МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН ЧАТУ З ВИКОРИСТАННЯМ PYTHON ТА SOCKET IO»**

Виконав: студент 2 курсу, групи 6.1213-пі-с3
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія (зі скороченим терміном навчання)
(назва освітньої програми)
Б.Д. Меркулов
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія (зі скороченим терміном навчання)

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Меркулову Богдану Денисовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн чату з використанням Python та Socket IO

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Актуальність та основи реалізації онлайн-чату.
2. Проєктування системи онлайн-чату.
3. Реалізація онлайн-чату.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	13.01.2025	
2.	Збір вихідних даних.	03.02.2025	
3.	Обробка методичних та теоретичних	24.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	14.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2025	

Студент _____ Б.Д. Меркулов
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн чату з використанням Python та Socket IO»: 48 с., 20 рис., 1 табл., 11 джерел, 1 додаток.

ОБРОБКА ПОВІДОМЛЕНЬ, ОНЛАЙН-ЧАТ, FLASK, MONGODB, PYTHON, SOCKET.IO, VUE.

Об’єкт дослідження – процес розробки онлайн-чатів.

Предмет дослідження – Python, бібліотека Socket.IO.

Мета роботи – розробка онлайн-чату із використанням Python та бібліотеки Socket.IO, який забезпечує обмін повідомленнями між користувачами в реальному часі з мінімальними затримками та високою надійністю.

Методи дослідження – аналіз сучасних рішень, проєктування архітектури системи, а також методи тестування програмного забезпечення.

У кваліфікаційній роботі проведено дослідження щодо актуальності використання онлайн-чатів, а також принципи їх реалізації. Створено вебзастосунок, який надає можливість кільком користувачам одночасно обмінюватися повідомленнями в режимі реального часу. Серверна частина реалізована мовою Python з використанням фреймворку Flask та бібліотеки Socket.IO. Вона обробляє підключення клієнтів, надсилання й отримання повідомлень, а також керує подіями в системі. Для збереження повідомлень і даних користувачів використовується база даних MongoDB. Клієнтська частина побудована з використанням VueJS.

Здійснено тестування системи за допомогою засобів ручного навантажувального тестування, що включало одночасне підключення кількох користувачів, а також перевірку коректності роботи функціоналу чатів, груп та системи аутентифікації.

SUMMARY

Bachelor’s qualifying paper “Online Chat Development Using Python and Socket IO”: 48 pages, 20 figures, 1 table, 11 references, 1 supplement.

FLASK, MESSAGE HANDLING, MONGODB, PYTHON, SOCKET.IO, VUE, ONLINE CHAT.

The object of the study is the process of developing online chat systems.

The subject of the research is the Python programming language and the Socket.IO library.

The aim of the study is to develop an online chat application using Python and the Socket.IO library that enables real-time message exchange between users with minimal latency and high reliability.

The research methods include the analysis of modern solutions, system architecture design, and software testing techniques.

This qualification thesis investigates the relevance and implementation principles of online chat systems. A web application was developed that allows multiple users to exchange messages in real time. The server side is implemented in Python using the Flask framework and the Socket.IO library. It handles client connections, message transmission and reception, and manages system events. MongoDB is used to store messages and user data. The client side is built using VueJS.

The system was tested using manual load testing techniques, which included simultaneous connections by multiple users and verification of the correct operation of the chat functionality, group features, and the authentication system.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 7
1 Актуальність та основи реалізації онлайн-чату 9
1.1 Онлайн-чати як вид сучасної комунікації 9
1.2 Ринок та принципи роботи сучасних онлайн-чатів 10
1.3 Опис технологій для створення застосунку онлайн спілкування 12
1.4 Висновки до розділу 1 14
2 Проєктування системи онлайн-чату 15
2.1 Технічне завдання 15
2.2 Діаграма прецедентів 16
2.3 Архітектура системи 17
2.4 Розробка моделі даних 21
2.5 Інтерфейс користувача 25
2.6 Висновки до розділу 2 26
3 Реалізація онлайн-чату 27
3.1 Реалізація серверної частини 27
3.2 Реалізація бази даних 30
3.3 Реалізація клієнтської частини 31
3.3.1 Архітектура та технології 32
3.3.2 Основні компоненти 33
3.3.3 Взаємодія з сервером та управління станом 36
3.4 Навантажувальне тестування застосунку 39
3.5 Висновки до розділу 3 42
Висновки 43
Перелік посилань 44
Додаток А Компонент реєстрації 46

ВСТУП

За останні десятиліття розвиток інформаційних технологій та інтернету суттєво змінив спосіб спілкування між людьми. Якщо раніше основними засобами передачі інформації були телефонні розмови чи листи, то сьогодні все більше користувачів обирають миттєвий обмін повідомленнями через онлайн-застосунки. Онлайн-комунікація охопила різноманітні сфери: від особистого листування й дистанційного навчання до корпоративної взаємодії та технічної підтримки клієнтів.

Сучасний ринок засобів для онлайн-комунікації є насиченим: існує велика кількість як комерційних, так і відкритих рішень. Проте багато з них мають складну архітектуру, високу вартість інтеграції або потребують значних ресурсів для налаштування. Саме тому актуальною є розробка власного, простого у використанні онлайн-чату з використанням відкритих технологій, який можна адаптувати під конкретні задачі.

Метою роботи є розробка онлайн-чату Blyx із використанням Python та бібліотеки Socket.IO, який забезпечує обмін повідомленнями між користувачами в реальному часі з мінімальними затримками та високою надійністю.

Досягнення поставленої мети передбачає реалізацію таких завдань:

– проаналізувати основи роботи онлайн-чатів та протоколів реального часу;
– дослідити можливості використання технологій Python, Flask, Socket.IO, MongoDB та Vue.js;
– спроєктувати архітектуру чат-застосунку;
– реалізувати серверну та клієнтську частини системи;
– здійснити навантажувальне тестування застосунку.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел та додатків. У першому розділі розглянуто

актуальність теми, еволюцію онлайн-комунікації, аналіз існуючих рішень, принципи роботи чат-систем і використані технології. У другому розділі виконано проєктування онлайн-чату, сформульовано вимоги до системи, описано архітектуру, компоненти й логіку взаємодії користувачів. У третьому розділі представлено реалізацію чат-застосунку, описано код серверної та клієнтської частин, процес підключення бази даних та проведення тестування.

1 АКТУАЛЬНІСТЬ ТА ОСНОВИ РЕАЛІЗАЦІЇ ОНЛАЙН-ЧАТУ

1.1 Онлайн-чати як вид сучасної комунікації

У сучасному світі майже кожна людина користується застосунками для онлайн-спілкування та їх різноманітними функціями – від надсилання звичайних повідомлень до участі у великих онлайн-заходах за участю десятків осіб. Історія систем онлайн-комунікації бере свій початок у 1973 році, коли для першої системи електронного навчання PLATO було розроблено перший чат–інтерфейс Talkomatic. За визначенням чат – це практично будь-яке онлайн-спілкування, яке забезпечує передачу текстових повідомлень від відправника до одержувача в реальному часі [1]. Але, при цьому треба розрізняти поняття онлайн-чат та месенджер – в цій роботі розроблено додаток, який є повністю самостійним, тому більше підходить під визначення месенджер.

Месенджер, на відміну від чату, є повноцінним застосунком або сервісом для спілкування. Він дозволяє створювати облікові записи, додавати контакти, вести тривалі діалоги, створювати групи, надсилати медіафайли, здійснювати дзвінки тощо. Месенджери підтримують синхронізацію між пристроями, зберігають історію повідомлень у хмарному сховищі й надають розширені функції взаємодії. Прикладами таких платформ є Telegram, WhatsApp, Signal.

Онлайн-чати можна класифікувати за різними критеріями, а саме:

– за способом доступу: відкриті (гостьові), обмежені (реєстрація або авторизація обов’язкова), корпоративні (внутрішні чати в організаціях);
– за кількістю учасників: приватні чати (1:1), групові чати, загальні публічні кімнати;
– за технологічною реалізацією: синхронні (реального часу) та асинхронні (на кшталт форумів чи електронної пошти);

– за метою використання: підтримка клієнтів, навчання, ігрова комунікація, внутрішнє спілкування в командах, соціальна взаємодія [2].

Онлайн спілкування використовується у багатьох сферах, наприклад:

– освіта – ще з часів пандемії коронавірусу завдяки онлайн чатам та відеоконференціями навчальні заклади продовжили навчальний процес;
– бізнес – великі компанії та корпорації впроваджують у свої системи та сайти онлайн-чати для спілкування з клієнтами або між співробітниками;
– медицина – впроваджуються онлайн прийоми лікарів з можливістю не виходячи з дому розповісти про проблему та отримати лікування;
– державні послуги – органи державної влади впроваджують чати на своїх порталах для полегшення подачі запитів громадянами.

Таким чином, онлайн-чати залишаються важливим елементом цифрової взаємодії, охоплюючи дедалі більше галузей. Зі стрімким розвитком штучного інтелекту дедалі частіше з’являються AI-чат-боти, які дозволяють автоматизувати спілкування, підвищити ефективність обслуговування користувачів та зменшити навантаження на операторів.

1.2 Ринок та принципи роботи сучасних онлайн-чатів

Прямо зараз ринок рішень онлайн спілкування насичений безліччю пропозицій – від найпримітивніших до найскладніших систем із плагінами та доповненнями. У топ 5 лідерів входять:

– Whatsapp – понад 2 мільярди користувачів;
– Facebook Messenger – понад 1,3 мільярда користувачів;
– WeChat – понад 1,2 мільярда користувачів;
– Instagram Direct – понад 1 мільярд користувачів;
– Telegram – понад 800 мільйонів користувачів [3].

Ці додатки надають на безоплатній основі користувачам функції чату, груп із десятками та тисячами учасників, надсилання відео та аудіо повідомлень, каналів, емодзі та стікери, відеоконференції, тощо.

Окрім цих гігантів, свою нішу займають спеціалізовані сервіси – наприклад, Slack та Microsoft Teams, які орієнтовані на внутрішню комунікацію в командах та підприємствах. Для сайтів і малого бізнесу популярними є вбудовані чати у реальному часі, як Tawk.to та LiveChat, які дозволяють інтегрувати чат напряму в інтерфейс вебресурсу.

За даними Statista, загальна кількість активних користувачів месенджерів станом на 2024 рік перевищує 5 мільярдів, а темпи зростання особливо високі в країнах із молодим населенням та швидкою цифровізацією [4].

