Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка програмного забезпечення платформи для онлайн-комунікацій

Виконав: студент 4 курсу групи ЗПІ-21б
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Павленко М. І.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Майданюк В. П.
(прізвище та ініціали)

Рецензент: д.ф., ст. викл. каф. ОТ Обертюх М. Р.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н.
«9» 06 2025 р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Павленку Максиму Ігоровичу

1. Тема роботи – Розробка програмного забезпечення платформи для онлайн-комунікацій.

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доц. каф. ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: середовище розробки – WebStorm, кольоровий режим – TrueColor, розмір екрану – не менше 1920 на 1080, мови розробки – Javascript, Typescript, Node.js; фреймворки – React, Next, Express; операційна система – Windows 8/10/11; браузер – на базі двигуна V8 і Chromium 110.

4. Зміст текстової частини: вступ, аналіз стану розвитку програм для комунікацій в системах онлайн-спілкування та постановка задач розробки, розробка методу, моделі та алгоритмів роботи програми, розробка програмного забезпечення, тестування програми, висновки, список використаних джерел, додатки, ілюстративна частина.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науковий керівник бакалаврської кваліфікаційної роботи, тема; актуальність розробки; мета, об'єкт та предмет дослідження; постановка задачі; наукова новизна; практична цінність; огляд та аналіз аналогів; модель роботи системи, загальний алгоритм роботи системи; діаграми станів; схеми інтерфейсу; Блок-схема алгоритму відправки запрошення у друзі з real-time; використані технології; функціонал розробленої системи; апробація та публікація результатів роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Майданюк В. П. к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку програм для комунікацій в системах онлайн-спілкування та постановка задач розробки	25.03.25 – 31.03.25	вик.
2	Розробка методів, моделі та алгоритмів роботи програми	01.04.25 – 14.04.25	вик.
3	Розробка програмного забезпечення	15.04.25 – 12.05.25	вик
4	Тестування програми	13.05.25 – 15.05.25	вик
5	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25	вик.

Студент _____ **Павленко М. І.**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Майданюк В. П.**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.42

Павленко М. І. Розробка програмного забезпечення платформи для онлайн-комунікацій: бакалаврська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. – Вінниця: ВНТУ, 2025. – 66 с. – На укр. мові. Бібліогр. : 22 назв ; рис. : 58 ; табл. : 5.

У бакалаврській кваліфікаційній роботі проведено аналіз сучасного стану та тенденцій розвитку програмних рішень, призначених для онлайн-комунікації. В межах дослідження проаналізовано існуючі програмні продукти, визначено їхні ключові переваги та недоліки. На основі отриманих результатів обґрунтовано доцільність розробки власної програмної платформи для онлайн-спілкування.

Виконано обґрунтування вибору технологічного стеку, що включає мову програмування, середовище розробки та інструменти для створення клієнтської та серверної частин системи. Розроблено кросплатформний вебзастосунок, клієнтська частина якого реалізована з використанням мови TypeScript і фреймворку Next.js, а серверна частина – за допомогою Node.js з використанням фреймворку Express.js, який призначений для створення RESTful API. У якості середовища розробки використано WebStorm.

Отримані результати можуть бути використані для підвищення ефективності процесів онлайн-комунікації у сферах бізнесу, освіти, а також у будь-яких інших галузях, де необхідні надійні засоби віддаленої взаємодії.

Ключові слова: вебзастосунок, онлайн-комунікація, TypeScript, Next.js, Node.js, WebRTC, інтерфейс користувача.

ABSTRACT

UDC 004.42

Pavlenko M. I. Development of software of a platform for online communication: bachelor's thesis in the specialty 121-software engineering, educational program-software engineering. - Vinnitsa: VNTU, 2025. 66 p. In Ukrainian speech. Bibliogr. : 22 titles; Fig. : 58; table : 5.

The bachelor's thesis has analyzed the current state and trends of software development, intended for online communication. The study analyzes existing software products, identifies their key advantages and disadvantages. On the basis of the results obtained, the feasibility of developing your own software platform for online communication is substantiated.

The selection of technological stacks, including programming language, development environment and tools to create the client and server parts of the system, is justified. A cross -platform web application has been developed, the client part of which is implemented using the Typescript language and the Next.JS framework and the server part is using Node.js and Express. Webstorm was utilized as a development environment.

The results can be used to increase the efficiency of online communication processes in business, education, and in any other areas that require reliable means of remote interaction.

Keywords: webapplication, online communication, Typescript, next.js, node.js, Webrtc, user interface.

