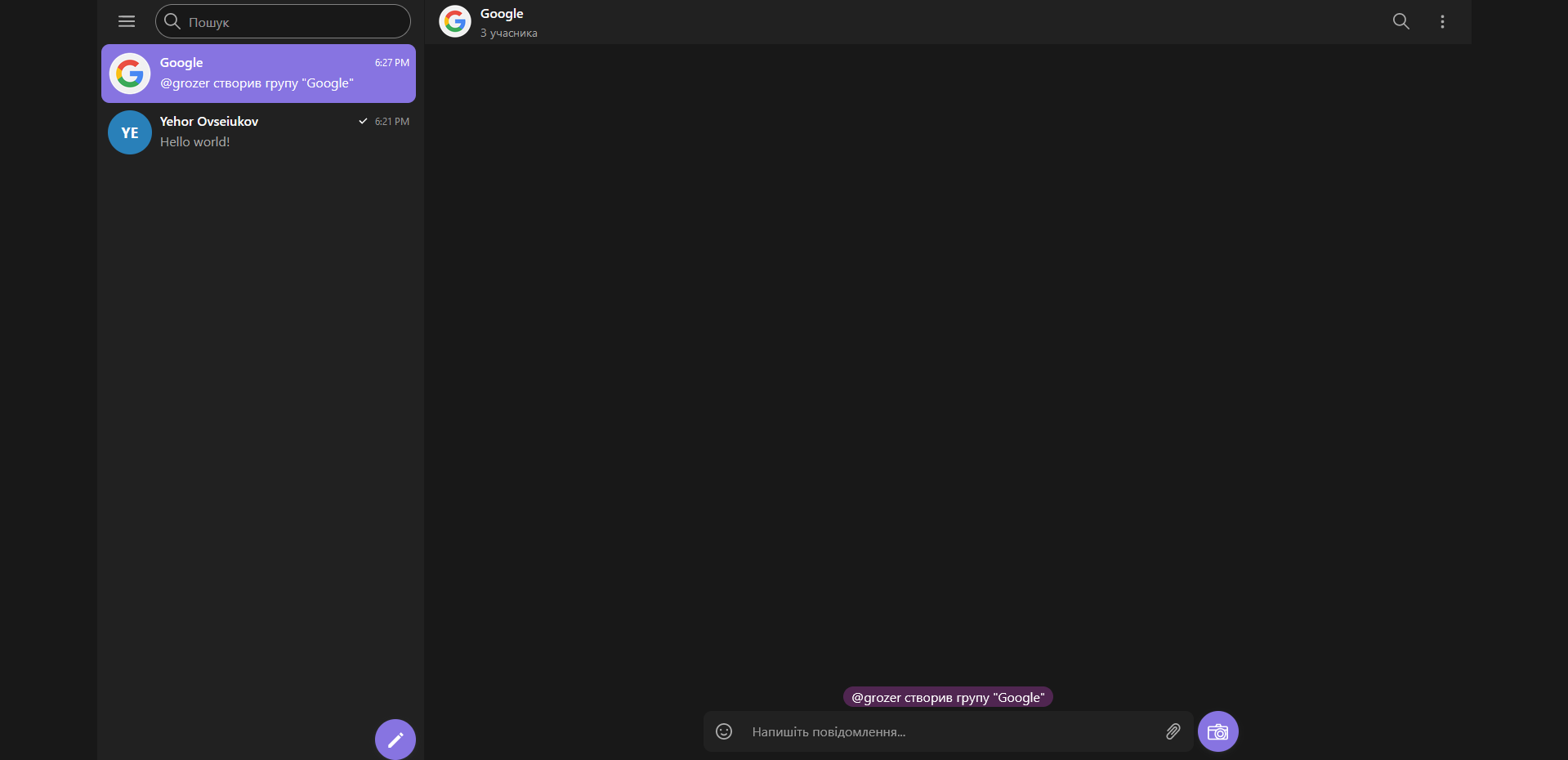


Рисунок 3.11 - Новий чат

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.

На рис. 3.12 зображено контекстне меню повідомлення.

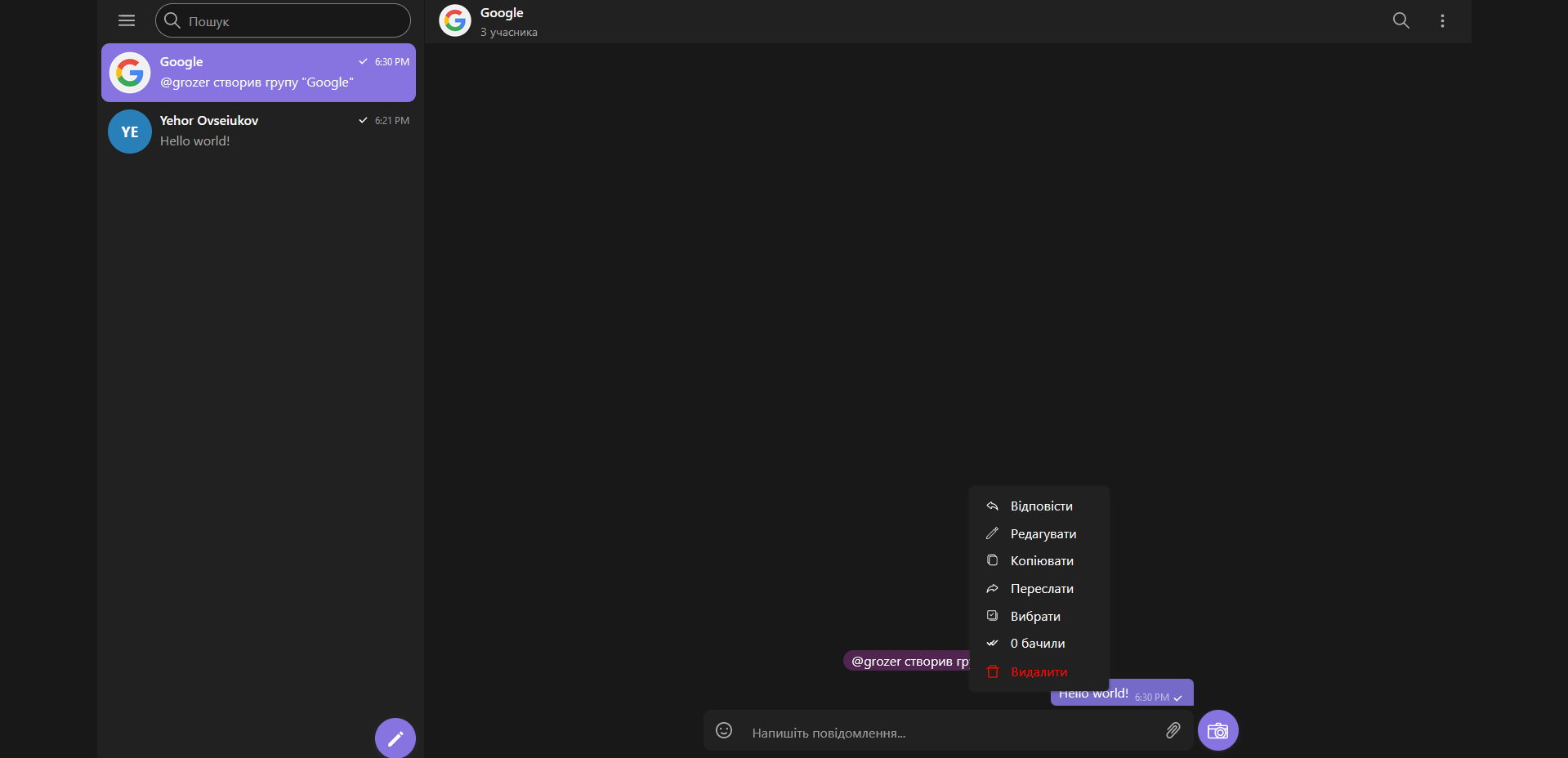


Рисунок 3.12 – Контекстне меню повідомлення

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

На рис. 3.13-3.14 зображено процес «Відповісти» на повідомлення.