Подібні застосунки реалізують функціонал синхронного спілкування, де повідомлення доставляються та відображаються відразу після надсилання. Це забезпечує високу швидкість взаємодії між користувачами без помітних затримок. Принцип роботи онлайн-чату полягає в організації постійного зв’язку між клієнтами (браузерами або мобільними додатками) та сервером, що обробляє й передає повідомлення. Коли користувач вводить повідомлення і натискає кнопку «Надіслати», клієнтська частина передає це повідомлення серверу. Далі сервер зберігає його у базі даних (або буфері), а також передає іншим підключеним клієнтам, які відображають його у своєму інтерфейсі. Така модель забезпечує негайну реакцію системи, що і є ключовою характеристикою чату [5].

Онлайн-чати, як правило, реалізують логіку обміну повідомленнями, обробки подій, керування сесіями користувачів та історії повідомлень. Також важливою складовою є система облікових записів: користувачі проходять реєстрацію або авторизацію, після чого мають можливість надсилати повідомлення, приєднуватися до чатів і переглядати історію спілкування.

Функціонально сучасні чати зазвичай підтримують:

– приватне спілкування та групові чати;
– передачу мультимедійних даних (фото, відео, документи);

– сповіщення про нові повідомлення;
– статуси активності користувачів;
– можливість перегляду історії повідомлень.

Таким чином, основний принцип роботи онлайн-чату полягає в забезпеченні миттєвого, двостороннього обміну повідомленнями між учасниками системи за допомогою постійного з’єднання з сервером. Це дозволяє користувачам ефективно спілкуватися незалежно від відстані, пристрою чи платформи.

1.3 Опис технологій для створення застосунку онлайн спілкування

Для реалізації онлайн-чату було обрано сучасні вебтехнології, які дозволяють створити інтерактивний, масштабований та зручний у використанні застосунок. Рішення базується на клієнт-серверній архітектурі, де клієнтська частина відповідає за інтерфейс користувача, а серверна – за обробку логіки обміну повідомленнями та взаємодію з базою даних.

Python є сучасною та потужною мовою програмування, відомою своєю легкістю у засвоєнні. Вона поєднує зручні високорівневі структури даних з простим, але ефективним об’єктно-орієнтованим підходом. Завдяки зрозумілому синтаксису, динамічній типізації та інтерпретованому виконанню, Python підходить для написання сценаріїв і швидкої розробки програм у різних галузях та на більшості операційних систем. У розроблюваному додатку Python застосовується для створення серверної частини застосунку, оскільки підтримує асинхронну обробку запитів і має великий вибір бібліотек для роботи з мережею.

Flask – це легкий і гнучкий фреймворк Python, призначений для розробки маленьких і середніх вебдодатків. Він надає базові інструменти для створення вебдодатків і дає розробникам велику свободу у виборі бібліотек та інструментів. Flask має мінімалістичний дизайн, завдяки чому підходить для

простих і швидких проєктів [6]. Під час розробки кваліфікаційної роботи став у нагоді для створення REST API, обробки маршрутів, авторизації користувачів та організації архітектури проєкту.

Socket.IO – це бібліотека з відкритим кодом, крос-платформенний інструмент, який забезпечує двосторонню комунікацію між клієнтом і сервером на основі подій. Вона підтримує двосторонню подієву комунікацію, що дозволяє як клієнту, так і серверу миттєво надсилати і отримувати повідомлення. Socket.IO також включає механізми резервного копіювання, що забезпечують надійну комунікацію [7]. Саме завдяки Socket.IO в чаті реалізовано миттєву передачу повідомлень без потреби оновлення сторінки.

MongoDB – це документоорієнтована система управління базами даних, що не вимагає опису схеми таблиць [8]. MongoDB – це нереляційна база даних (NoSQL), у якій інформація зберігається не у вигляді таблиць, як у реляційних (SQL), а у більш гнучкому форматі – наприклад, у вигляді документів, графів або пар «ключ-значення». Дані зберігаються у вигляді JSON документів, що дозволяє легко працювати з різною структурою даних без складних зв’язків і схем. Є легко масштабованою та ідеально підходить для роботи онлайн-чату: гнучкого збереження повідомлень, облікових записів користувачів та історії чатів.

VueJS – це прогресивний фреймворк для JavaScript, який використовується для створення вебінтерфейсів і односторінкових програм [9]. Він дозволяє будувати інтерактивні інтерфейси, оновлювати частини сторінки без перезавантаження та динамічно реагувати на дії користувача.

У проєкті Vue.js використовується для реалізації вікна чату, форм авторизації, списку повідомлень тощо. Також, слід зауважити, що в VueJS працює на основі стандартних технологій HTML, CSS та Javascript. HTML – це мова розмітки, що використовується для розташування елементів, CSS – таблиці стилів для стилізації додатку, а Javascript – це логіка застосунку.

1.4 Висновки до розділу 1

У цьому розділі розглянуто основні аспекти, що стосуються актуальності, розвитку, принципів роботи та технологій, які застосовуються для реалізації сучасних онлайн-чатів.

Коротко проаналізовано ринок та принцип роботи найбільш популярних чат-застосунків. Визначено, що всі сучасні чати працюють за клієнт-серверною моделлю із застосуванням двостороннього зв’язку для передачі повідомлень у реальному часі.

Також було обґрунтовано вибір технологій для реалізації власного чат-застосунку. Використання мови програмування Python, фреймворку Flask, бібліотеки Socket.IO, бази даних MongoDB та фреймворку Vue.js дає змогу створити швидкий, масштабований і зручний застосунок, адаптований до потреб користувачів.

2 ПРОЄКТУВАННЯ СИСТЕМИ ОНЛАЙН-ЧАТУ

2.1 Технічне завдання

Blyx – легкий і сучасний вебзастосунок для обміну повідомленнями в реальному часі. Blyx може бути розгорнутий як самостійний застосунок або адаптований як частина більшої системи, що передбачає обмін повідомленнями.

Під час розробки програмного забезпечення важливо чітко визначити завдання, які необхідно виконати для досягнення основної мети проєкту. У межах створення вебзастосунку Blyx – системи онлайн-спілкування в реальному часі – було сформульовано функціональні та нефункціональні вимоги.

Розглянемо **функціональні** вимоги.

Реєстрація та авторизація користувачів:

– створення облікового запису з унікальним логіном;
– авторизація зареєстрованих користувачів через форму входу;
– збереження та захист даних облікового запису.

Обмін повідомленнями:

– можливість надсилати текстові повідомлення в реальному часі;
– миттєве відображення отриманих повідомлень без перезавантаження сторінки;
– збереження історії повідомлень у базі даних.

Чат-кімнати:

– створення кімнат для приватного чи групового спілкування;
– можливість приєднання до активних кімнат;
– відображення списку учасників кімнати.

Інтерфейс користувача:

– реалізація зручного та інтуїтивного вебінтерфейсу для взаємодії з чатом;

– відображення активних чатів, повідомлень та статусу підключення;
– форма введення повідомлення з кнопкою надсилання.

Реалізація логіки чату:

– встановлення двостороннього з’єднання між сервером і клієнтами через Socket.IO;
– обробка подій підключення, відключення, надсилання й отримання повідомлень.

Нефункціональні вимоги.

Забезпечення надійності системи:

– обробка некоректного введення;
– перевірка підключення до сервера;
– забезпечення роботи при одночасному підключенні кількох користувачів.

Безпека даних:

– хешування паролів користувачів;
– захист від несанкціонованого доступу;
– обмеження доступу до певних функцій для неавторизованих користувачів.

Масштабованість:

– проєктування архітектури, що дозволяє додавати нові функції без суттєвих змін у коді;
– можливість підключення інших клієнтів (наприклад, мобільного застосунку) у майбутньому.

2.2 Діаграма прецедентів

Діаграма прецедентів – графічне зображення основних можливостей системи та взаємодії між користувачами (акторами) і самим застосунком. Вона допомагає зрозуміти, які дії користувачі можуть виконувати, а також які сервіси та процеси підтримує система.

Діаграма прецедентів використовує 2 основних елементи:

– actor (користувач) – множина логічно пов’язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (система, підсистема або клас) (учасником може бути людина, роль людини в системі чи інша система, підсистема або клас, які представляють щось поза сутністю);
– use case (прецедент) – опис окремого аспекту поведінки системи з боку користувача [10].

Для розроблюваної системи була розроблена діаграма прецедентів, що зображена на рисунку 2.1.

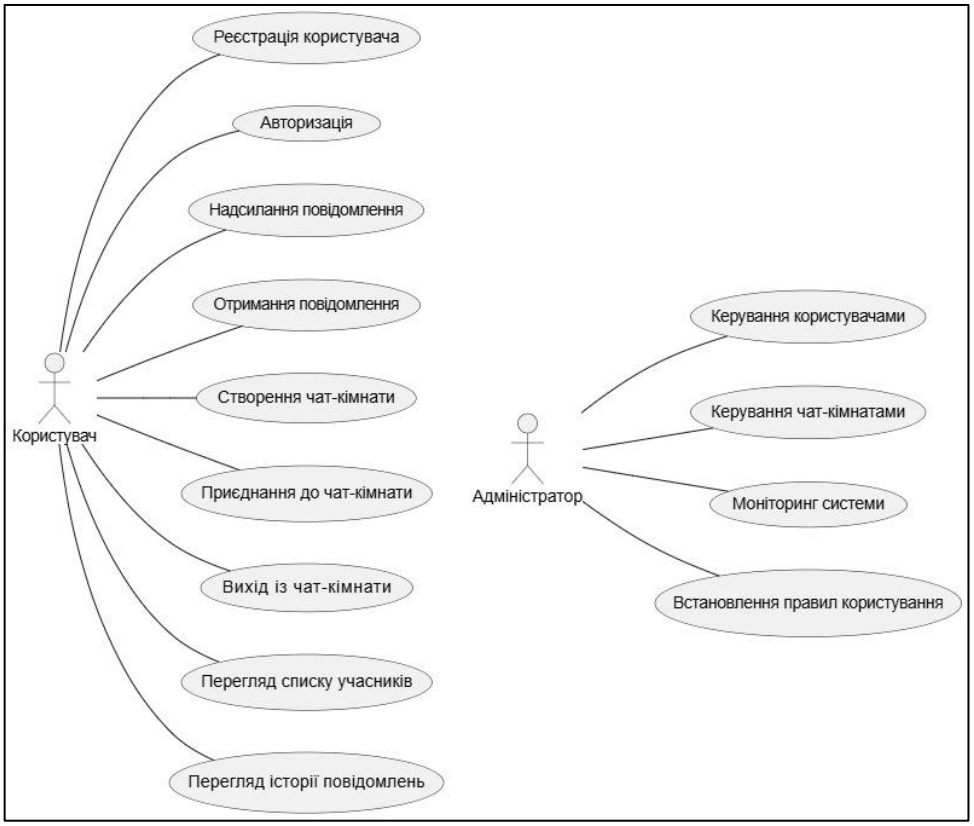


Рисунок 2.1 – Діаграма варіантів використання (прецедентів)

2.3 Архітектура системи

Загалом система побудована за клієнт-серверною моделлю, де фронтенд та бекенд взаємодіють між собою. Користувач за допомогою клієнтської

оболонки посилає запит на сервер, який обробляється, заносить необхідну інформацію до бази даних та повертає до інтерфейсу користувача відповідь у необхідному форматі для виводу.