ЗМІСТ

ВСТУП.....4
1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМ ДЛЯ КОМУНІКАЦІЙ В СИСТЕМАХ ОНЛАЙН-СПІЛКУВАННЯ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ.....8
1.1 Аналіз предметної області.....8
1.2 Аналіз аналогів розроблюваного програмного застосунку.....9
1.3 Аналіз методів розв’язання задачі.....18
1.5 Висновки.....20
2 РОЗРОБКА МЕТОДІВ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМИ.....22
2.1 Аналіз даних.....22
2.2 Розробка методу побудови вікон застосунку.....24
2.3 Розробка моделі системи.....25
2.4 Розробка загального алгоритму роботи системи та діаграм станів.....27
2.5 Висновки.....32
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....33
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення.....33
3.2 Розробка модуля для авторизації та реєстрації користувача.....37
3.3 Розробка модуля перегляду списку контактів.....40
3.4 Розробка модуля для здійснення відеодзвінків.....41
3.5 Розробка моделі системи.....43
3.6 Розробка структури інтерфейсу програмного застосунку.....44
3.7 Висновки.....54
4 ТЕСТУВАННЯ ПРОГРАМИ.....55
4.1 Аналіз методів тестування програмного забезпечення.....55
4.2 Тестування розробленого програмного застосунку.....56
4.3 Розробка інструкції користувача вебзастосунку.....62
4.4 Висновки.....63

ВИСНОВКИ 65
ДОДАТКИ 68
ДОДАТОК А (обов’язковий). Технічне завдання 69
ДОДАТОК Б (обов’язковий). Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень 72
ДОДАТОК В (довідниковий). Лістинг програми 73
ДОДАТОК Г (обов’язковий). Ілюстративна частина 121

ВСТУП

Обґрунтування вибору теми дослідження. Розвиток цифрових технологій суттєво впливає на способи комунікації в мережі, створюючи нові можливості для швидкого та ефективного обміну інформацією. Це забезпечує швидкий і зручний обмін інформацією між користувачами незалежно від їхнього місцезнаходження. Сучасні платформи для онлайн-спілкування дозволяють здійснювати текстові чати, голосові та відеодзвінки, що значно підвищує ефективність дистанційної взаємодії. Використання онлайн-комунікацій сприяє оптимізації робочих процесів, покращенню співпраці між командами та забезпечує можливість підтримувати зв’язок у будь-який час і з будь-якого пристрою. Це особливо важливо в умовах глобалізації і умовах вимушеної ізоляції (як-от карантин або війна) та зростання популярності віддаленої роботи, коли ефективне спілкування є ключовим фактором продуктивності [1].

Створення ефективної платформи для онлайн-комунікації потребує комплексного підходу, що включає використання сучасних вебтехнологій, реалізацію механізмів забезпечення безпеки даних, масштабованості та інтеграції з іншими сервісами. Важливим аспектом розробки є забезпечення високої стабільності роботи системи, а також інтуїтивно зрозумілого інтерфейсу для користувачів.

Попри наявність широкого спектра платформ для онлайн-спілкування, багато з них мають певні недоліки, серед яких:

- обмежена функціональність або складність у використанні;
- недостатній рівень захисту персональних даних;
- нестача підтримки інтеграції з іншими системами та сервісами;
- платний функціонал.

Інноваційні технології, такі як WebSockets для миттєвої передачі даних та WebRTC для організації відео- та аудіозв'язку, дозволяють створити ефективну, безпечну та зручну систему для користувачів.

Сучасні вебінтерфейси розробляються з урахуванням принципів зручності та інтуїтивної зрозумілості, що дозволяє користувачам швидко адаптуватися до роботи з ними без необхідності володіння спеціальними навичками [2]. Враховуючи зазначені задачі, розробка власної платформи для онлайн-комунікацій, що надаватиме доступність, кросплатформенність і гнучкість у використанні, є актуальним завданням.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення часу взаємодії, стабільності інтерактивного відео- та текстового спілкування, яке забезпечить користувачам широкий набір комунікаційних функцій, зручність використання та високий рівень безпеки.

Основними задачами дослідження є:

– провести аналіз інформаційного забезпечення системи;

– розробити метод, модель та алгоритм роботи системи;

– розробити систему для онлайн-комунікацій, що включає:

1) реєстрацію облікового запису для користувача;

2) можливість додати попередньо зареєстрованих друзів;

3) відстеження статусу активності друзів (в мережі, друкує повідомлення тощо);

4) функціонал відхилення і прийняття запрошень у друзі;

5) можливість надсилати текстові повідомлення і переглядати історію спілкування з користувачами;

6) функціонал для спілкування онлайн з використанням вебкамери і мікрофону пристрою користувачів;

7) функціонал інтернаціоналізації і локалізації.

Об’єкт дослідження - процес розробки програмного забезпечення для систем комунікацій в онлайн режимі.

Предмет дослідження - методи та програмні засоби платформи для онлайн-комунікацій.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, теорія баз даних, технології розробки програмного забезпечення для розробки платформи для онлайн-комунікацій; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод візуалізації, який, на відміну від існуючих, забезпечує можливість демонстрації робочого простору користувача, що покращує динамічні властивості взаємодії.

2. Розроблено універсальну модель онлайн-платформи, яка, на відміну від аналогів, поєднує інструменти для чатів, відеоконференцій та інтеграції з іншими сервісами.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби платформи для онлайн-комунікацій.

Особистий внесок здобувача. Усі результати, викладені в бакалаврській кваліфікаційній роботі, отримані автором особисто. У науковій праці [3], опублікованій у співавторстві, автору належать такі результати: таблиця порівняння функціональних характеристик аналогів, порівняння з аналогами, перелік розроблених функцій.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення бакалаврської кваліфікаційної роботи доповідалися та обговорювалися на науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2025 р.).