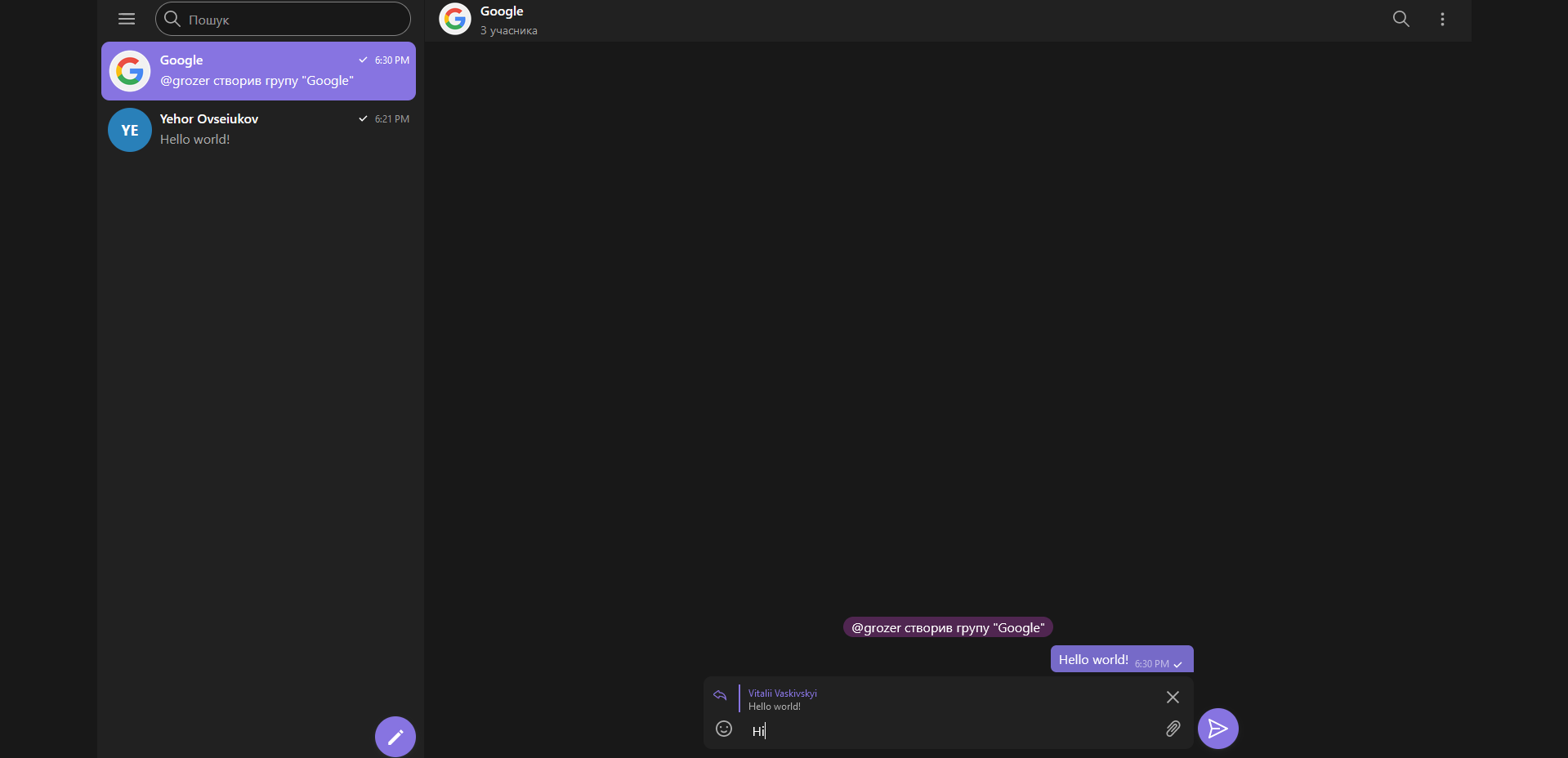


Рисунок 3.13 – Відповідь на повідомлення

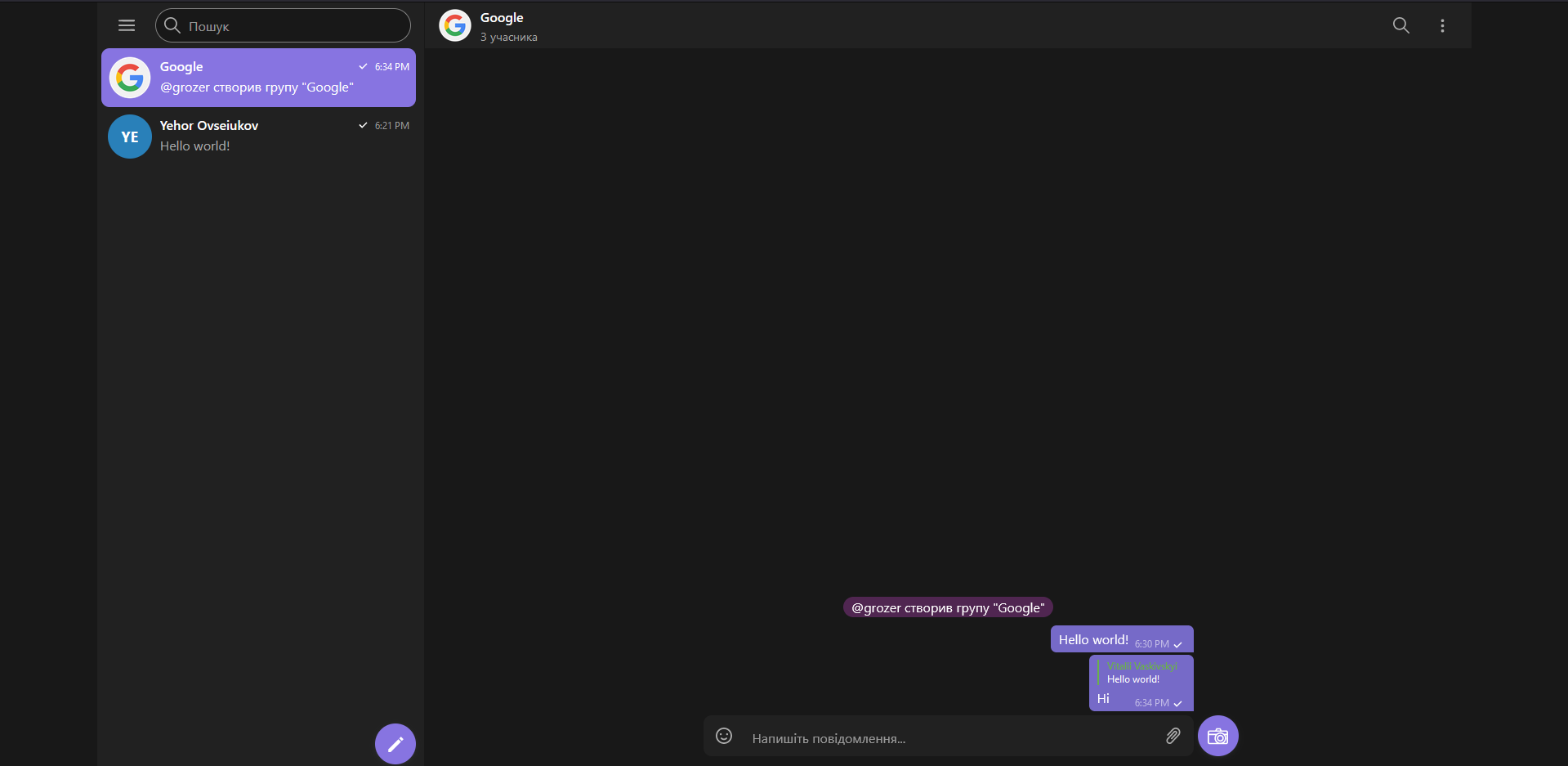


Рисунок 3.14 – Відправлене повідомлення, яке є відповідю на інше повідомлення

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

Це продемонстровано на рис. 3.15.