Клієнтська частина була побудована за технологією Vue.js, з використанням різних функцій фреймворку, наприклад, Vue – model, Vue – router, Vue – click, хуків, тощо. Клієнтська частина реалізує інтерфейс, через який користувачі можуть переглядати повідомлення, надсилати власні повідомлення, приєднуватися до чат-кімнат та керувати своїм профілем. Ключовими елементами клієнта є:

- вікно чату – відображає поточні повідомлення в реальному часі, включаючи текстові повідомлення, статуси активності інших користувачів, сповіщення про нові повідомлення;
- форма введення повідомлення – дозволяє користувачам вводити текст і відправляти його одним кліком або натисканням клавіші Enter;
- список чат-кімнат – відображає доступні кімнати, в які користувач може приєднатися або створити нову;
- інтерфейс авторизації та реєстрації – забезпечує доступ до системи тільки зареєстрованим користувачам.

Для забезпечення зв’язку з сервером клієнтська частина використовує бібліотеку Socket.IO, яка встановлює двонаправлене WebSocket – з’єднання з сервером. Це дає змогу миттєво передавати та отримувати повідомлення без необхідності оновлення сторінки, що є ключовою вимогою для систем реального часу.

Інтерфейс підтримує адаптивний дизайн, що дозволяє зручно використовувати чат як на десктопах, так і на мобільних пристроях.

Серверна частина системи онлайн-чату Blyx виконує ключову роль у обробці запитів клієнтів, управлінні логікою обміну повідомленнями, а також збереженні даних. Вона реалізована на мові програмування Python із використанням вебфреймворку Flask та бібліотеки Flask-SocketIO, що забезпечує підтримку двонаправленої комунікації в режимі реального часу.

Основні функції серверної частини включають:

– обробку HTTP-запитів для реєстрації, авторизації користувачів, а також отримання та надсилання даних через REST API;
– управління WebSocket-з’єднаннями для миттєвого надсилання та прийому повідомлень між користувачами за допомогою Socket.IO;
– маршрутизацію подій, пов’язаних із підключенням і відключенням користувачів, надсиланням повідомлень, приєднанням і виходом із чат-кімнат;
– збереження інформації про користувачів, чат-кімнати та повідомлення у базі даних MongoDB, що дозволяє зберігати історію листування та забезпечує швидкий доступ до даних;
– забезпечення безпеки шляхом хешування паролів, аутентифікації користувачів і контролю доступу до захищених ресурсів.

Архітектура серверної частини спроєктована таким чином, щоб ефективно працювати з одночасним підключенням великої кількості користувачів, забезпечуючи високу продуктивність і стабільність. Асинхронна обробка подій через Flask-SocketIO дозволяє системі реагувати на події миттєво, що є критичним для застосунків реального часу, таких як онлайн-чати.

На рисунку 2.2 зображено діаграму послідовності, яка ілюструє послідовність взаємодій між користувачем, клієнтською частиною, сервером та базою даних під час надсилання та отримання повідомлень у чаті. Вона демонструє ключові кроки обробки повідомлень у системі реального часу.

Такий застосунок обов’язково має мати базу даних – була обрана нереляційна база даних MongoDB, яка на відміну від реляційних баз даних має значно вищу гнучкість та адаптованість для різних форматів даних, що є особливо важливим для застосунків із великим обсягом повідомлень і користувачів.

Архітектурно база даних працює окремо від сервера, але тісно з ним взаємодіє. Сервер отримує запити від клієнтів, обробляє їх і через спеціальні

команди передає дані у базу. Потім отримує відповіді – наприклад, історію повідомлень або інформацію про користувачів – і надсилає їх назад клієнтам. На рисунку 2.3 зображено діаграму розгортання такої архітектури.

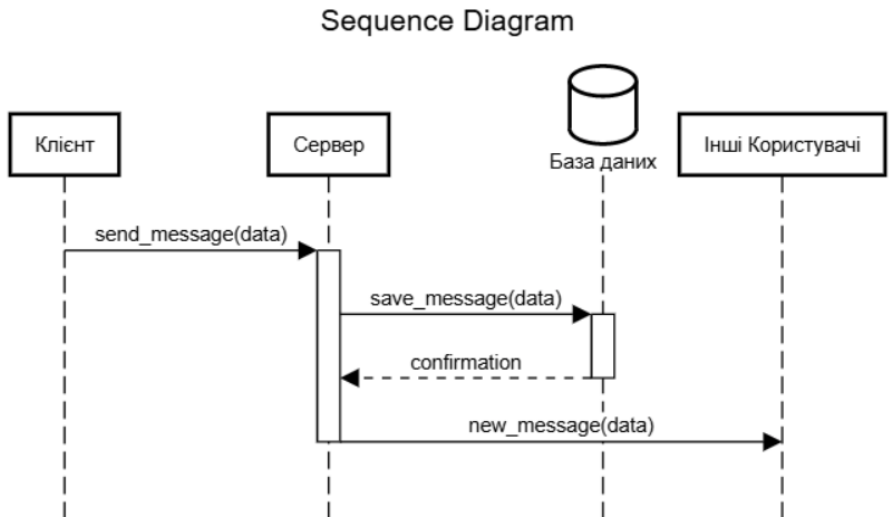


Рисунок 2.2 – Діаграма послідовності обробки повідомлень

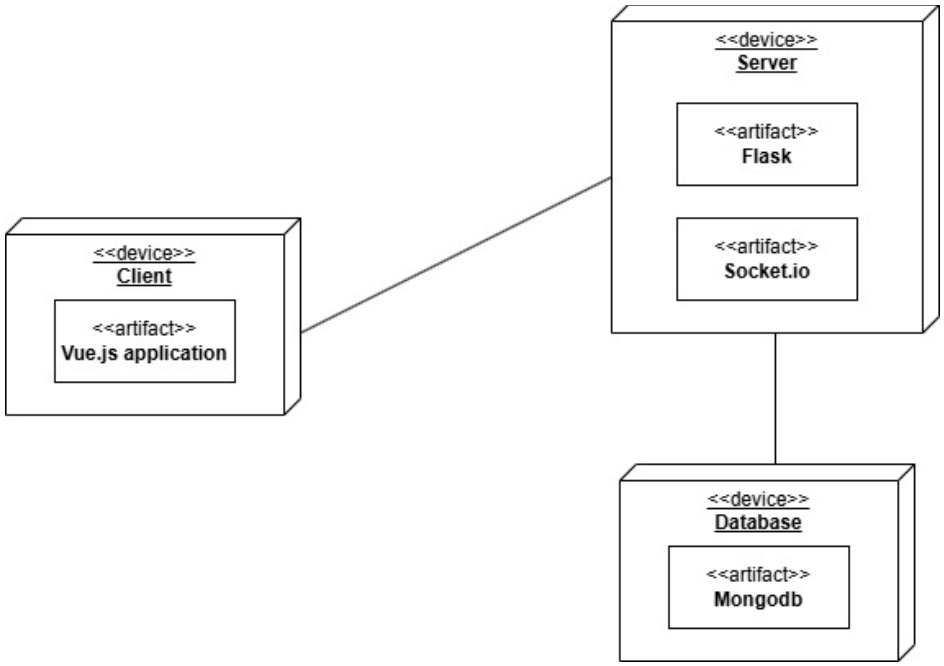


Рисунок 2.3 – Діаграма розгортання

Слід зазначити, що подібна архітектура використовується у всесвітньо відомому месенджері WhatsApp, який завдяки їй досягає високої стабільності, швидкості роботи і надійності, обслуговуючи мільярди користувачів по всьому світу [11].

2.4 Розробка моделі даних

Для забезпечення ефективного обміну повідомленнями та збереження даних про користувачів, чати та повідомлення, було розроблено модель даних, що включає основні сутності, їх атрибути та зв’язки між ними.

Користувачі (user) є основною сутністю, яка взаємодіє з усіма іншими елементами системи. Для кожного користувача створюється обліковий запис, який зберігає його персональну інформацію, логін, пароль та інші дані.

Поля:

– user_id: унікальний ідентифікатор користувача;
– username: логін користувача;
– email: електронна пошта користувача;
– password_hash: хеш пароля користувача;
– first_name: ім’я користувача;
– last_name: прізвище користувача;
– profile_picture: посилання на фото профілю користувача;
– created_at: час створення облікового запису;
– last_online: час останньої активності користувача.

Користувач має змогу реєструватися, авторизуватися та змінювати дані свого профілю. Всі ці дані зберігаються в базі даних, забезпечуючи доступність для взаємодії з іншими елементами системи.

Чат-кімнати (ChatRoom) – це простори для обміну повідомленнями, де можуть взаємодіяти кілька користувачів. Вони можуть бути як приватними, так і публічними, залежно від типу кімнати. Кожна чат-кімната має свої атрибути та особливості.

Поля:

– chatroom_id: унікальний ідентифікатор чат-кімнати;
– name: назва чат-кімнати;
– description: опис чат-кімнати;
– type: тип чат-кімнати (приватна, групова, публічна);

– created_at: дата і час створення чат-кімнати;
– creator_id: ідентифікатор користувача, який створив чат-кімнату.

Чат-кімнати дозволяють користувачам приєднуватися до спільних обговорень. Вони можуть бути організовані за інтересами, проєктами або іншими критеріями.

Повідомлення (Message) є основним елементом обміну інформацією в чат-кімнатах. Кожне повідомлення містить текст або інші мультимедійні файли, і зберігається в системі для подальшого доступу.

Поля:

– message_id: унікальний ідентифікатор повідомлення;
– content: текст повідомлення;
– Sent_at: час відправлення повідомлення;
– user_id: ідентифікатор користувача, який надіслав повідомлення;
– chatroom_id: ідентифікатор чат-кімнати, до якої належить повідомлення;
– message_type: тип повідомлення (текст, медіа, тощо);
– media_url: url до медіафайлів (якщо повідомлення містить медіафайл).

Повідомлення зберігаються в базі даних для забезпечення доступу до історії повідомлень у чатах. Також система підтримує різні типи повідомлень, такі як текстові, зображення, відео або аудіо.

Учасники (Participant) є основними учасниками чат-кімнат. Кожен користувач може бути учасником кількох чатів, і на основі цього визначаються його права (учасник, адміністратор).

Поля:

– participant_id: унікальний ідентифікатор учасника;
– User_id: ідентифікатор користувача;
– chatroom_id: ідентифікатор чат-кімнати;
– role: роль учасника в чат-кімнаті (учасник, адміністратор);
– joined_at: час приєднання до чат-кімнати.

Учасники можуть змінювати свої налаштування, а також додавати нових користувачів до чатів (якщо вони мають відповідні права).