Публікації. Основні результати досліджень опубліковано в одній статті у матеріалах конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2025 р.).

1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМ ДЛЯ КОМУНІКАЦІЙ В СИСТЕМАХ ОНЛАЙН-СПІЛКУВАННЯ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз предметної області

Зі стрімким розвитком цифрових технологій онлайн-комунікація стала невід'ємною частиною повсякденного життя, бізнесу, освіти та соціальної взаємодії. Системи для текстового, голосового та відеозв'язку розвиваються швидкими темпами, пропонуючи користувачам нові можливості для ефективного спілкування. Впровадження сучасних технологій, таких як WebRTC, хмарні сервіси та шифрування даних, сприяє підвищенню якості зв'язку, безпеки та стабільності роботи платформ [4].

Однією з ключових переваг онлайн-комунікаційних додатків є їхня доступність та мобільність. Завдяки вебдодаткам і хмарним технологіям користувачі можуть спілкуватися незалежно від свого місцезнаходження, що особливо важливо для віддаленої роботи, дистанційного навчання та міжнародного бізнесу. Однак розробка таких платформ є складним завданням, яке потребує ретельного аналізу вимог користувачів, інтеграції з базами даних, забезпечення високої продуктивності та масштабованості системи [5].

Багато компаній звертаються до спеціалізованих IT-команд для розробки онлайн-комунікаційних платформ, що відповідають їхнім потребам. Це дозволяє швидко впроваджувати рішення з підтримкою відеоконференцій, групових чатів, інтеграції з календарями та іншими сервісами. Успішні приклади таких платформ: Zoom, Tandem, Slack – демонструють важливість комплексного підходу до розробки: оптимізація навантаження на сервери, безпека даних та зручний користувацький інтерфейс [6].

Крім того, сучасні платформи онлайн-комунікацій часто використовують технології штучного інтелекту для покращення користувацького досвіду, такі як

автоматичне розпізнавання мовлення, переклад у реальному часі та інтелектуальні рекомендації контактів. Інтеграція з API сторонніх сервісів також відіграє важливу роль, дозволяючи користувачам отримувати єдиний комунікаційний простір для роботи та особистого спілкування.

Таким чином, розробка програмного забезпечення для онлайн-комунікацій є важливим напрямком у сфері інформаційних технологій. Вона сприяє оптимізації робочих процесів, покращенню взаємодії між людьми та забезпечує високий рівень доступності комунікацій.

1.2 Аналіз аналогів розроблюваного програмного застосунку

З пришвидшеним темпом розвитку цифрових технологій сфера онлайн-комунікацій зазнає значних змін, особливо у контексті автоматизації та оптимізації процесів взаємодії між користувачами.

Завдяки сучасним платформам для онлайн-комунікацій компанії, освітні заклади та інші організації можуть ефективніше організовувати свою роботу, покращувати віддалену взаємодію та підвищувати продуктивність завдяки швидкому та надійному обміну інформацією. У цьому контексті з’являється все більше вебзастосунків, які надають користувачам можливість здійснювати відеозв’язок, організовувати групові чати, обмінюватися файлами та інтегрувати зовнішні сервіси для зручнішої співпраці.

Серед численних аналогів розроблюваної системи варто виділити кілька ключових платформ: Zoom, Tandem та Slack, кожна з яких має свої унікальні особливості та переваги.

Zoom [7] (див. рисунок 1.1) – це одна з найпопулярніших платформ для відеоконференцій, створена компанією Zoom Video Communications. Вона пропонує зручне та ефективне рішення для організації онлайн-зустрічей, що робить її незамінною як для бізнес-користувачів, так і для освітніх установ та приватних осіб.