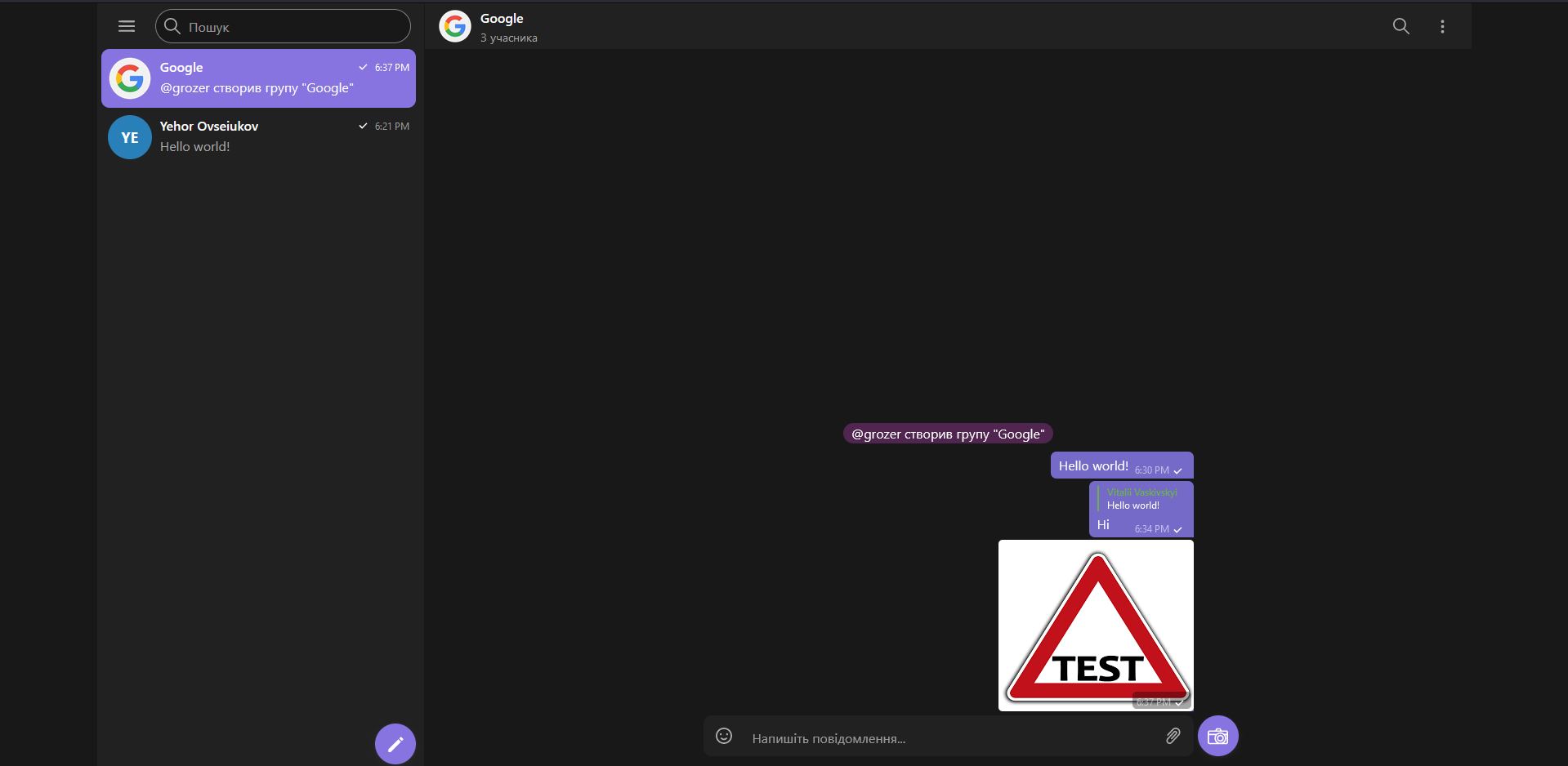


Рисунок 3.15 – Відправка файлу

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

На рис. 3.16 зображено модальне вінко записування відео-повідомлення.

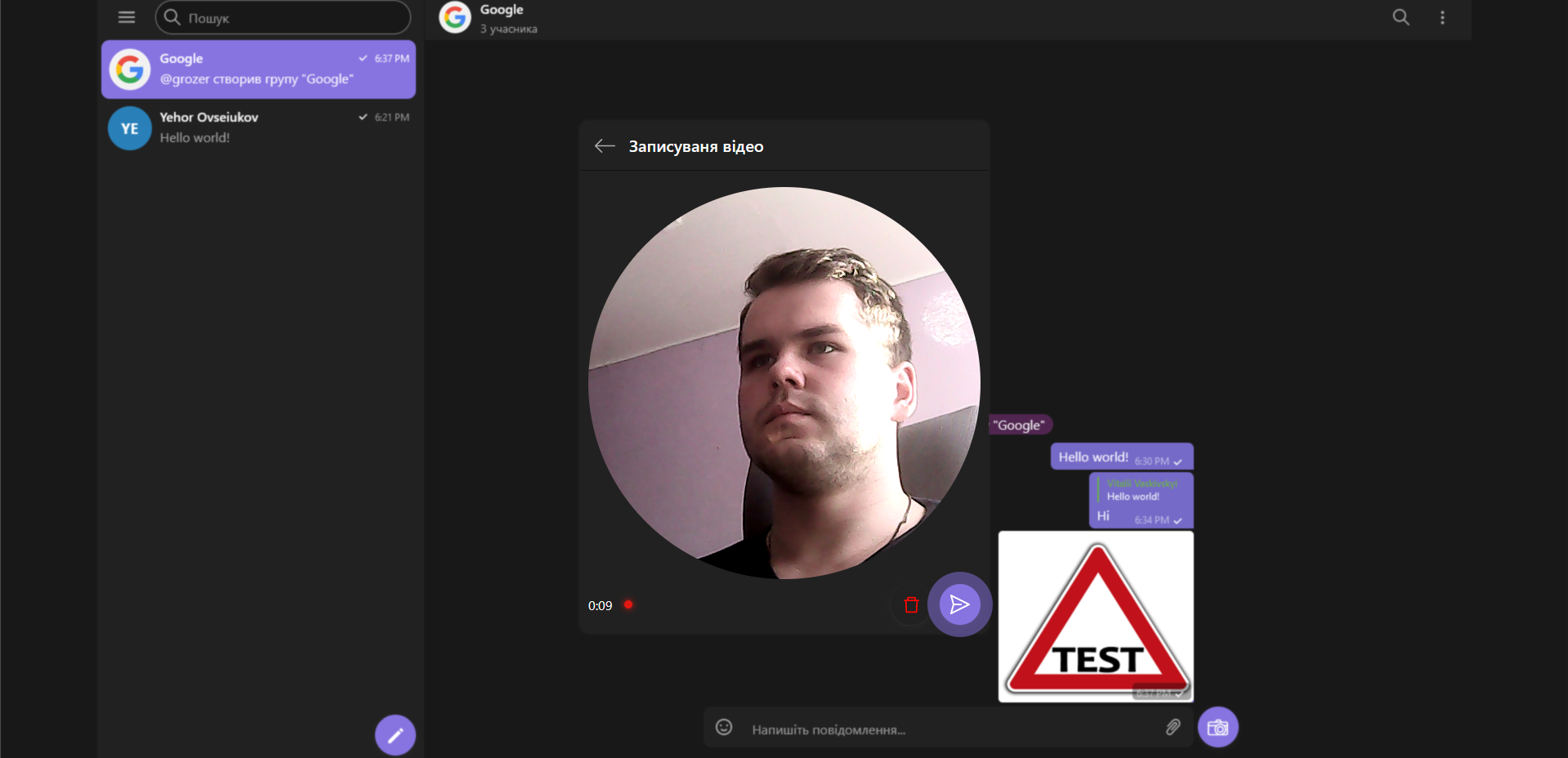


Рисунок 3.16 - Модальне вінко записування відео-повідомлення

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

На рис. 3.17 зображено перегляд надісланого відео-повідомлення.

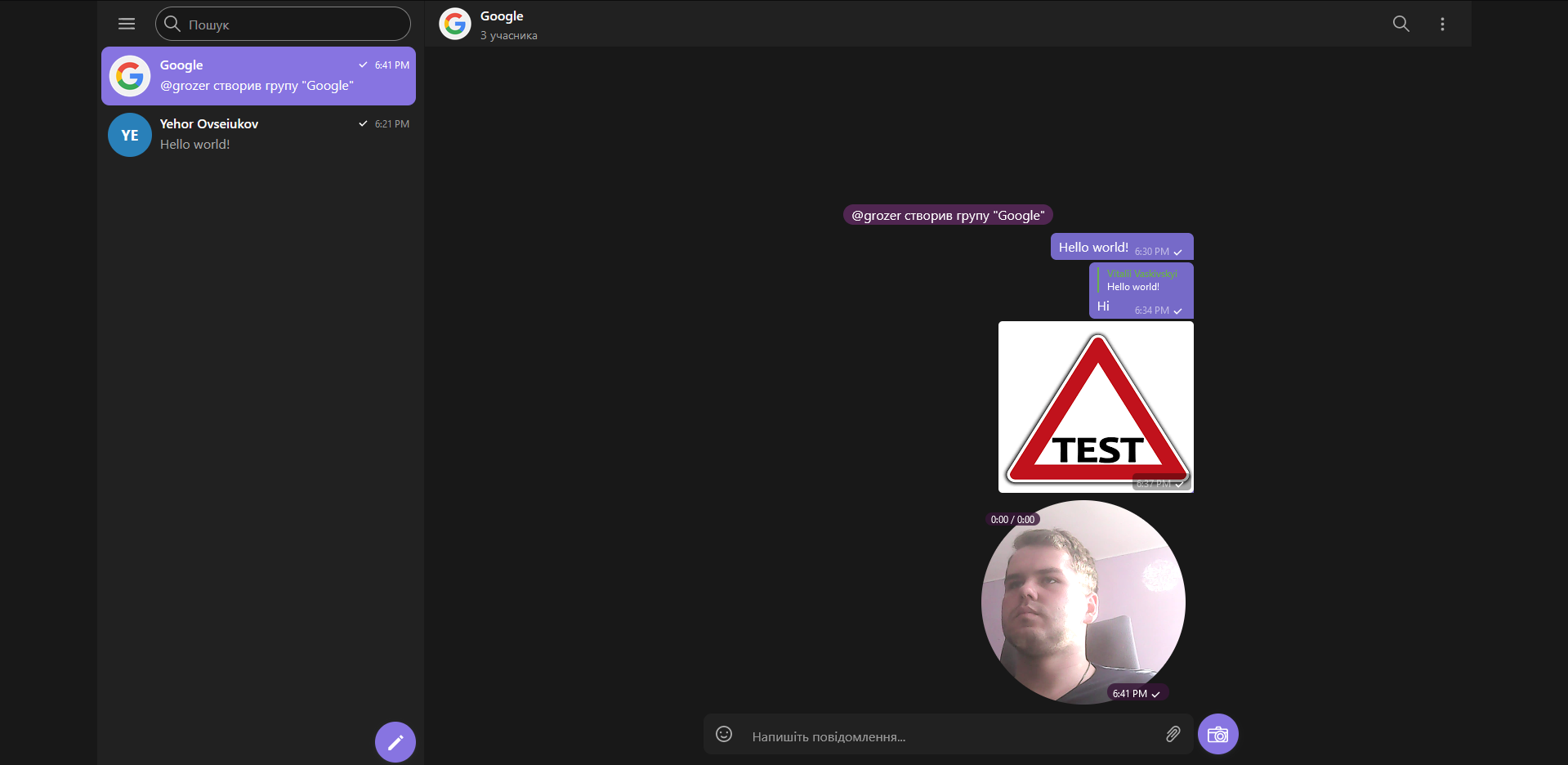


Рисунок 3.17 - Перегляд надісланого відео-повідомлення

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.