Історія повідомлень (MessageHistory) містить інформацію про всі повідомлення, що були надіслані в чатах. Вона необхідна для зберігання та відновлення минулих бесід, доступних для учасників чат-кімнат.

Поля:

– history_id: унікальний ідентифікатор історії повідомлень;
– message_id: ідентифікатор повідомлення;
– chatroom_id: ідентифікатор чат-кімнати;
– user_id: ідентифікатор користувача, який надіслав повідомлення;
– sent_at: час відправлення повідомлення;
– content: текст повідомлення;
– message_type: тип повідомлення (текст, медіа).

Історія повідомлень зберігається для доступу учасникам чатів до минулих розмов і забезпечує збереження важливих даних на сервері.

Безпека та блокування (Security & Blocking) містить інформацію про блокування користувачів і забезпечення безпеки платформи.

Для візуалізації всіх сутностей та зв’язків між ними була створена ER-діаграма бази даних, яку можна побачити рисунку 2.4.

У розробленій моделі даних між таблицями встановлено декілька типів зв’язків, які забезпечують цілісність та логічну структуру системи. Нижче наведено пояснення основних зв’язків між сутностями бази даних.

Таблиця User має зв’язок один-до-багатьох з таблицями ChatRoom, Message та SecurityBlocking, що дозволяє одному користувачеві створювати множину чат-кімнат, надсилати багато повідомлень та мати декілька записів блокування.

Зв’язок багато-до-багатьох між User та ChatRoom реалізується через проміжну таблицю Participant, яка також зберігає роль користувача в кожній чат-кімнаті.

ChatRoom має зв’язок один-до-багатьох з Message, оскільки одна кімната може містити множину повідомлень.

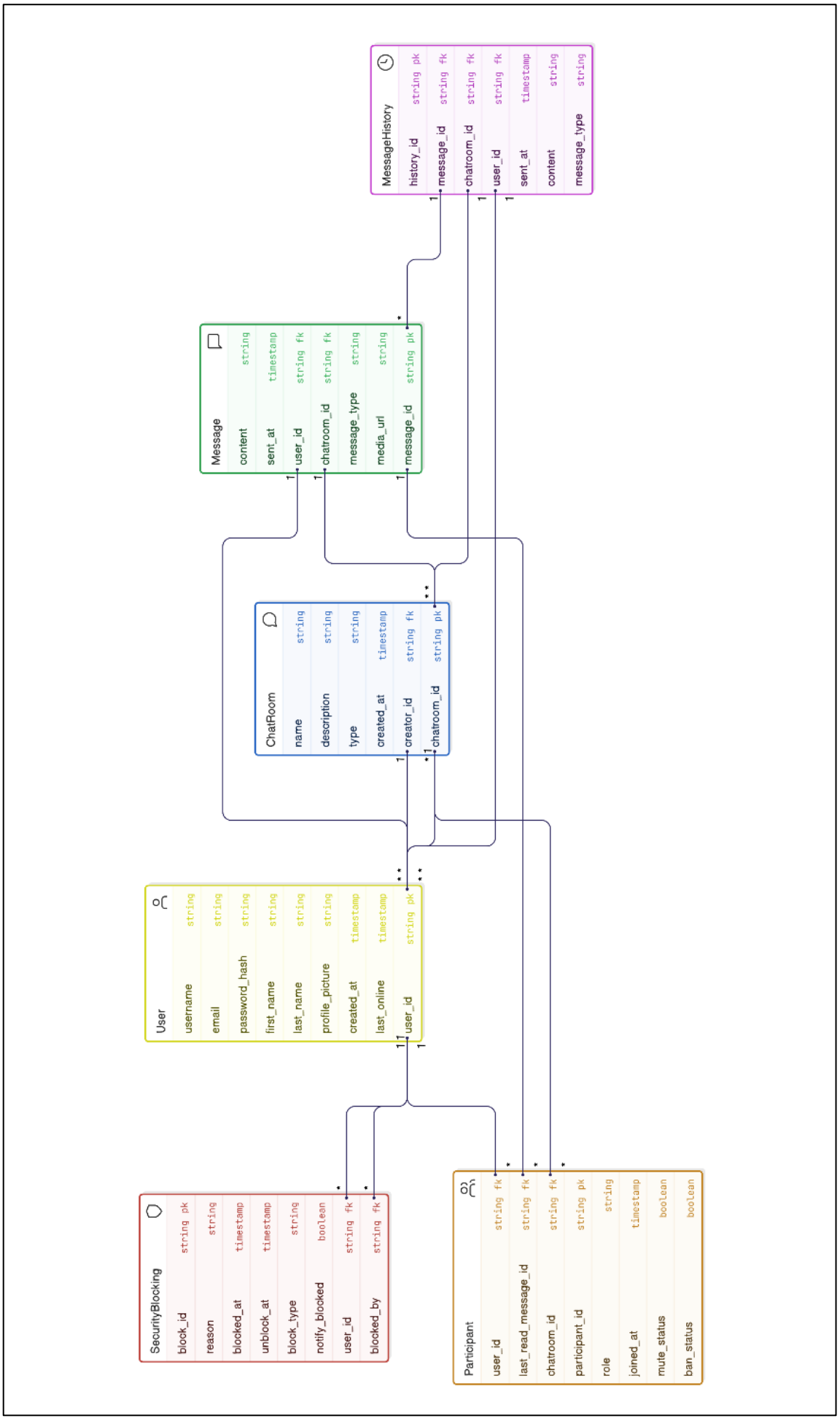


Рисунок 2.4 – ER-діаграма бази даних

Кожне повідомлення з таблиці Message має відповідний запис в MessageHistory, що реалізує зв’язок один-до-одного для збереження історії. Усі зв’язки забезпечуються через зовнішні ключі, що гарантує цілісність даних у системі.

2.5 Інтерфейс користувача

Інтерфейс користувача (UI) є одним із найважливіших аспектів у розробці будь-якої веб- або мобільної платформи, оскільки він визначає взаємодію користувача з системою та загальну зручність використання додатка. Для онлайн-чату Blyx було створено інтуїтивно зрозумілий, простий та функціональний інтерфейс, що забезпечує комфортне спілкування користувачів без складних та зайвих елементів дизайну. За допомогою застосунку Figma було розроблено майбутній інтерфейс застосунку. На рисунках 2.5 та 2.6 можна побачити приклади деяких екранів застосунку.