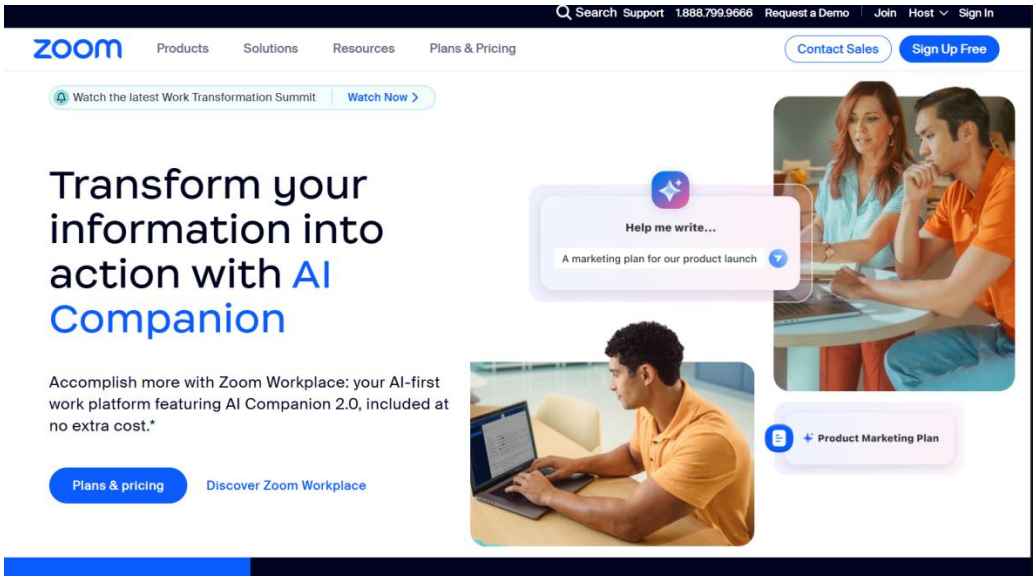

Рисунок 1.1 – Початкова сторінка застосунку Zoom

Завдяки інтуїтивно зрозумілому інтерфейсу та широкому набору функцій, користувачі можуть легко створювати та приєднуватися до відеоконференцій, використовуючи як вебверсію, так і мобільні додатки. Окрім стандартного відеозв'язку, платформа підтримує обмін файлами, чат, а також функцію демонстрації екрану, що особливо корисно для презентацій та віддаленого навчання (див. рисунок 1.2).

Рисунок 1.2 – Онлайн конференція в Zoom

Однією з головних переваг Zoom є можливість масштабування конференцій: платформа підтримує як індивідуальні дзвінки, так і великі вебінари з сотнями учасників. Крім того, додаток пропонує інструменти для запису зустрічей, що дозволяє зберігати важливу інформацію для подальшого перегляду, і багато інших налаштувань і персоналізацій (див. рисунок 1.3).

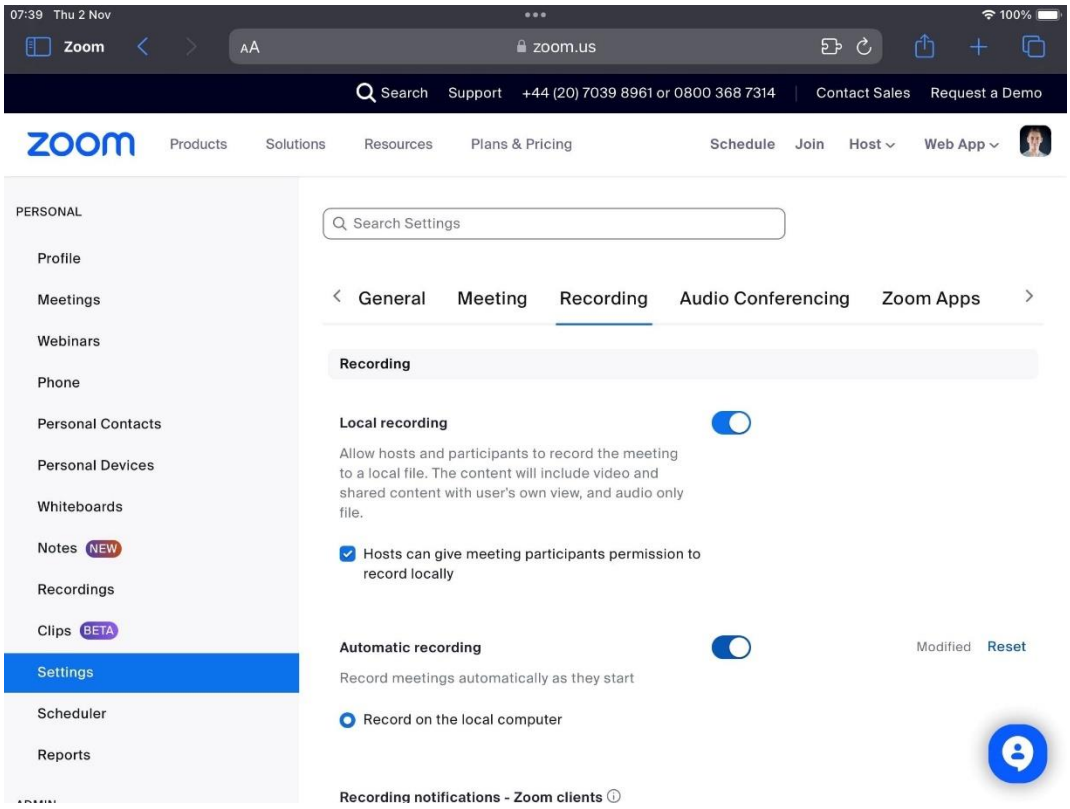

Рисунок 1.3 – Налаштування в Zoom

Завдяки цим можливостям Zoom став одним із ключових гравців на ринку онлайн-комунікацій, забезпечуючи надійний та зручний інструмент для взаємодії в режимі реального часу.

Tandem [8] (див. рисунок 1.4) – це платформа для голосового та відеозв'язку, що орієнтована на віддалену співпрацю команд у режимі реального часу. Вона дозволяє швидко підключатися до розмови, ніби всі учасники працюють в одному офісі, забезпечуючи доступ до спільного робочого простору.