На рис. 3.18 зображено запис голосового повідомлення.

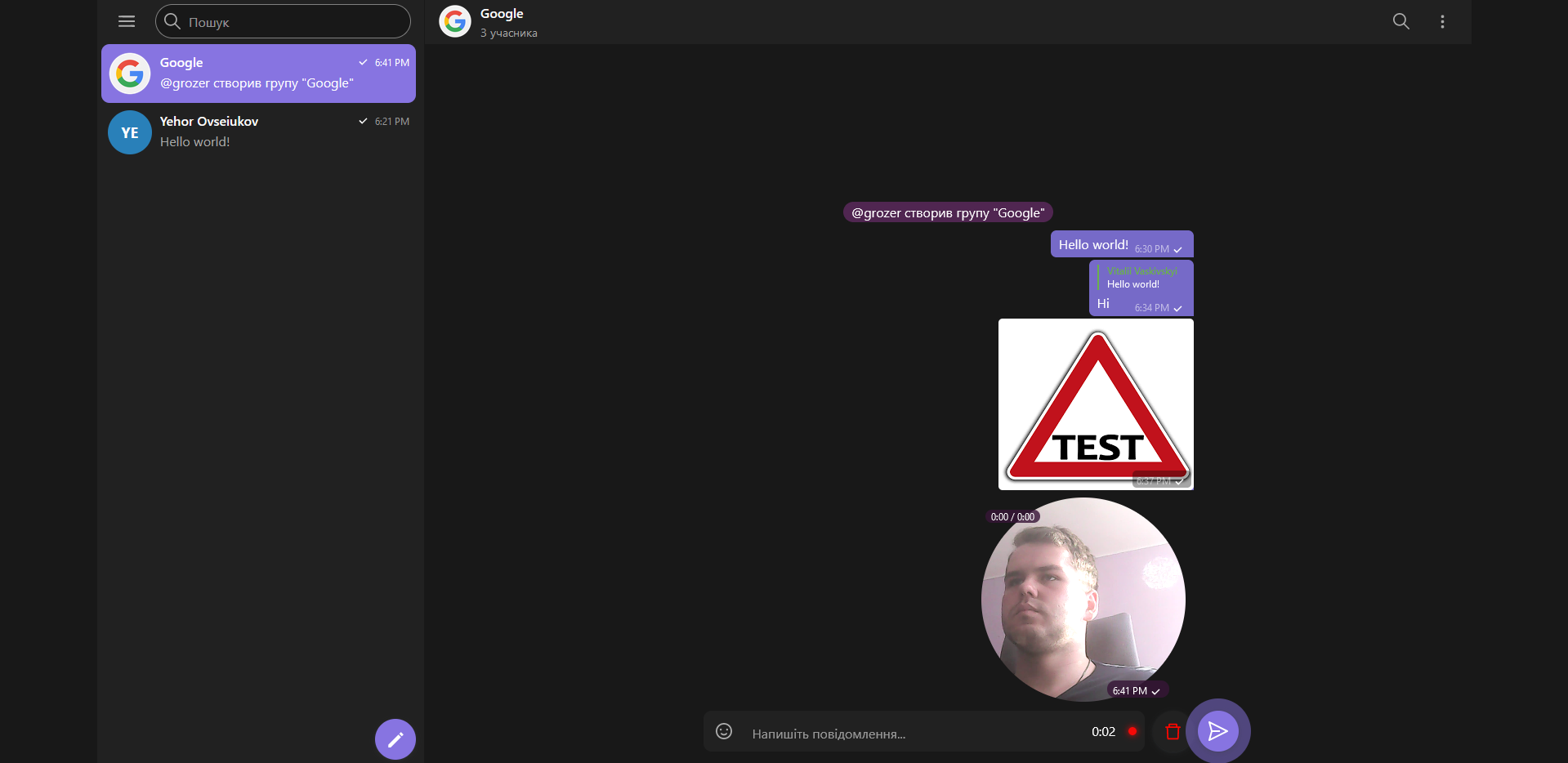


Рисунок 3.18 - Запис голосового повідомлення

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

На рис. 3.19 зображено налаштування користувача.

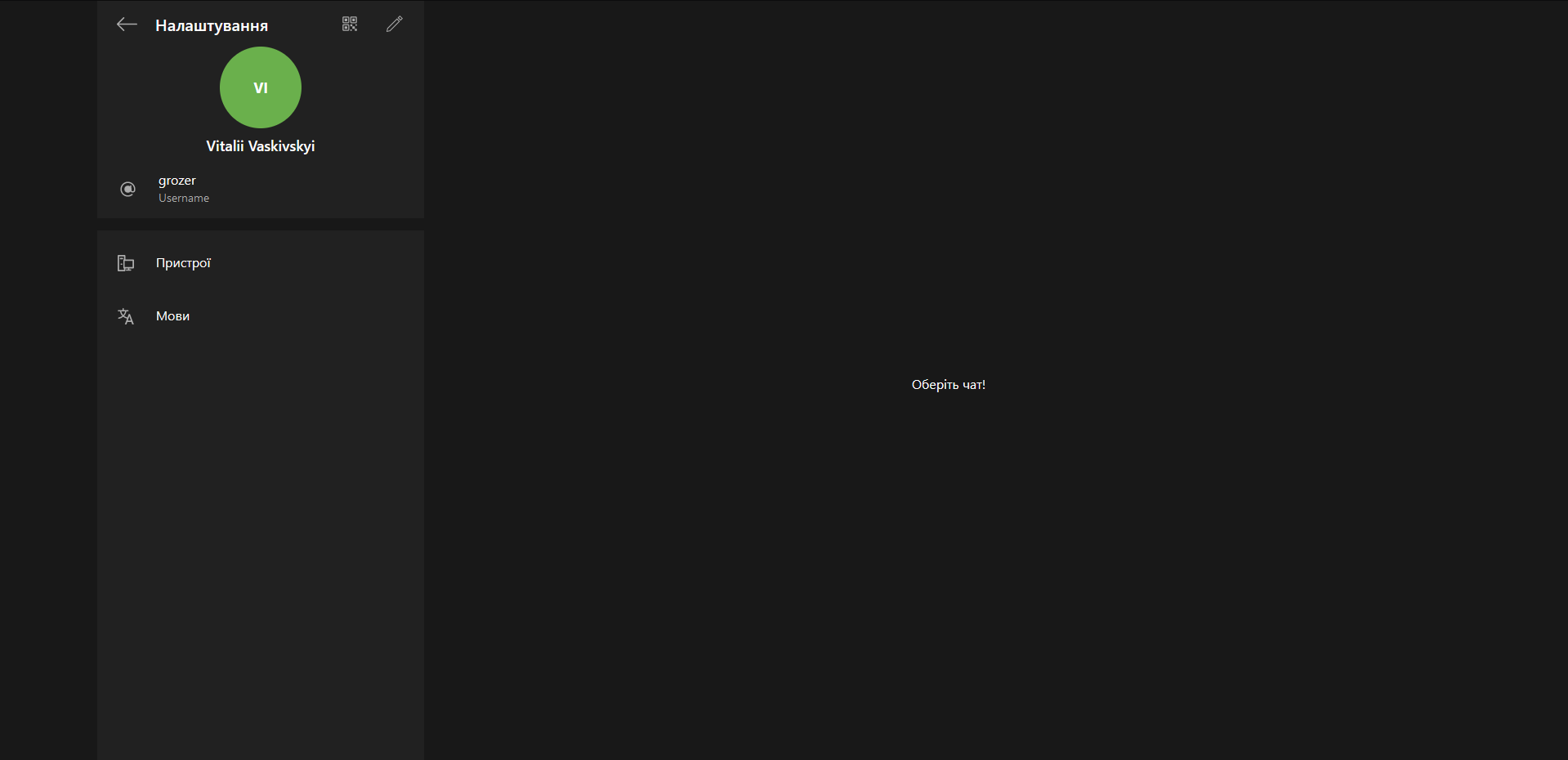


Рисунок 3.19 - Налаштування користувача

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.

На рис. 3.20 зображено форма редагування профілю.