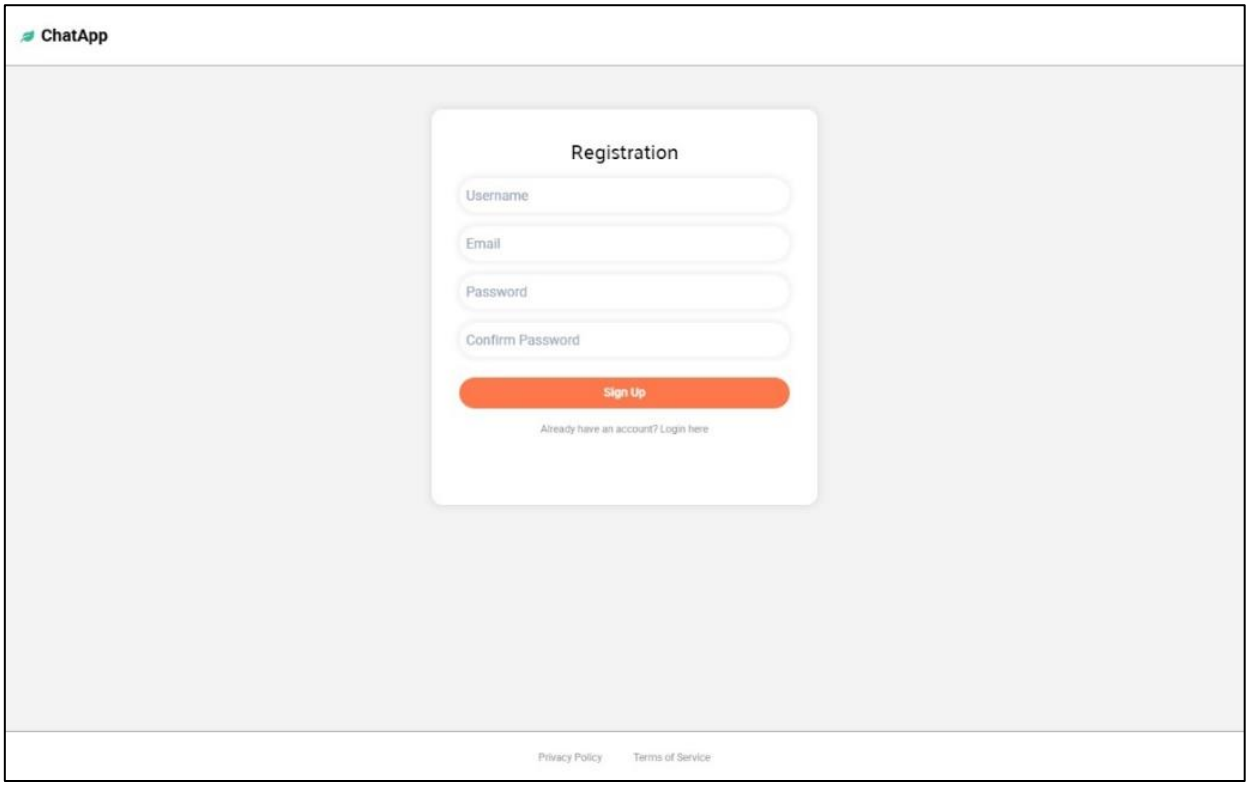


Рисунок 2.5 – Інтерфейс екрану реєстрації

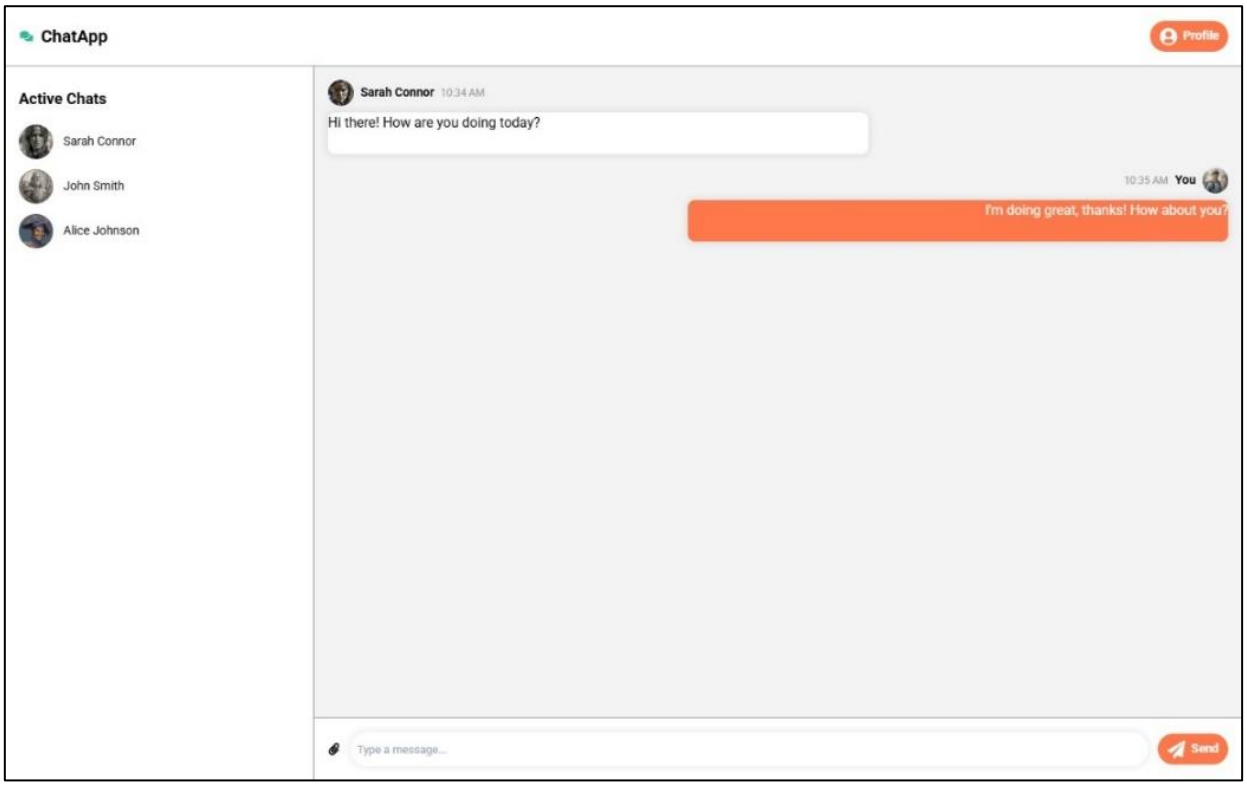


Рисунок 2.6 – Інтерфейс основного екрану усіх чатів

2.6 Висновки до розділу 2

У другому розділі розглянуто проєктування системи Blyx, що включає визначення функціональних і нефункціональних завдань. Було спроєктовано клієнтську частину на основі Vue.js та серверну частину з використанням Python, Flask, Socket.IO для забезпечення обміну повідомленнями в реальному часі. Розроблена модель даних включає основні сутності, такі як користувачі, чат-кімнати і повідомлення. Інтерфейс користувача створений простим і інтуїтивно зрозумілим. Це забезпечує ефективне функціонування системи для зручного спілкування користувачів.

3 РЕАЛІЗАЦІЯ ОНЛАЙН-ЧАТУ

3.1 Реалізація серверної частини

Серверна частина вебзастосунку Blyx реалізована на базі мікрофреймворку Flask із використанням бібліотеки Flask-SocketIO для підтримки двонаправленої комунікації в режимі реального часу. Ці технології побудовані на мові програмування Python.

Початок роботи сервера забезпечує файл run.py, який ініціалізує Flask-додаток та запускає Socket.IO сервер для обробки подій у реальному часі. Нижче на рисунку 3.1 наведено код ініціалізації.

```
from app import create_app
app, socketio = create_app()
socketio.run(app, host="0.0.0.0",
        port=5000,
        debug=True,
        use_reloader=True)
```

Рисунок 3.1 – Код ініціалізації серверу

Далі відбувається створення Flask-додатку, завантаження конфігурації та підключення необхідних розширень: MongoDB для зберігання даних, JWT для аутентифікації користувачів, а також Socket.IO для підтримки WebSocket-з’єднань.

Розроблено REST API для виконання операцій реєстрації, авторизації, оновлення профілю, отримання інформації про користувача, тощо. Усі REST API побудовані відповідно до принципів RESTful архітектури. Кожен маршрут відповідає за CRUD-операції над ресурсами, такими як користувачі, повідомлення, профілі. Запити підтримують передачу JWT-токенів у заголовках Authorization, що дозволяє обмежити доступ до захищених ресурсів. Основні маршрути API наведені у таблиці 3.1.

Таблиця 3.1 – Основні REST API маршрути серверної частини

Метод	**Шлях**	**Опис**
POST	/api/v1/auth/register	Реєстрація користувача
POST	/api/v1/auth/login	Авторизація користувача
POST	/api/v1/auth/logout	Вихід із системи
POST	/api/v1/auth/refresh	Оновлення JWT токена
GET	/api/v1/auth/me	Отримання даних поточного користувача
PUT	/api/v1/profile	Оновлення профілю
GET	/api/v1/chat/	Отримання списку чатів
POST	/api/v1/chat/	Створення нового чату

Для підтримки миттєвого обміну повідомленнями використовується бібліотека Socket.IO, яка встановлює постійне двонаправлене WebSocket-з’єднання між клієнтом і сервером. Обробка ключових подій наведена нижче:

– connect – аутентифікація користувача через JWT при встановленні з’єднання;
– disconnect – обробка відключення користувача;
– join_conversation – приєднання користувача до певної чат-кімнати;
– leave_conversation – вихід із чат-кімнати;
– send_message – отримання повідомлення від клієнта, збереження його у базі та трансляція іншим учасникам;
– new_message – розсилка нових повідомлень підключеним клієнтам.

Обробка WebSocket-подій реалізована асинхронно завдяки інтеграції Flask-SocketIO з бібліотекою Eventlet, яка дозволяє обробляти тисячі одночасних з’єднань у неблокуючому режимі. Такий підхід дозволяє серверу підтримувати масштабованість без втрати продуктивності під навантаженням. Крім того, під час встановлення з’єднання сервер перевіряє валідність JWT-токена користувача та додає його до відповідної “кімнати” для отримання повідомлень.

Нижче на рисунку 3.2 можна побачити реалізацію події send_message.

```
@socketio.on("send_message")
    @require_live_user
    def on_send(data):
        conversation_id = data.get("conversation_id")
        group_id = data.get("group_id")
        content = data.get("content")
        if (not conversation_id and not group_id) or not content:
            emit("error", {"message": "conversation_id or group_id and content are required"})
            return
        try:
            token = request.environ.get("jwt")
            payload = decode_token(token, allow_expired=True)
            user_id = payload["sub"]
            if conversation_id:
                message = Message.create_message(
                    conversation_id=conversation_id,
                    sender_id=user_id,
                    content=content.strip()
                )
                room = f"conversation_{conversation_id}"
            elif group_id:
                message = Message.create_message(
                    group_id=group_id,
                    sender_id=user_id,
                    content=content.strip()
                )
                room = f"group_{group_id}"
            if not message:
                emit("error", {"message": "Failed to create message"})
                return
            message_data = message.to_dict()
            emit("new_message", message_data, room=room)
        except Exception as e:
            emit("error", {"message": "Failed to send message"})
```

Рисунок 3.2 – Код події send_message

В системі реалізовані надійні механізми безпеки. Автентифікація користувачів здійснюється за допомогою JWT з використанням короткоживучих доступ токенів та довгоживучих токенів оновлення, що дозволяє безперервно підтримувати сесію користувача. Access-токени живуть 1 хвилину, у той час, як Refresh-токени 7 днів – така система була розроблена для постійної фонової перевірки користувача та доступних йому функцій. WebSocket-з'єднання також захищені шляхом перевірки JWT при підключенні.

3.2 Реалізація бази даних

У онлайн-чаті Blyx для зберігання інформації використовується документно-орієнтована база даних MongoDB, яка належить до категорії NoSQL баз даних.

Для роботи з MongoDB у Python використовується MongoEngine – об'єктно-документний маппер (ODM), який полегшує роботу з базою, надаючи зручний інтерфейс для створення моделей даних і виконання запитів.

Для зручної роботи з базою даних також використовувався сервіс MongoDB Compass. На рисунку 3.3 можна побачити, як виглядають записи спілкувань у БД.



Рисунок 3.3 – Записи спілкування між користувачами у базі даних

Створено різні моделі даних за допомогою MongoEngine. Наприклад, нижче на рисунку 3.4 можна побачити фрагмент коду моделі Group – модель групових чатів із підтримкою списку учасників, інформації про групу та керування доступом.

```
class Group(Document):
name = StringField(required=True, max_length=100)
description = StringField(max_length=500)
creator = ReferenceField(User, required=True)
members = ListField(ReferenceField(User))
admins = ListField(ReferenceField(User))
created = DateTimeField(default=lambda: dt.datetime.now(timezone.utc))
last_message_at = DateTimeField(default=lambda: dt.datetime.now(timezone.utc)) is_active =
BooleanField(default=True)
   meta = {
      'indexes': [
         'creator',
         'members',       'last_message_at',
         'is_active'
      ]
```

Рисунок 3.4 – Лістинг коду моделі Group

Цю модель бази даних з клієнтом пов’язує API маршрут який відповідає за створення нових груп у системі. Цей маршрут отримує запит від клієнта з даними про нову групу, включаючи назву, опис і початковий список учасників. Після валідації вхідних даних маршрут викликає відповідний метод моделі Group для створення запису в базі даних.

3.3 Реалізація клієнтської частини

Клієнтська частина онлайн-чату Blyx побудована за сучасними підходами та технологіями, які забезпечують зручний, швидкий і адаптивний інтерфейс для користувачів.

3.3.1 Архітектура та технології

Основою клієнтської частини є фреймворк Vue.js, який є одним із найпопулярніших інструментів для створення інтерактивних вебзастосунків.

Vue.js дозволяє розробникам створювати компонентний інтерфейс, де кожна частина UI – це окремий компонент зі своєю логікою, шаблоном та стилями. Це допомагає підтримувати код у чистоті та легко масштабувати проєкт.

У застосунку Blyx використовується Composition API – сучасний спосіб написання компонентів у Vue 3, який робить код більш організованим і повторно використовуваним.

За допомогою Composition API легко керувати станом компонентів, відслідковувати реактивні дані та створювати логіку, яку можна підключати до різних частин застосунку.

Для маршрутизації (переходу між різними сторінками) використовується Vue Router – офіційна бібліотека Vue, яка дозволяє створювати чисту, швидку і гнучку навігацію без перезавантаження сторінки. Завдяки цьому користувач може перемикатися між сторінками реєстрації, чату, профілю тощо, при цьому відчуваючи роботу звичайного десктопного застосунку. Це дозволяє додатку працювати дуже швидко та коректно.

Для управління станом (зберігання даних, які використовуються у багатьох компонентах, наприклад, інформації про користувача, список чатів чи повідомлень) застосовується Pinia – сучасна, легка і проста у використанні альтернатива Vuex.

Pinia дозволяє централізовано зберігати і оновлювати дані, що робить роботу із станом більш передбачуваною і зручною.