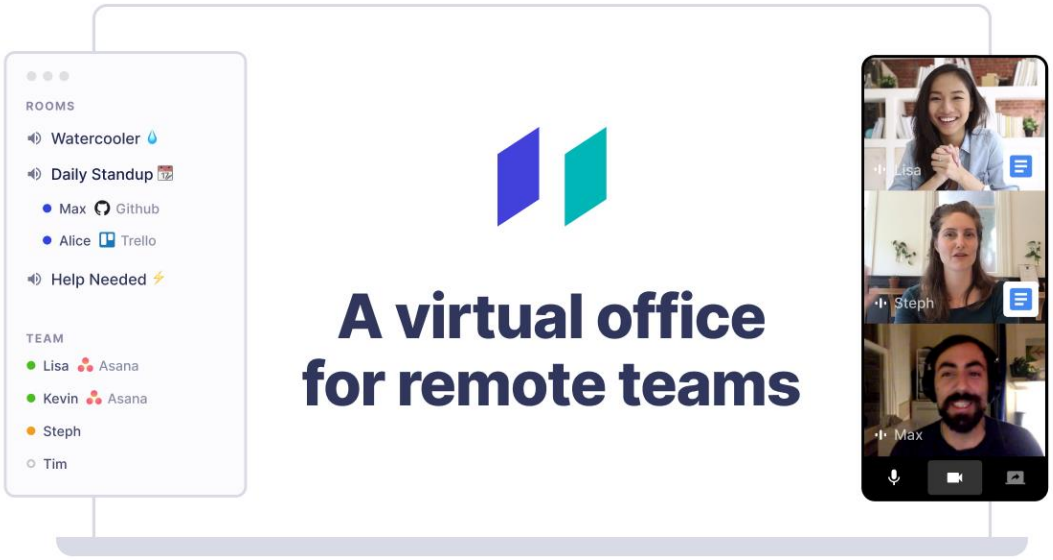

Рисунок 1.4 – Приклад конференції Tandem

Завдяки гнучкій інтеграції з популярними сервісами, такими як Spotify, Google Drive і Trello (див. рисунок 1.5), Tandem значно спрощує роботу в команді, об'єднуючи всі необхідні інструменти в одному місці. Користувачі можуть бачити, над чим працюють їхні колеги в реальному часі, а також приєднуватися до голосових чи відеодзвінків одним кліком.

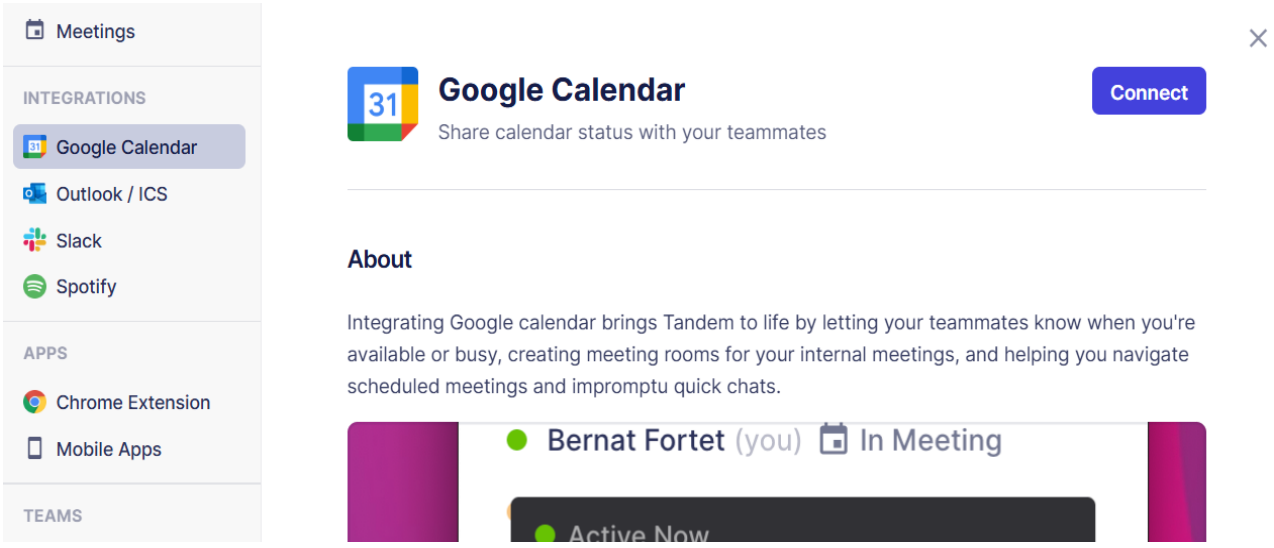

Рисунок 1.5 – Приклад інтеграцій в Tandem

Особливістю Tandem.chat є можливість створення тематичних кімнат, які дозволяють командам організовувати спільні зустрічі за напрямками роботи або проєктами (див. рисунок 1.6). Це сприяє ефективнішій комунікацій, зменшуючи потребу в зайвих текстових повідомленнях та електронних листах, а також дозволяє учасникам швидше знаходити інформацію та вирішувати робочі питання в межах спеціалізованих груп. Такі кімнати також забезпечують зручний простір для обміну файлами та ідеями, що покращує співпрацю в реальному часі.