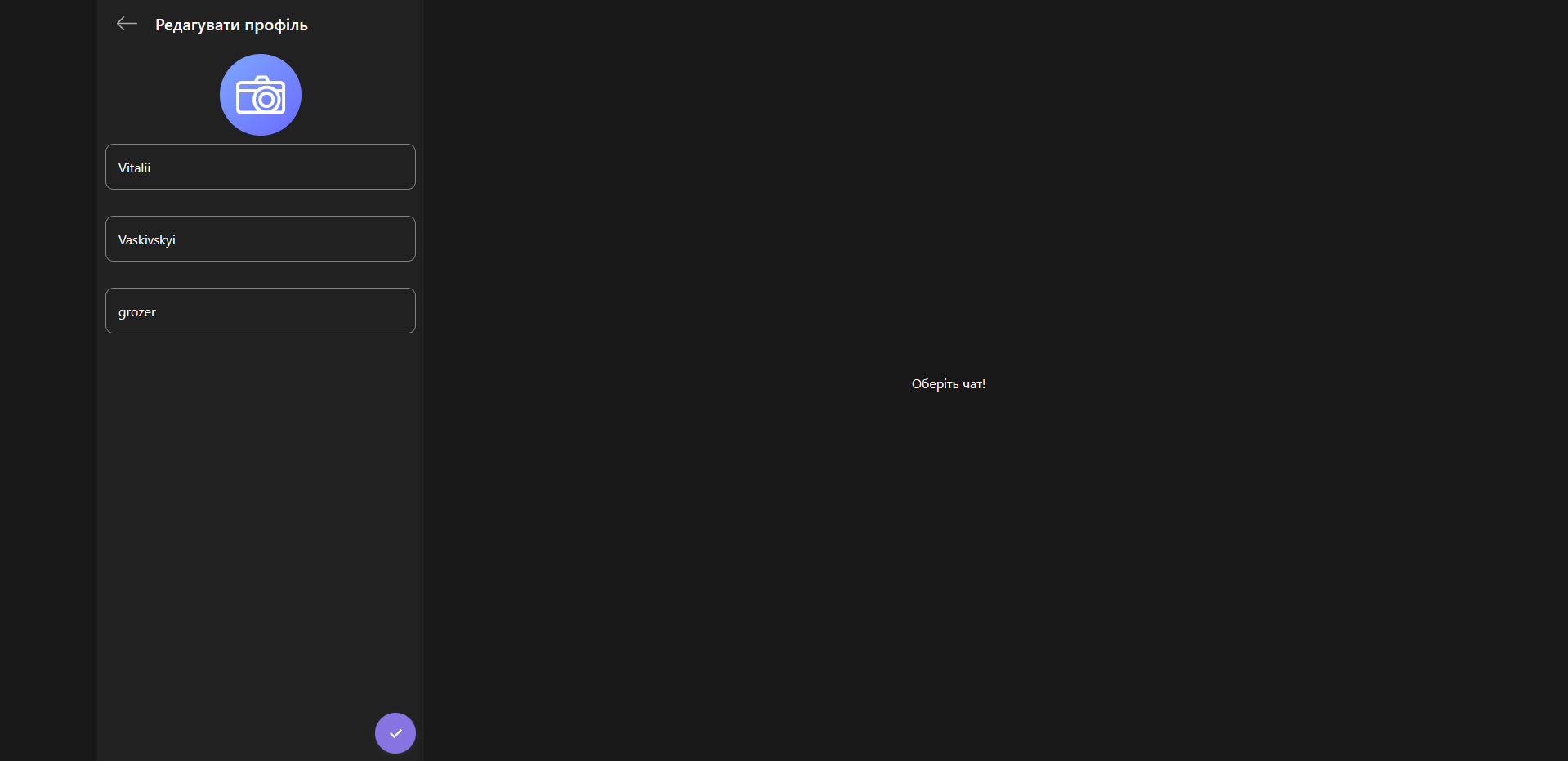


Рисунок 3.20 - Форма редагування профілю

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.

На рис. 3.21 зображено список пристроїв поточного користувача.

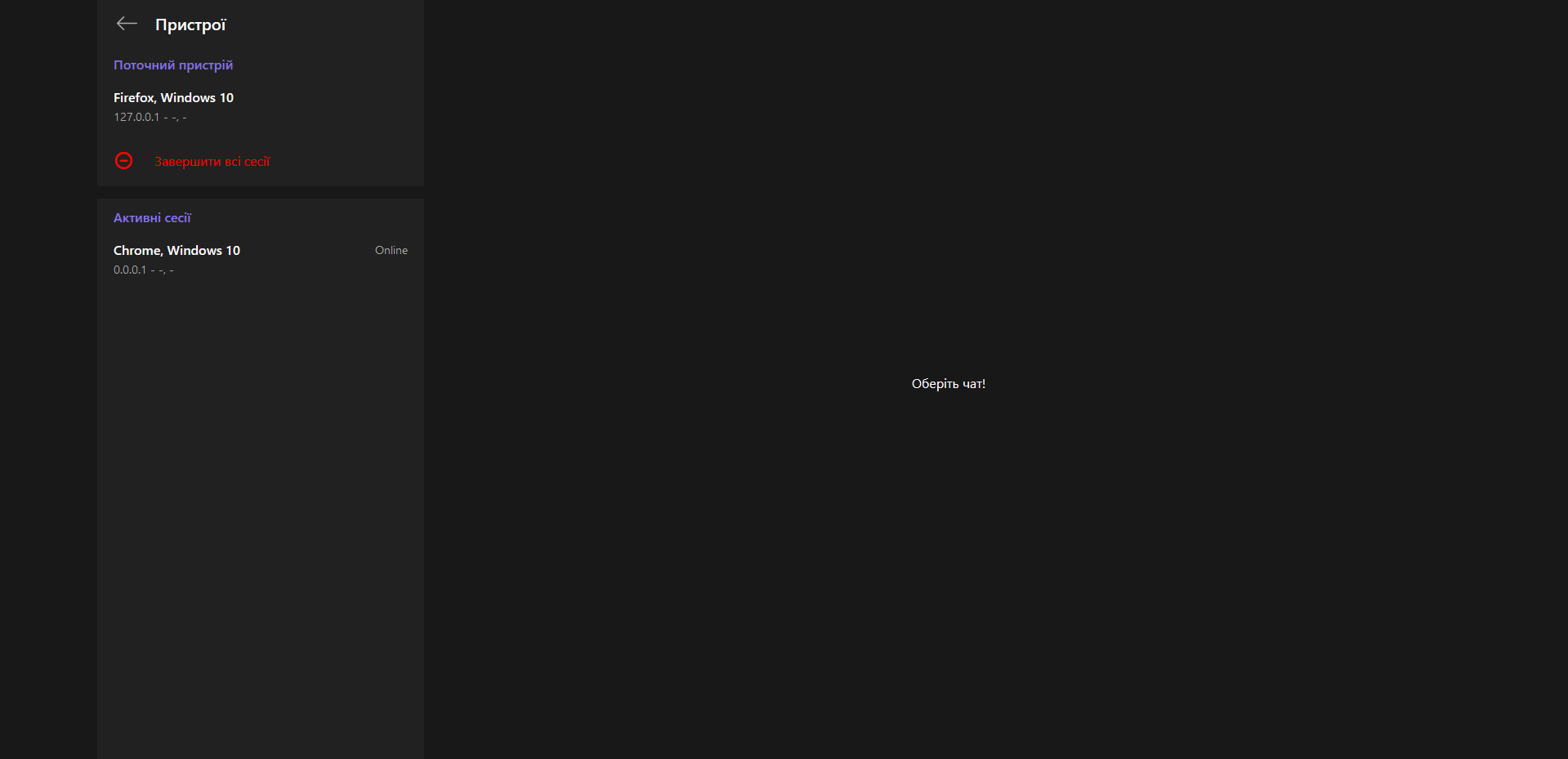


Рисунок 3.21 - Список пристроїв поточного користувача

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 тестовий артефакт, який передбачає виконання певної кількості дій та/або умов для перевірки конкретної функціональності програмної системи, що розробляється. Представимо створені тестові випадки у вигляді таблиці.

Таблиця 3.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case Description | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail |
| *1.* | *2.* | *3.* | *4.* | *5.* | *6.* |
| Перевірка авторизації з дійсними даними | 1. Перейти на сторінку http://127.0.0.1:3000/login 2. Введіть email або логін 3. Введіть пароль 4. Натисніть "Увійти" | email = test@gmail.com Пароль = 12345678 | Користувач повинен увійти в систему | Як і очікувалося | Pass |
| Перевірка авторизації з недійсними даними | 1. Перейти на сторінку http://127.0.0.1:3000/login 2. Введіть email або логін 3. Введіть пароль 4. Натисніть "Увійти" | email = test@gmail.com Пароль = 12345679 | Користувач не повинен входити в програму | Як і очікувалося | Pass |
| Перевірка реєстрації з дійсними даними | 1. Перейти на сайт http://127.0.0.1:3000/register 2. Введіть логін 3. Введіть email 4. Введіть пароль 5. Підтвердіть пароль 6. Натисніть "Зареєструватися" | Логін = test email = test@gmail.com Пароль = 12345678 Підтвердження пароля = 12345678 | Повинен створитися акаунт і користувач повинен увійти в систему | Як і очікувалося | Pass |

Продовження табл. 3.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *1.* | *2.* | *3.* | *4.* | *5.* | *6.* |
| Перевірка реєстрації з недійсними даними | 1. Перейти на сайт http://127.0.0.1:3000/register 2. Введіть логін 3. Введіть email 4. Введіть пароль 5. Підтвердіть пароль 6. Натисніть "Зареєструватися" | Логін = test email = test@gmail.com Пароль = 12345678 Підтвердження пароля = 12345679 | Користувач повинен отримати повідомлення про помилку, яку він припустив | Як і очікувалося | Pass |
| Перевірка скидання пароля | 1. Перейти на сторінку http://127.0.0.1:3000/login 2. Натиснути "Забули пароль" 3. Введіть email 4. Натисніть "Скинути пароль" 5. Перейти до пошти та перейти за вказаним посиланням 6. Введіть email 7. Введіть пароль 8. Підтвердіть пароль 9. Натисніть "Змінити пароль" | Логін = test email = test@gmail.com Пароль = 12345678 Підтвердження пароля = 12345678 | Пароль користувача повинен змінитися | Як і очікувалося | Pass |
| Перевірка додавання курсу з дійсними даними | 1. Увійти в систему у ролі адміністратора 2. Перейти до адмін-панелі 3. Перейти до вкладки курси і натиснути "Створити курс" 4. Введіть дійсні дані 5. Натисніть "Зберегти" | Назва = Прийменники в українській мові Опис = Опанування теми "Прийменники в українській мові" Час проходження = 45 | Повинен створитися новий курс з повідомленням про його успішне створення | Як і очікувалося | Pass |
| Перевірка сортування курсів | 1. Увійти в систему у ролі користувача 2. Перейти на сторінку з усіма курсами 3. Змінити тип сортування за допомогою відповідного спадного списку | Критерій = За популярністю | Відображення порядку курсів повинно змінитися відповідно до вказаного критерію | Як і очікувалося | Pass |

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

Також, необхідно перевірити обмеження певного функціоналу для користувачів. Для цього скористаємося API-платформої Postman [26]. Вона дозволяє розробникам розробляти, створювати, тестувати та повторювати свої API.