Для оформлення інтерфейсу використовується Tailwind CSS – CSS-фреймворк, який надає безліч готових класів для швидкого стилізування елементів. Це значно прискорює розробку та дозволяє створювати адаптивний дизайн, який однаково добре виглядає на мобільних пристроях і десктопах.

3.3.2 Основні компоненти

Застосунок складається з набору окремих компонентів Vue.js, кожен із яких відповідає за певну функціональність та частину інтерфейсу. Такий поділ дозволяє зробити код організованим, простим у підтримці та повторному використанні.

Компонент LoginPage.vue відповідає за сторінку авторизації користувача, містить форму введення логіна і пароля, обробляє відправку даних на сервер через API для отримання JWT токена, а також управляє помилками, наприклад, у випадку неправильних даних.

Після успішної авторизації користувач перенаправляється на головну сторінку чату. На рисунку 3.5 можна побачити інтерфейс авторизації застосунку. Також у додатку А розміщений повний лістинг коду компоненту реєстрації.

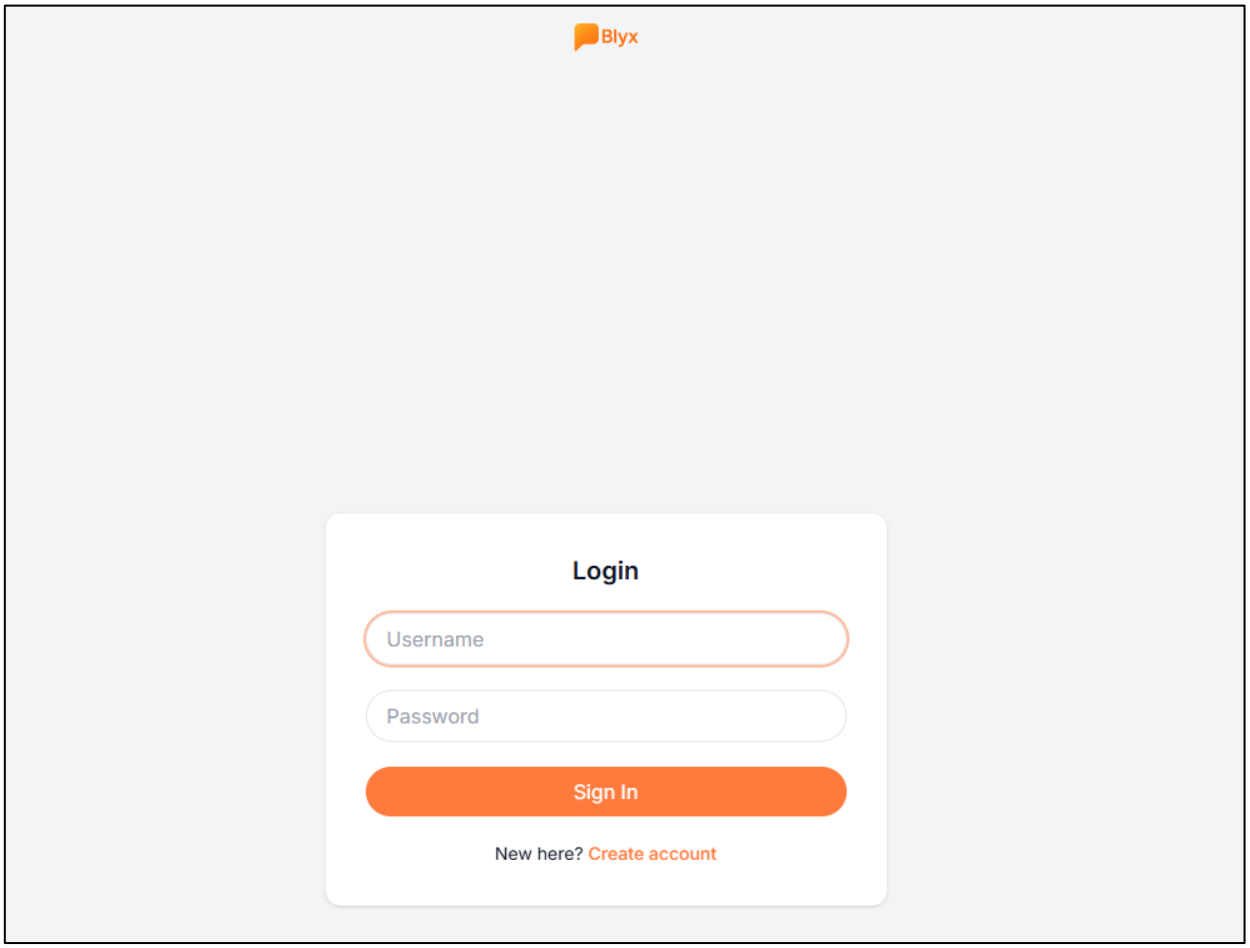


Рисунок 3.5 – Компонент авторизації у додатку

Основний інтерфейс спілкування реалізовано у компоненті ChatPage.vue. Він відображає список чатів та груп, у яких бере участь користувач, відкриває вибраний чат із повідомленнями, які завантажуються через API, і підключається до WebSocket для миттєвого прийому нових повідомлень. Для забезпечення стабільної роботи інтерфейсу чатів при великій кількості повідомлень реалізовано механізм віртуалізації повідомлень – лише частина з них рендериться одночасно. Це дозволяє знизити навантаження на DOM та підтримувати високу швидкодію навіть при тривалих чатах. В межах цього компонента розміщене поле введення тексту та кнопка відправлення повідомлення, а також підтримується автоматичне прокручування до останнього повідомлення. На рисунку 3.6 зображено основний компонент чату у процесі спілкування з іншим користувачем.

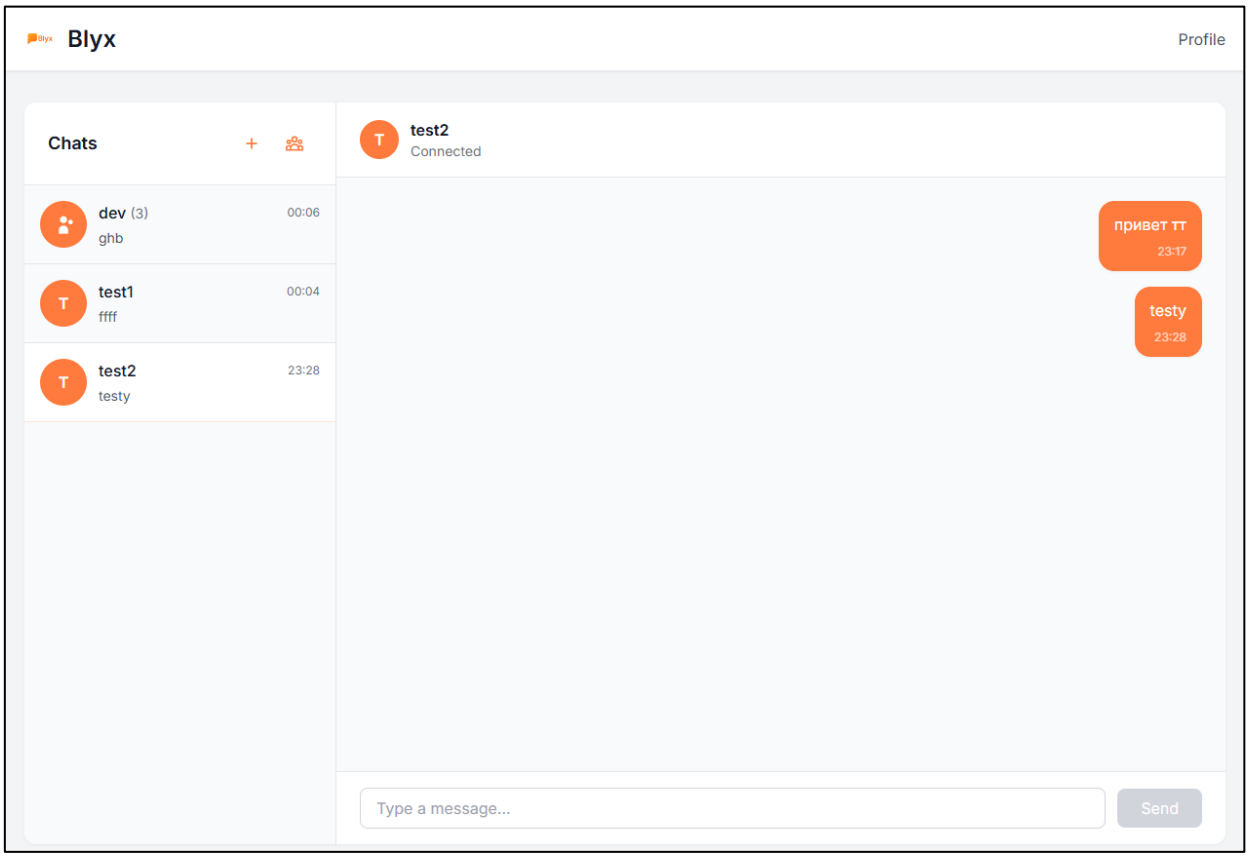


Рисунок 3.6 – Компонент чату

Профіль користувача реалізовано в компоненті ProfilePage.vue, де можна переглядати і редагувати особисті дані, такі як ім’я, електронна пошта,

аватар, тощо. Зміни відправляються на сервер через API, також доступна функція логауту з облікового запису. На рисунку 3.7 зображено цей елемент і його функціонал.