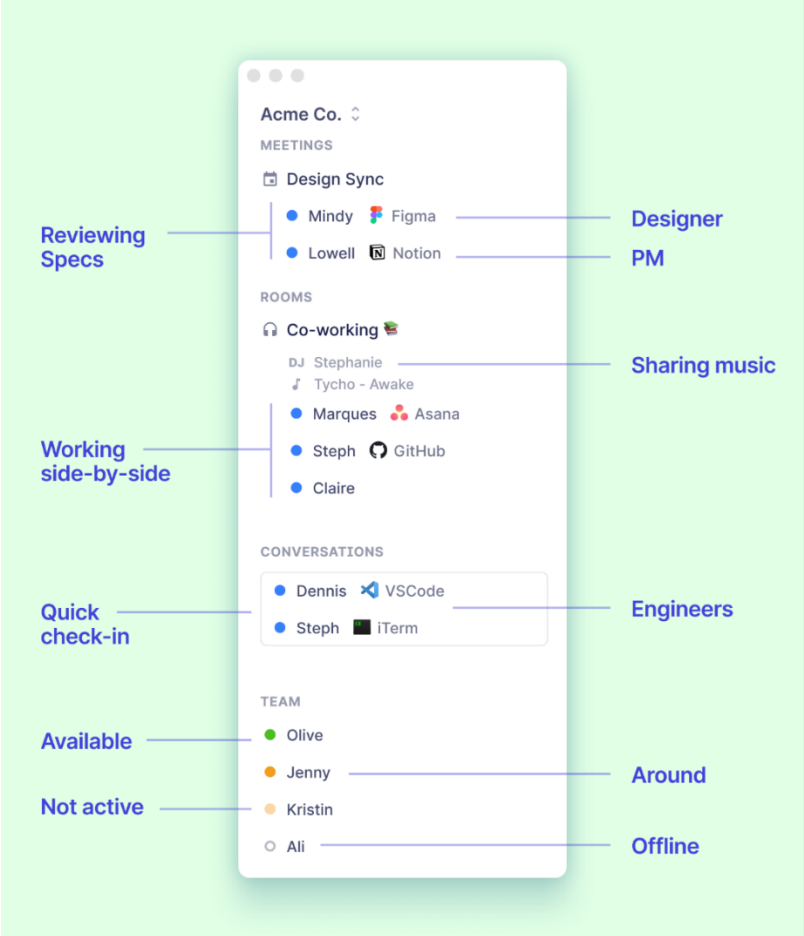

Рисунок 1.6 – Легенда позначень в інтерфейсі Tandem

Завдяки цим функціям Tandem.chat є потужним інструментом для розподілених команд, який сприяє продуктивному спілкуванню та координації в робочому процесі.

Slack [9] (див. рисунок 1.7-1.8), з іншого боку, є популярною платформою для комунікацій та співпраці в командах, яка забезпечує ефективний обмін інформацією в реальному часі. Користувачі можуть створювати канали для різних проєктів або тем, обмінюватися повідомленнями, файлами, а також організовувати голосові та відео дзвінки.