На рис. 3.19 зображено спробу редагувати не свій коментар.

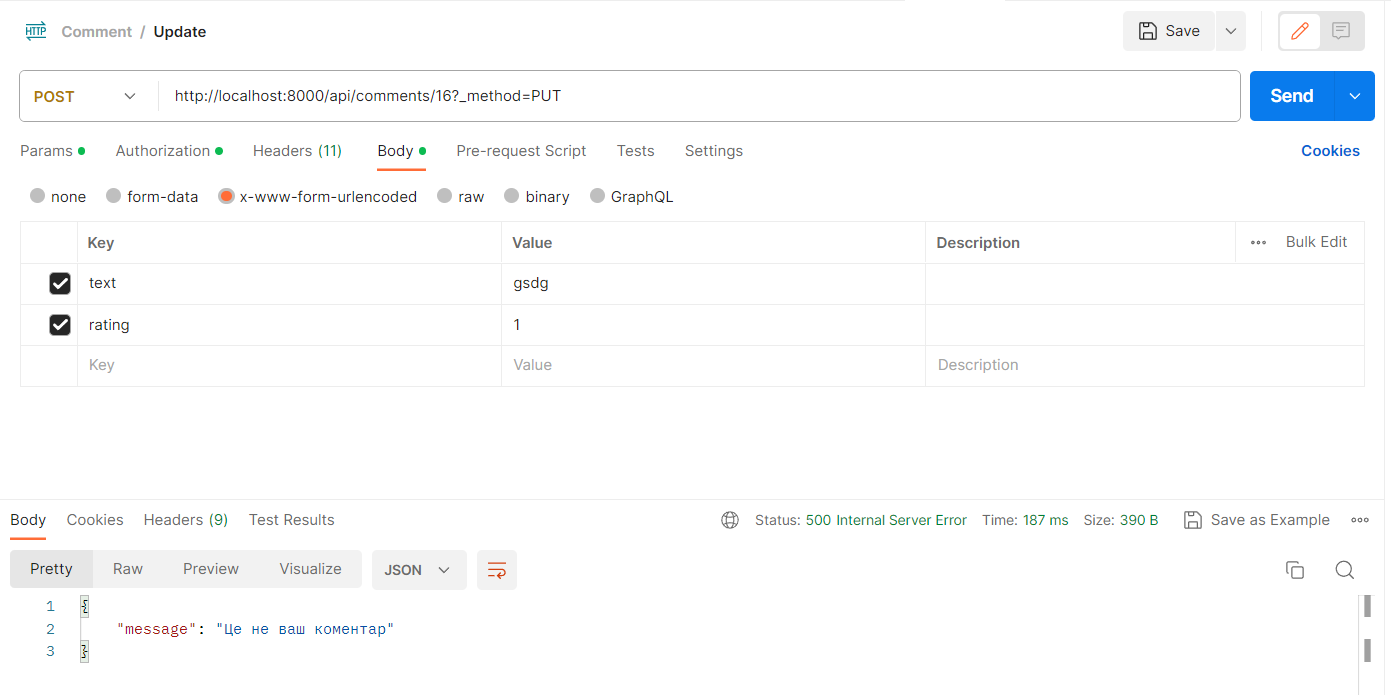


Рисунок 3.19 - Спроба редагувати свій коментар

Цю дію виконувати не вдалося, як і очікувалося. Отримано відповідне повідомлення.

Перевіримо спробу отримати розділ курсу без реєстрації на даний курс.

На рис. 3.20 зображено спробу отримати розділ курсу без реєстрації на даний курс.

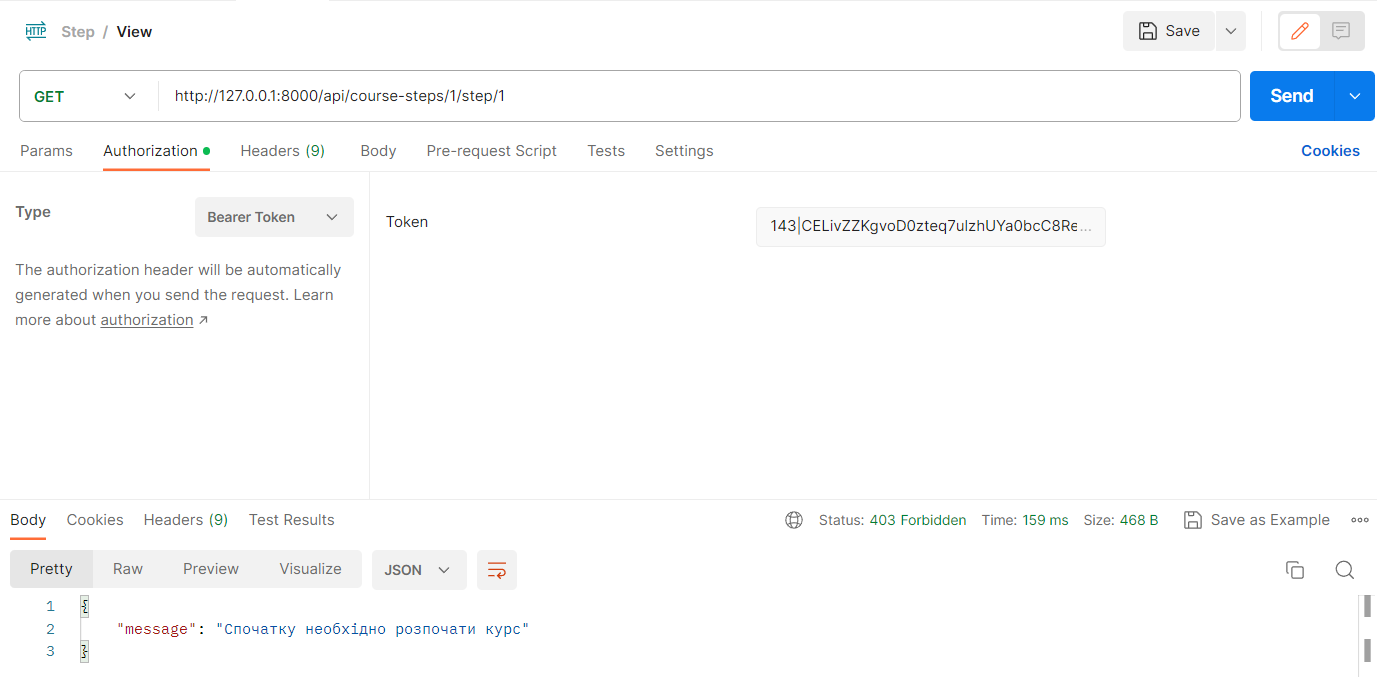


Рисунок 3.20 - Спроба отримати розділ курсу без реєстрації на даний курс

Цю дію виконувати не вдалося, як і очікувалося. Отримано відповідне повідомлення.

В ході проведення тестування було перевірено додаток та виправлено знайдені помилки. В результаті отримано застосунок відповідної якості.

Висновки до третього розділу

В даному розділі було побудовано діаграму розгортання та отримано дані щодо порядку встановлення та налаштування параметрів системи.

Також, було продемонстровано структуру інтерфейсу та порядок взаємодії з онлайн-сервісом. Було докладно описано функціональні можливості додатку в залежності від ролі користувача.

Було проведено тестування отриманого вебзастосунку за допомогою Test Cases та API-платформи Postman.

В ході проведення тестування було перевірено додаток та виправлено знайдені помилки. В результаті отримано застосунок відповідної якості.

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано онлайн-сервіс проходження курсів української мови. Виконання поставленої задачі було розділено на 3 розділи.

У першому розділі було виконано постановку задачі на кваліфікаційну роботу. Визначено функціональні вимоги та основні етапи роботи.

Проведено аналіз аналогів до продукту, що розроблюється. І завдяки цьому отримано наступні критерії для розробки: інтуїтивний та зрозумілий дизайн; наявність відеоконтенту; наявність коментарів; українська мова інтерфейсу; достатня кількість курсів; розподіл за категоріями; наявність вправ для перевірки своїх знань.