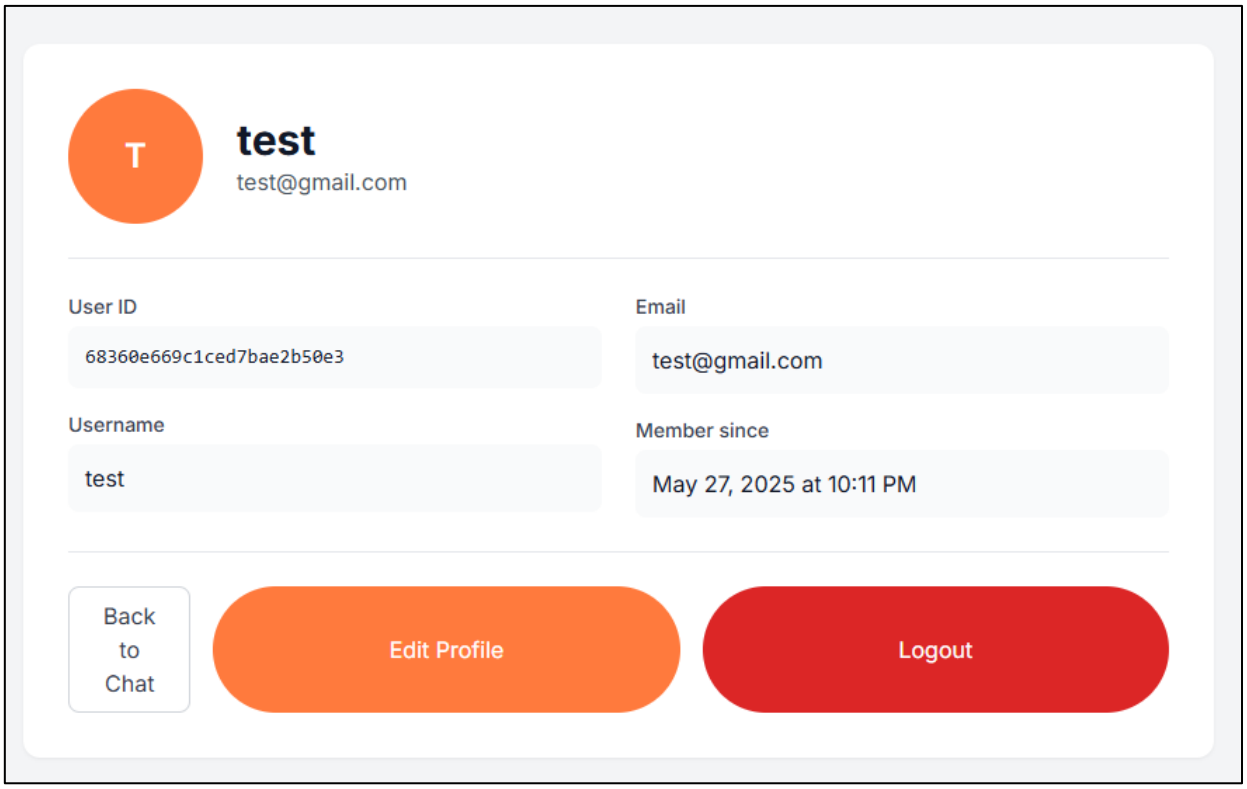


Рисунок 3.7 – Компонент профілю

Компоненти організовані ієрархічно, причому головний компонент ChatPage містить у собі список чатів і вікно повідомлень, а модальні вікна підключаються динамічно в залежності від дій користувача. Основні модальні вікна включають CreateGroupModal (створення групи), GroupSettingsModal (налаштування групи), UserSearchModal (пошук користувачів) та EditProfileModal (редагування профілю), тощо. Нижче, на рисунку 3.8 можна побачити рисунок з лістингом основної функції створення групи у модальному вікні CreateGroupModal.

```
const createGroup = async () => {
  if (!groupName.value.trim()) {
    toast.error('Group name is required')
```

```
    return
  }
  isCreating.value = true
  try {
    const { data } = await api.post('/groups', {
      name: groupName.value.trim(),
      description: groupDescription.value.trim(),
      members: selectedMembers.value.map(member => member.id)
    })
    toast.success('Group created successfully')
    emit('groupCreated', data.group)
    closeModal()
  } catch (error) {
    console.error('Failed to create group:', error)
    if (error.response?.data?.error) {
      toast.error(error.response.data.error)
    } else {
      toast.error('Failed to create group')
    }
  } finally {
    isCreating.value = false
  }
}
```

Рисунок 3.8 – Основна функція створення групи

3.3.3 Взаємодія з сервером та управління станом

Основним механізмом передачі даних між фронтендом і бекендом є REST API, що забезпечує стандартні HTTP-запити для отримання, створення, оновлення та видалення ресурсів, таких як користувачі, чат-кімнати, повідомлення, групи, тощо. Для реалізації цих запитів на стороні клієнта використовується бібліотека Axios, яка дозволяє зручно виконувати асинхронні HTTP-запити з налаштуванням заголовків, включаючи JWT-

токени для аутентифікації. Всі запити містять у заголовку токен доступу, що дозволяє безпечно ідентифікувати користувача та обмежувати доступ до захищених ресурсів. Підключення бібліотеки Axios можна побачити нижче на рисунку 3.9.

```
import axios from 'axios'
import { useAuth } from '../stores/auth'
const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL
})
api.interceptors.request.use(config => {
  const { token } = useAuth()
  if (token) config.headers.Authorization = `Bearer ${token}`
  return config
})
export default api
```

Рисунок 3.9 – Підключення бібліотеки Axios

Управління станом у фронтенд-застосунку здійснюється за допомогою Pinia. Завдяки Pinia зберігаються актуальні списки чатів, активні повідомлення, інформація про користувача та статус підключення.

Для синхронізації стану застосунку з WebSocket-з’єднанням реалізовано двосторонній потік даних: коли клієнт надсилає нове повідомлення, воно одразу додається до локального стану addMessage(), а потім підтверджується сервером і транслюється іншим користувачам.

Така архітектура дозволяє уникнути затримок у відображенні повідомлень та забезпечити максимально реалістичний чат-досвід у реальному часі.

Далі на рисунку 3.10 наведено частковий код роботи системи станів Pinia у компоненті чату.

```
import { defineStore } from 'pinia'
export const useChat = defineStore('chat', {
 state: () => ({
  conversations: [],
  currentConversation: null,
  messages: [],
  isConnected: false
 }),
 actions: {
  setConversations(conversations) {
   this.conversations = conversations
  },
   setCurrentConversation(conversation) {
   if (!this.currentConversation || this.currentConversation.id !== conversation?.id) {
    this.messages = []
   }
   this.currentConversation = conversation
  },
  setMessages(messages) {
   this.messages = messages
  },
  addMessage(message) {
   this.messages.push(message)
  },
  setConnectionStatus(status) {
   this.isConnected = status
  },   updateConversationLastMessage(conversationId, message) {
   const conversation = this.conversations.find(c => c.id === conversationId)
   if (conversation) {
    conversation.last_message_at = message.created_at
    conversation.last_message = message.content
    this.conversations.sort((a, b) => new Date(b.last_message_at) - new
Date(a.last_message_at))
   }
  },
```

Рисунок 3.10 – Частковий код роботи системи станів Pinia

3.4 Навантажувальне тестування застосунку

Навантажувальне тестування застосунку Blyx було проведено вручну шляхом одночасного входу до системи з кількох акаунтів, що дозволило оцінити стабільність роботи серверної та клієнтської частин під реальним використанням.

Для тестування було створено декілька тестових облікових записів, з яких здійснювалися паралельні дії: надсилання повідомлень у приватних чатах, створення та спілкування в групах, зміна налаштувань користувача. Такий підхід давав змогу імітувати поведінку реальних користувачів та перевірити коректність роботи системи при одночасній активності.

Під час тестування були перевірені ключові аспекти:

- моментальна доставка повідомлень у приватних чатах та групах без затримок;
- коректне оновлення інтерфейсу користувача в режимі реального часу;
- стабільність підключення WebSocket – відсутність випадкових відключень або втрат повідомлень;
- функціональність створення, редагування та видалення груп, а також керування учасниками;
- робота аутентифікації та авторизації при одночасних сесіях;
- перевірка коректної роботи всіх CRUD операцій з чатами і повідомленнями через REST API.

Моментальну доставку повідомлень у приватних чатах та групах без затримок продемонстровано на рисунку 3.11.

Перевірку функціональності створення, редагування та видалення груп, а також керування учасниками, продемонстровано на рисунку 3.12.

Для забезпечення коректності та безпеки введених користувачами даних у застосунку Blyx реалізовано систему валідації. У роутерах бекенду проводиться первинна валідація вхідних запитів: перевіряється наявність обов’язкових полів, довжина рядків, формат даних.

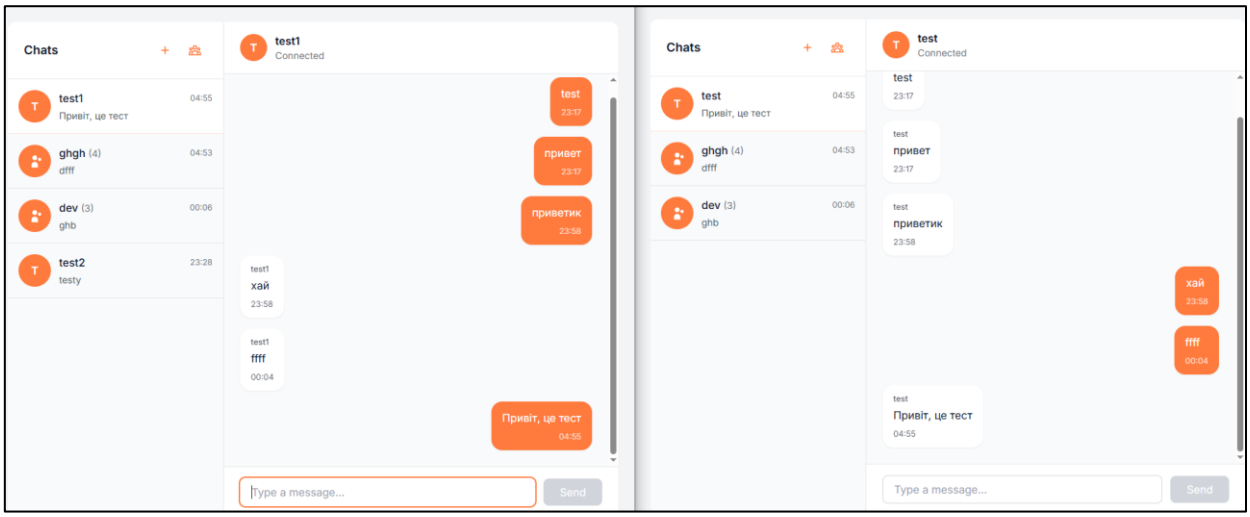


Рисунок 3.11 – Доставка повідомлень у двох паралельних чатах

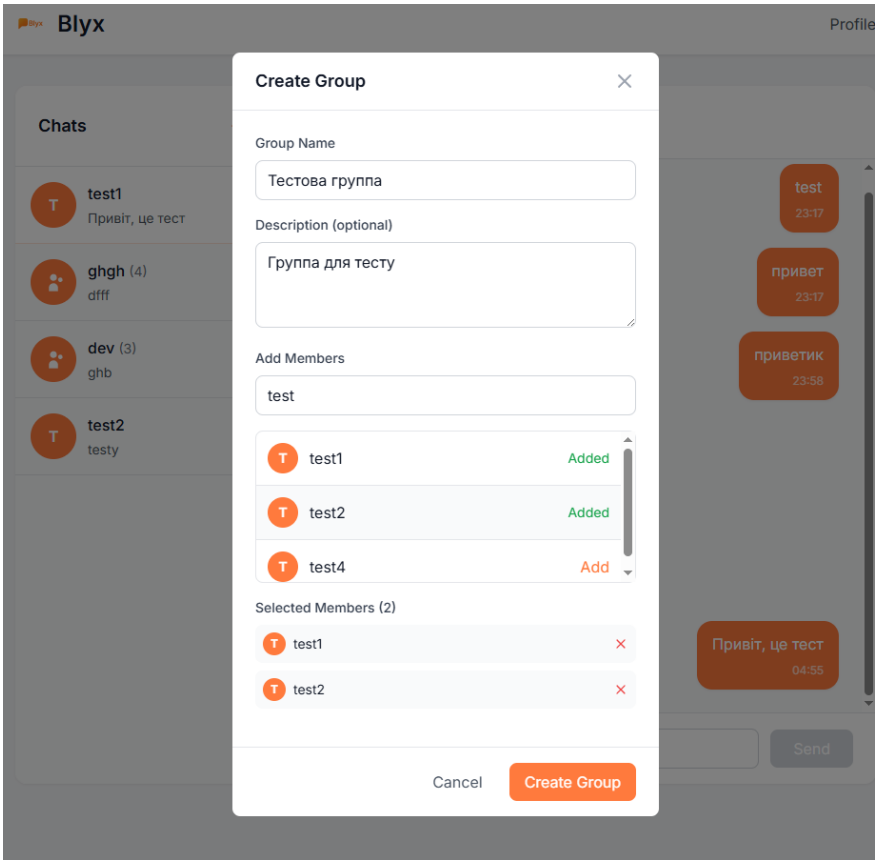


Рисунок 3.12 – Створення нової групи

Наприклад, у роуті створення групи перевіряється, чи вказано ім’я групи, чи не перевищує воно максимальну довжину, а також обмежується розмір опису. В разі некоректних даних сервер повертає відповідні помилки з HTTP кодами 400, що дозволяє клієнтській частині відобразити повідомлення користувачу. Один з прикладів валідації даних – неможливість створення