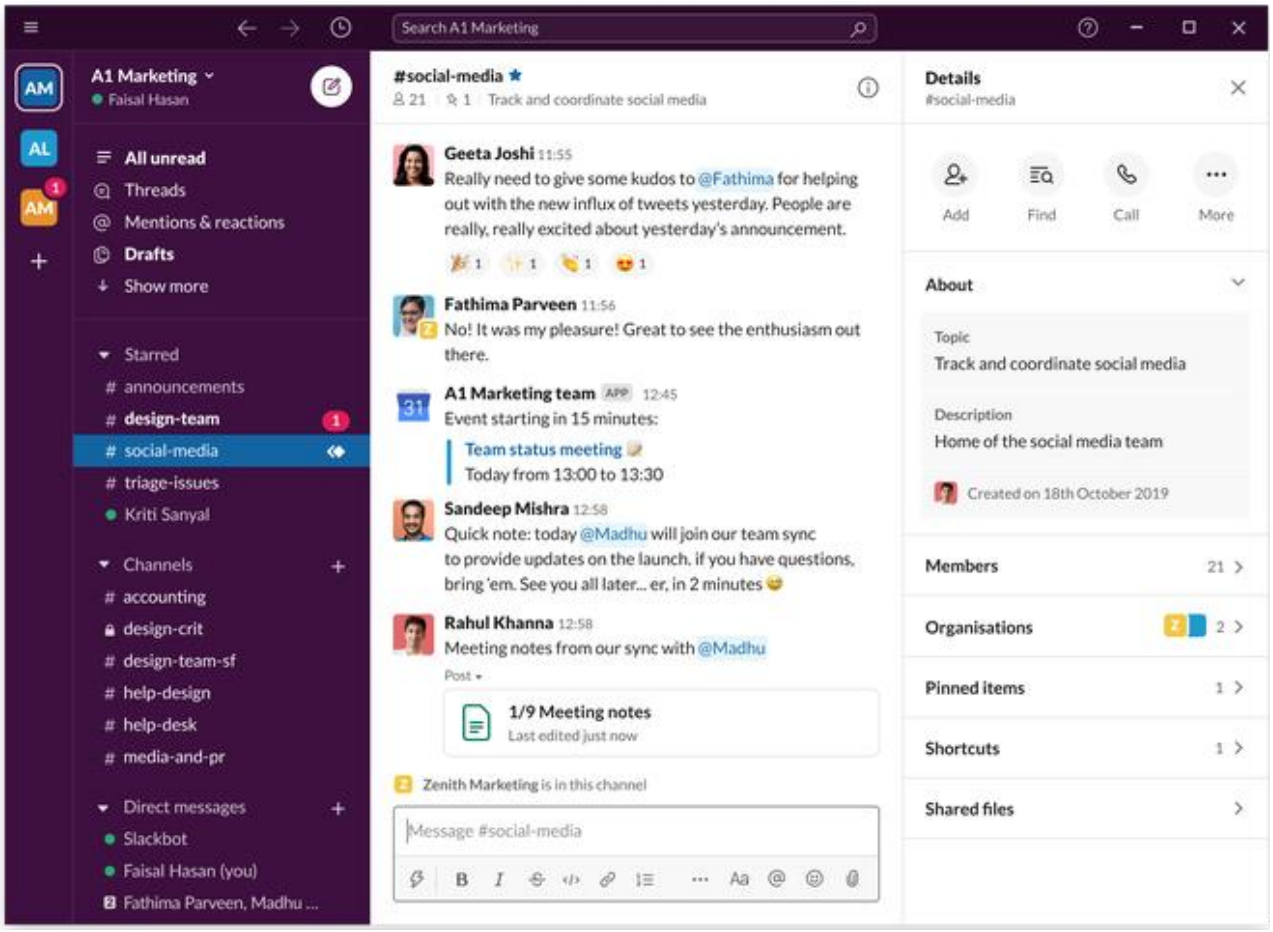

Рисунок 1.7 – Чат в Slack

Використання інтеграцій з іншими сервісами дозволяє ще більше оптимізувати робочі процеси та зберігати всі необхідні дані в одному місці, що робить Slack незамінним інструментом для корпоративної комунікацій. Такий підхід не лише підвищує продуктивність, але й створює комфортні умови для ефективної співпраці всередині команд.

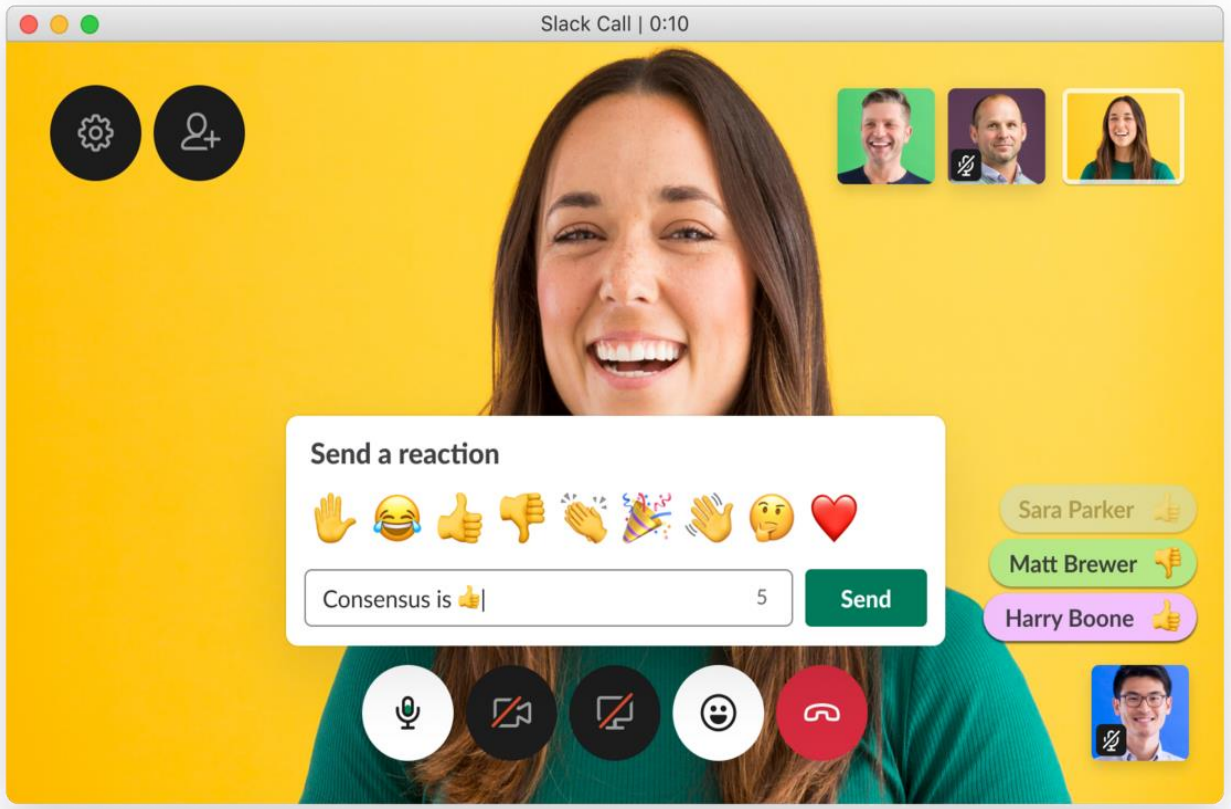

Рисунок 1.8 – Відео дзвінок в Slack

Ці платформи демонструють важливість онлайн-комунікацій у сучасному робочому середовищі, забезпечуючи командам гнучкі та ефективні інструменти для співпраці. Вони дозволяють користувачам швидко обмінюватися інформацією, організовувати відеозустрічі, інтегрувати сторонні сервіси та підвищувати загальну продуктивність роботи.