Було проаналізовано існуючі архітектури і обрано використовувати схему SPA для забезпечення користувачу досвіду, близького до користування настільною програмою.

Також, було обґрунтовано вибір використання наступного стеку технологій для реалізації застосунку: Laravel для побудови бекенд частини, React – для фронтенд частини; в якості системи керування базами даних обрано MySQL; в якості локального вебсервера буде використовуватися OpenServer.

Були встановлені вимоги до серверного обладнання та робочої станції користувача.

В другому розділі для наочного зображення відношення між акторами та прецедентами в системі, було побудовано діаграму варіантів використання. Було сформовано вимоги до системи. Було обґрунтовано та описано використання патернів при розробці застосунку. Було спроектовано базу даних та описано поля та їх призначення.

Описано, як було використано архітектуру Single-Page Application та обґрунтовано використання підходу RESTful API. Було продемонстровано загальний алгоритм роботи та побудовані наступні діаграми для презентативності: діаграму активностей системи, діаграму послідовності для оплати та діаграма компонентів системи.

Було продемонстровано реалізацію програмного комплексу та наведено фрагменти коду, що відповідають за авторизацію, оплату та завантаження відео на сервер з поясненням принципу їх роботи.

В третьому розділі було побудовано діаграму розгортання та отримано дані щодо порядку встановлення та налаштування параметрів системи.

Також, було продемонстровано структуру інтерфейсу та порядок взаємодії з онлайн-сервісом. Було докладно описано функціональні можливості додатку в залежності від ролі користувача.

Було проведено тестування отриманого вебзастосунку за допомогою Test Cases та API-платформи Postman.

В ході проведення тестування було перевірено додаток та виправлено знайдені помилки. В результаті отримано застосунок відповідної якості.

Розроблений програмний продуки готовий до використання. Онлайн-сервіс може бути використаний для покращення знань української мови громадян України. В майбутньому цей програмний продукт може еволюціонувати шляхом виявлення потреб у новому функціоналі, його реалізації та оновлення для всіх користувачів системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація React [електронний ресурс]. Режим доступу: <https://reactjs.org/> .
2. Документація Redux [електорнний ресурс]. Режим доступу: <https://redux.js.org/> .
3. Список відтворення «Курс «React JS шлях самурая 1.0, уроки практика» » [електорнний ресурс]. Режим доступу: <https://www.youtube.com/playlist?list=PLcvhF2Wqh7DNVy1OCUpG3i5lyxyBWhGZ8> .
4. Документація MS SQL [електронний ресурс]. Режим доступу: [https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16](https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16%20) .
5. Документація GraphQL.NET [електронний ресурс]. Режим доступу: <https://graphql-dotnet.github.io> .
6. Документація NHibernate [електорнний ресурс]. Режим доступу: <https://nhibernate.info>.
7. Jeffrey Richter. CLR via C#, 2012. - 894 c.

ДОДАТКИ

Додаток А

Клас, що відповідає за завантаження відео на сервер

<?php

namespace App\Repositories;

use App\Interfaces\BigFileInterface;

use App\Traits\FileUpload;

use App\Traits\ResponseApi;

use Illuminate\Http\JsonResponse;

use Illuminate\Http\Request;

use Illuminate\Http\UploadedFile;

use Illuminate\Support\Facades\App;

use Illuminate\Support\Facades\Storage;

use Illuminate\Support\Facades\Validator;

use Pion\Laravel\ChunkUpload\Exceptions\UploadFailedException;

use Pion\Laravel\ChunkUpload\Exceptions\UploadMissingFileException;

use Pion\Laravel\ChunkUpload\Handler\HandlerFactory;

use Pion\Laravel\ChunkUpload\Receiver\FileReceiver;

class BigFileRepository implements BigFileInterface

{

use ResponseApi, FileUpload;

public string $save\_path = 'public/uploadedVideos';

/\*\*

\* @param Request $request

\* @return JsonResponse

\*/

public function destroy(Request $request): JsonResponse

{

if (Storage::delete('/public/uploadedVideos/' . $request->file))

return $this->success('Файл успішно видалено');

return $this->error('Сталася помилка видалення');

}

/\*\*

\* @param Request $request

\* @return JsonResponse

\* @throws UploadMissingFileException

\* @throws UploadFailedException

\*/

public function uploadFile(Request $request): JsonResponse

{

$receiver = new FileReceiver('file', $request, HandlerFactory::classFromRequest($request));

if ($receiver->isUploaded() === false) {

throw new UploadMissingFileException();

}

$save = $receiver->receive();

if ($save->isFinished()) {

return $this->saveFile($save->getFile());

}

$handler = $save->handler();

return $this->success('Chunk успішно передано', ['done' => $handler->getPercentageDone(), 'status' => true]);

}

protected function saveFile(UploadedFile $file): JsonResponse

{

$fileName = $this->createFilename($file);

$mime = str\_replace('/', '-', $file->getMimeType());

$path = Storage::putFileAs($this->save\_path, $file, $fileName);

$path = preg\_replace('/public/', '', $path);

unlink($file->getPathname());

return $this->success('Файл успішно збережено', [

'complete' => true,

'path' => @App::make('url')->to('/') . '/storage' . $path,

'name' => $fileName,

'mime\_type' => $mime

]);

}

}

Додаток Б

Клас, що відповідає за тестування в системі

<?php

namespace App\Repositories;

use App\Http\Requests\QuestionRequest;

use App\Interfaces\QuestionInterface;

use App\Models\Course;

use App\Models\Order;

use App\Models\Question;

use App\Models\QuestionChoice;

use App\Models\User;

use App\Models\UserCourse;

use App\Traits\ResponseApi;

use Illuminate\Database\Eloquent\ModelNotFoundException;

use Illuminate\Http\Request;

use Illuminate\Http\JsonResponse;

use Symfony\Component\HttpFoundation\Response;

class QuestionRepository implements QuestionInterface

{

use ResponseApi;

public function store(QuestionRequest $request, int $course\_id): JsonResponse

{

try {

$question = Question::create([

'question' => $request->question,

'is\_active' => $request->is\_active,

'course\_id' => $course\_id

]);

foreach ($request->choices as $id => $choice) {

$question->choices()->create([

'choice' => $choice,

'is\_right' => $id == $request->is\_right ? 1 : 0,

'question\_id' => $question->id

]);

}

return $this->success('Запитання успішно створено', ['question' => $question]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function update(QuestionRequest $request, int $id): JsonResponse

{

try {

$question = Question::findOrFail($id);

$question->update([

'question' => $request->question,

'is\_active' => $request->is\_active,

'course\_id' => $question->course\_id

]);

foreach ($question->choices as $id => $choice) {

$choice->choice = $request->choices[$id];

$choice->is\_right = $id == $request->is\_right ? 1 : 0;

$choice->save();

}

return $this->success('Запитання успішно відредаговано', ['question' => $question]);

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function index(Request $request): JsonResponse

{

try {

if (!Order::where(['course\_id' => $request->course\_id, 'user\_id' => auth()->id(), 'status' => 'success'])->first() &&

!User::find(auth()->id())->isAdmin() && Course::findOrFail($request->course\_id)->price) {

return $this->error('Спочатку необхідно купити курс', Response::HTTP\_FORBIDDEN);

}

if (!UserCourse::where(['course\_id' => $request->course\_id, 'user\_id' => auth()->id()])->first() &&

!User::find(auth()->id())->isAdmin()) {

return $this->error('Спочатку необхідно розпочати курс', Response::HTTP\_FORBIDDEN);

}

$defaultPerPage = 10;

if ($request->random) {

$questions = Question::with(['choices' => function ($q) {

$q->inRandomOrder();

}])

->where(['is\_active' => 1, 'course\_id' => $request->course\_id])

->inRandomOrder()->limit(10)->get();

} else {

$questions = Question::where(['course\_id' => $request->course\_id])->paginate($request->per\_page ?? $defaultPerPage);

}

return $this->success('Запитання успішно отримані', ['questions' => $questions]);

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function show(int $id): JsonResponse

{

try {

$question = Question::with('choices')->findOrFail($id);

$question->choices->makeVisible('is\_right');

return $this->success('Запитання успішно отримано', ['question' => $question]);

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function destroy(int $id): JsonResponse

{

try {

$question = Question::findOrFail($id);

$question->delete();

return $this->success('Запитання успішно видалено');

} catch (ModelNotFoundException $e) {

return $this->error($e->getMessage(), Response::HTTP\_NOT\_FOUND);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function checkAnswers(Request $request): JsonResponse

{

try {

if (!$userCourse = UserCourse::where(['course\_id' => $request->all()[0]['course\_id'], 'user\_id' => auth()->id()])->first()) {

return $this->error('Спочатку необхідно розпочати курс', Response::HTTP\_FORBIDDEN);

}

$answers = [];

$right = 0;

foreach ($request->all() as $id => $item) {

$choice = QuestionChoice::findOrFail($item['choice']['id']);

$rightAnswer = QuestionChoice::where(['is\_right' => 1, 'question\_id' => $item['id']])->first();

if ($choice->is\_right) {

$answers['res'][$id] = ['res' => true];