аккаунта з вже існуючим логіном чи поштою. Імплементацію цієї функції можна побачити на рисунку 3.13.

При зміні даних користувача у особистому кабінеті обов'язково необхідно ввести пароль – якщо цього не зроблено, то система не дозволяє цю дію. Таку поведінку можна побачити на рисунку 3.14.

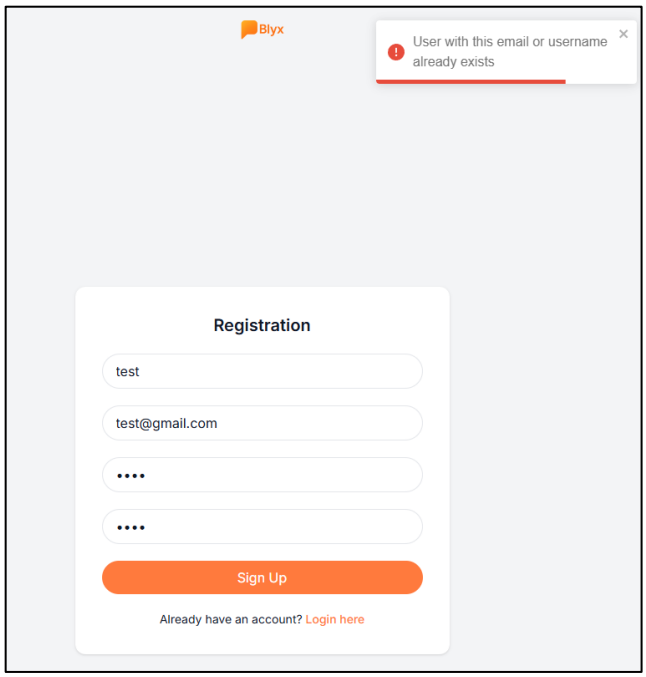


Рисунок 3.13 – Валідація форми реєстрації

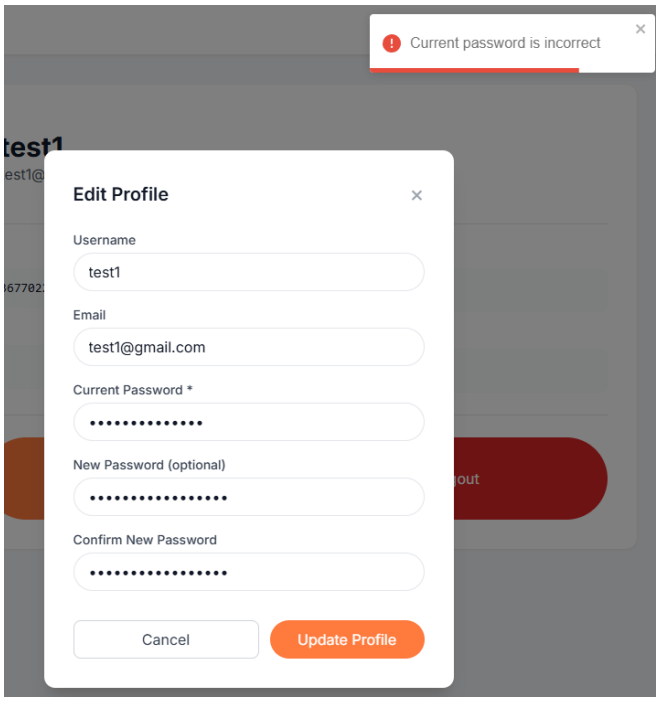


Рисунок 3.14 – Валідація форми зміни даних аккаунту

3.5 Висновки до розділу 3

У цій частині роботи було реалізовано серверну і клієнтську частини онлайн-чату Blyx із використанням Flask, Socket.IO та Vue.js. Забезпечено ефективну роботу real-time обміну повідомленнями, управління чатами та користувачами. Після проведення тестування підтверджено стабільність і коректність функціонування системи. Всі основні модулі працюють без помилок, а взаємодія клієнта з сервером відбувається швидко і надійно. Застосунок готовий до експлуатації та подальшого розвитку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було створено онлайн-чат Blyx, який базується на сучасних та ефективних технологіях, таких як Python, Flask, Socket.IO, MongoDB і Vue.js. Розробка охопила повний цикл створення вебзастосунку – від проєктування архітектури та моделювання даних до реалізації серверної та клієнтської частин. Серверна частина забезпечує надійний обмін повідомленнями в реальному часі, обробку аутентифікації, управління сесіями та збереження історії спілкування у базі даних MongoDB. Клієнтська частина, побудована на Vue.js, забезпечує інтуїтивно зрозумілий, адаптивний інтерфейс користувача з підтримкою реального часу через WebSocket-з’єднання.

Під час роботи було реалізовано функціонал приватних і групових чатів, а також керування профілями користувачів, що робить застосунок зручним для широкого кола користувачів. Для забезпечення безпеки даних використано JWT-аутентифікацію з механізмом оновлення токенів, а також валідацію даних. Навантажувальні тести, проведені вручну з одночасним підключенням кількох користувачів, показали стабільність роботи системи, коректність доставки повідомлень і відсутність помилок у роботі основних модулів.

Розроблений онлайн-чат є масштабованим та готовим до інтеграції з іншими системами або подальшого розширення функціоналу. Отримані результати підтверджують практичну цінність роботи та її відповідність сучасним вимогам до застосунків реального часу. До подальших удосконалень проєкту належать підтримка мультимедійних повідомлень, автоматизація тестування, розробка мобільної версії, покращення безпеки через багатофакторну аутентифікацію та впровадження системи сповіщень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Rouse M. Real-Time chat. 2014. *Techopedia.* URL: https://www.techopedia.com/definition/23690/real-time-chat (дата звернення: 20.02.2025).

2. Learning in Virtual Worlds: Research and Applications / ed. by S. Gregory et al. Athabasca University Press, 2016. 347 p. URL: https://doi.org/10.15215/aupress/9781771991339.01 (дата звернення: 20.02.2025).

3. Duarte F. Most Popular Messaging Apps. 2025. *Exploding Topics.* URL: https://explodingtopics.com/blog/messaging-apps-stats (дата звернення: 25.02.2025).

4. Most popular global mobile messenger apps as of February 2025, based on number of monthly active users. *Statista.* URL: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/ (дата звернення: 26.02.2025).

5. Online Chat Application Project in Software Development. *GeeksforGeeks.* URL: https://www.geeksforgeeks.org/online-chat-application-project-in-software-development/ (дата звернення: 03.03.2025).

6. Вступ до Python фреймворків: які вони бувають і для чого використовуються. *FoxmindEd.* URL: https://foxminded.ua/freimvorky-python/ (дата звернення: 10.03.2025).

7. Socket-IO для початківців. *Javascript.org.ua.* URL: https://javascript.org.ua/socket-io-dlya-pochatkivcziv-🔌 (дата звернення: 11.03.2025).

8. MongoDB Basics : Beginners Guide. *MongoDB.* URL: https://www.mongodb.com/resources/products/fundamentals/basics (дата звернення: 11.03.2025).

9. Vue.js Documentation. *Vue.js.* URL: https://vuejs.org/guide/introduction.html (дата звернення: 12.03.2025).

10. Діаграми UML для моделювання процесів і архітектури проекту. *Evergreen.* URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 20.03.2025).

11. Lakhwinder. Understanding WhatsApp Architecture. 2022. *HackerNoon.* URL: https://hackernoon.com/understanding-whatsapp-architecture (дата звернення: 12.03.2025).

ДОДАТОК А

Компонент реєстрації

```
<script setup>
import { ref } from 'vue'
import { useRouter } from 'vue-router'
import api from '@/utils/api'
import { toast } from 'vue3-toastify'
import AuthCard       from '@/components/AuthCard.vue'
import AuthInput      from '@/components/AuthInput.vue'
import PrimaryButton  from '@/components/PrimaryButton.vue'

const username = ref('')
const email = ref('')
const pass = ref('')
const confirm = ref('')
const busy = ref(false)
const router = useRouter()

async function submit() {
  if (pass.value !== confirm.value) {
    toast.error('Passwords do not match')
    return
  }
  busy.value = true
  try {
    await api.post('/auth/register', {
      username: username.value,
      email: email.value,
```

```
      password: pass.value
    })
    router.push('/login')
  } catch (e) {
    const msg = e.response?.data?.error || 'Registration failed'
    toast.error(msg)
  } finally {
    busy.value = false
  }
}
</script>

<template>
  <div class="flex flex-col items-center">
    <!-- Логотип приложения -->
    <div class="mb-8">
      <img src="/logo.png" alt="Blyx" class="h-16 w-auto" />
    </div>

    <AuthCard title="Registration">
      <AuthInput v-model="username" placeholder="Username" />
      <AuthInput v-model="email"    placeholder="Email" />
      <AuthInput v-model="pass"     type="password" placeholder="Password" />
          <AuthInput   v-model="confirm"    type="password"   placeholder="Confirm
Password" />

      <PrimaryButton :disabled="busy" @click="submit">
        Sign Up
      </PrimaryButton>
```

```
      <p class="text-center text-sm mt-2">
        Already have an account?
                <RouterLink to="/login" class="text-primary font-medium">Login here</RouterLink>
      </p>
    </AuthCard>
  </div>
</template>
```