$right++;

} else {

$answers['res'][$id] = ['res' => false, 'right' => $rightAnswer->choice];

}

}

$total = round($right / count($request->all()) \* 100);

$answers['total'] = $total;

if ($userCourse->grade < $total || !$userCourse->grade) {

$userCourse->grade = $total;

}

$userCourse->is\_finished = 1;

$userCourse->save();

return $this->success('Відповіді успішно опрацьовано', ['answers' => $answers]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

}

Додаток В

Клас, що відповідає за авторизацію

<?php

namespace App\Repositories;

use App\Http\Requests\ForgotPasswordRequest;

use App\Http\Requests\RegisterRequest;

use App\Http\Requests\ResetPasswordRequest;

use App\Interfaces\AuthInterface;

use App\Mail\ForgotMail;

use App\Models\PasswordReset;

use App\Models\User;

use App\Traits\ResponseApi;

use Illuminate\Http\JsonResponse;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\Mail;

use Symfony\Component\HttpFoundation\Response;

class AuthRepository implements AuthInterface

{

use ResponseApi;

/\*\*

\* Create User

\* @param RegisterRequest $request

\* @return JsonResponse

\*/

public function register(RegisterRequest $request): JsonResponse

{

try {

$user = User::create([

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

'color' => $this->generateRandomColor()

]);

$token = $user->createToken($user->name . '\_Token', [''])->plainTextToken;

return $this->success('Акаунт успішно створено', ['token' => $token, 'user' => $user]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Login User

\* @param Request $request

\* @return JsonResponse

\*/

public function login(Request $request): JsonResponse

{

$login = $request->login;

$fieldType = filter\_var($login, FILTER\_VALIDATE\_EMAIL) ? 'email' : 'name';

request()->merge([$fieldType => $login]);

try {

if (!Auth::attempt($request->only([$fieldType, 'password']))) {

return $this->error(

'Не знайдено користувача з вказаними даними.',

Response::HTTP\_UNAUTHORIZED);

}

$user = auth()->user();

if ($user->isAdmin()) {

$token = $user->createToken($user->name . '\_AdminToken', ['server:admin'])->plainTextToken;

$user->role = 1;

} else {

$token = $user->createToken($user->name . '\_Token', [''])->plainTextToken;

$user->role = 0;

}

return $this->success('Ви успішно увійшли в систему', ['token' => $token, 'user' => $user]);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Logout User

\* @return JsonResponse

\*/

public function logout(): JsonResponse

{

auth()->user()->tokens()->delete();

return $this->success('Ви успішно вийшли з системи');

}

/\*\*

\* Generate token for reset

\* @param ForgotPasswordRequest $request

\* @return JsonResponse

\*/

public function forgotPassword(ForgotPasswordRequest $request): JsonResponse

{

$email = $request->email;

$user = User::where('email', $email);

if ($user->doesntExist()) {

return $this->error('Не існує користувача з вказаним email', Response::HTTP\_UNAUTHORIZED);

}

$token = rand(10, 100000);

try {

if (!$userPassReset = PasswordReset::where('email', $email)->first()) {

PasswordReset::create([

'email' => $email,

'token' => $token

]);

} else {

$userPassReset->update([

'email' => $email,

'token' => $token

]);

}

Mail::to($email)->send(new ForgotMail($token, $user->first()->name));

return $this->success("Лист надіслано на пошту " . $email);

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Reset User's password

\* @param ResetPasswordRequest $request

\* @return JsonResponse

\*/

public function resetPassword(ResetPasswordRequest $request): JsonResponse

{

$user = User::where('email', $request->email)->first();

if (!$user) {

return $this->error('Не існує користувача з вказаним email', Response::HTTP\_UNAUTHORIZED);

}

$resetRequest = PasswordReset::where('email', $user->email)->first();

if (!$resetRequest || $resetRequest->token != $request->token) {

return $this->error('Не дійсний токен скидання', Response::HTTP\_BAD\_REQUEST);

}

try {

$user->fill([

'password' => Hash::make($request->password)

]);

$user->save();

$user->tokens()->delete();

$resetRequest->delete();

return $this->success("Пароль успішно змінено");

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

public function changePassword(Request $request)

{

try {

$user = User::findOrFail(auth()->id());

if ($request->is\_google && $user->password) {

return $this->error("У вас вже встановлений старий пароль");

}

if (!Hash::check($request->old\_password, $user->password)) {

return $this->error("Неправильний старий пароль");

}

$user->fill([

'password' => Hash::make($request->new\_password)

]);

$user->save();

return $this->success("Пароль успішно змінено");

} catch (\Throwable $th) {

return $this->error($th->getMessage());

}

}

/\*\*

\* Return random color

\*

\* @return string

\*/

private function generateRandomColor(): string

{

$red = dechex(rand(0, 200));

$green = dechex(rand(0, 200));

$blue = dechex(rand(0, 200));

$lightness = (hexdec($red) + hexdec($green) + hexdec($blue)) / 3;

if ($lightness > 180) {

return $this->generateRandomColor();

}

return '#' . $red . $green . $blue;

}

}

Додаток Г

Основний компонент клієнтської частини

import './App.css';

import {createBrowserRouter, RouterProvider} from "react-router-dom";

import GoogleLogin from "./pages/Authorization/GoogleLogin";

import {LoginForm} from "./pages/Authorization/LoginForm/LoginForm";

import {RegisterForm} from "./pages/Authorization/RegisterForm/RegisterForm";

import {ResetForm} from "./pages/Authorization/ResetForm/ResetForm";

import Layout from "./components/layouts/Layout";

import React from "react";

import ProtectedRoute from "./components/routes/ProtectedRoute";

import Courses from "./pages/Course/Courses/Courses";

import AdminProtectedRoute from "./components/routes/AdminProtectedRoute";

import CourseForm from "./pages/Course/CourseForm/CourseForm";

import CoursesIndex from "./pages/Course/CoursesIndex/CoursesIndex";

import Forbidden from "./pages/Errors/Forbidden/Forbidden";

import NotFound from "./pages/Errors/NotFound/NotFound";

import CourseView from "./pages/Course/CourseView/CourseView";

import CourseStepForm from "./pages/CourseStep/CourseStepForm/CourseStepForm";

import CourseStepView from "./pages/CourseStep/CourseStepView/CourseStepView";

import CourseStepsIndex from "./pages/CourseStep/CourseStepIndex/CourseStepsIndex";

import TagForm from "./pages/Tag/TagForm/TagForm";

import TagIndex from "./pages/Tag/TagIndex/TagIndex";

import AdminLayout from "./components/layouts/AdminLayout";

import Profile from "./pages/Authorization/Profile/Profile";

import QuizForm from "./pages/Quiz/QuizForm/QuizForm";

import QuizIndex from "./pages/Quiz/QuizIndex/QuizIndex";

import QuizView from "./pages/Quiz/QuizView/QuizView";

import {HelmetProvider} from "react-helmet-async";

import OrderIndex from "./pages/Order/OrderIndex";

import Main from "./pages/Main/Main";

function App() {

const router = createBrowserRouter([

{

path: "/",

element: <Layout/>,

children: [

{

path: "/",

element: <Main/>

},

{

path: "/login",

element: <LoginForm/>

},

{

path: "/register",

element: <RegisterForm/>

},

{

path: "/reset-password/:token",

element: <ResetForm/>

},

{

path: "/auth/login",

element: <GoogleLogin/>

},

{

path: "/error/403",

element: <Forbidden/>

},

{

path: "/error/404",

element: <NotFound/>

},

{

path: "\*",

element: <NotFound/>

}

],

},

{

path: "/",

element: <ProtectedRoute><Layout/></ProtectedRoute>,

children: [

{

path: "/courses",

element: <Courses/>,

},

{

path: "/course/:id",

element: <CourseView/>

},

{

path: "/course/:course\_id/step/:number\_step",

element: <CourseStepView/>

},

{

path: "/profile",

element: <Profile/>

},

{

path: "/course/:course\_id/quiz",

element: <QuizView/>

}

]

},

{

path: "/",

element: <AdminProtectedRoute><AdminLayout/></AdminProtectedRoute>,

children: [

{

path: "/course/add",

element: <CourseForm/>

},

{

path: "/tag/add",

element: <TagForm/>

},

{

path: "/tag/update/:id",

element: <TagForm/>

},

{

path: "/tags",

element: <TagIndex/>

},

{

path: "/course/:course\_id/step/update/:number\_step",

element: <CourseStepForm/>

},

{

path: "/course/:course\_id/step/add/",

element: <CourseStepForm/>

},

{

path: "/course/:course\_id/steps",

element: <CourseStepsIndex/>

},

{

path: "/courses/index",

element: <CoursesIndex/>

},

{

path: "/course/update/:id",

element: <CourseForm/>

},

{

path: "/course/:course\_id/quiz/update/:id",

element: <QuizForm/>

},

{

path: "/course/:course\_id/quiz/add",

element: <QuizForm/>

},

{

path: "/course/:course\_id/quizzes",

element: <QuizIndex/>

},

{

path: "/orders",

element: <OrderIndex/>

},

]

}

]);

return (

<div className="App">

<HelmetProvider>

<RouterProvider router={router}/>

</HelmetProvider>

</div>

);

}

export default App;