Крім того, такі платформи активно впроваджують інноваційні технології, такі як штучний інтелект для автоматизації завдань та аналітики даних, що робить їх ще більш привабливими для бізнесу.

Розробка подібних систем потребує глибокого розуміння потреб користувачів і сучасних технологічних рішень для створення надійних і функціональних платформ. Враховуючи досвід і можливості розглянутих аналогів, була складена порівняльна таблиця (див. таблиця 1), що дозволяє оцінити їхній функціонал у порівнянні з власною розробкою.

Таблиця 1 – Порівняльні характеристика онлайн платформ

Порівняльна характеристика	Zoom	Tandem	Slack	Власна розробка
Аутентифікація	1	1	1	1
Оновлення інформації в реальному часі	1	1	0	1
Підтримка технологій ШІ	1	0	1	1
Інтернаціоналізація	1	0	0	1
Інтеграція зі сторонніми сервісами	0	1	1	1
Сумарний коефіцієнт	4	4	3	5

Відповідно до наведеної таблиці порівняльного аналізу, розробка власної онлайн-платформи для комунікацій є виправданою та має значні переваги порівняно з існуючими рішеннями, такими як Zoom, Tandem і Slack. Запропонована система включатиме всі ключові функції конкурентів, але водночас забезпечить унікальні можливості, яких бракує в аналогах, зокрема підтримку інтернаціоналізації для всіх користувачів та оновлення інформації в реальному часі.

На відміну від Tandem, який не має підтримки функцій ШІ, або Slack, що не підтримує оновлення інформації в реальному часі, власна розробка інтегрує ці функціональні можливості в єдину систему. Крім того, система підтримуватиме інтернаціоналізацію та інтеграцію зі сторонніми сервісами, що забезпечить гнучкість використання у різних сферах по всьому світу, включаючи бізнес, освіту та інші види онлайн-співпраці.

Аналізуючи дані з таблиці 1, можна відзначити, що власна розробка демонструє на 20% кращі результати, ніж Tandem і Zoom (100% - 4/5 * 100% = 20%), і на 40% ефективніша у порівнянні з Slack (100% - 3/5 * 100% = 40%). Це підкреслює її конкурентні переваги та потенціал на ринку онлайн-комунікацій.

Таким чином, розробка та впровадження власної платформи для онлайн-комунікації має всі шанси на успіх. Вона пропонуватиме ефективне рішення для організацій та користувачів, які прагнуть зручного та надійного інструменту для спілкування та співпраці. Інвестування в подібні розробки є стратегічно важливим кроком, що дозволить підвищити якість онлайн-спілкування, оптимізувати робочі процеси та забезпечити конкурентоспроможність у динамічному світі цифрових технологій.ap

Отже, розробка програмного забезпечення для онлайн-комунікацій надає користувачам та організаціям значні переваги. Використання таких систем дозволяє не лише спростити процес взаємодії між людьми, але й підвищити ефективність робочих процесів, зменшити час на обмін інформацією та покращити загальну організацію командної роботи. З огляду на потреби сучасного бізнесу, освіти та віддаленої співпраці, компанії мають унікальну можливість використовувати інноваційні комунікаційні платформи для оптимізації своєї діяльності.

Розуміючи специфічні потреби онлайн-комунікацій та застосовуючи передові технології при розробці програмного забезпечення, ІТ-компанії можуть створювати інтегровані, надійні та легкі у використанні платформи для відеозв’язку, текстового спілкування, обміну файлами та спільної роботи над проєктами. Такі системи не тільки спрощують комунікацію між користувачами, але й надають організаціям зручні інструменти для керування співпрацею, планування завдань та аналізу ефективності командної роботи. Використання спеціалізованих платформ для онлайн-комунікацій відкриває нові можливості для

бізнесу, освіти та інших сфер, покращує доступність інформації та сприяє підвищенню продуктивності користувачів.

У підсумку, розробка та впровадження ефективних систем для онлайн-комунікацій є ключовим фактором успіху для компаній та організацій, які прагнуть підвищити якість співпраці та задоволеність своїх користувачів. Інвестування в розвиток таких рішень є стратегічним кроком на шляху до оптимізації комунікаційних процесів та забезпечення високої конкурентоспроможності на ринку цифрових технологій. У зв'язку з об'єктивними обставинами, як-от пандемії і війни, важливо слідкувати за новітніми тенденціями та інноваціями в цій області, щоб надалі розвиватись і залишатись конкурентоспроможним на світовому ринку цифрових послуг.

1.3 Аналіз методів розв'язання задачі

Розробка програмного забезпечення для онлайн-комунікацій вимагає ретельного вибору технічних підходів, щоб забезпечити високу якість зв'язку, безпеку та масштабованість. Існує кілька стратегій, кожна з яких має свої переваги та недоліки, що впливають на загальну ефективність та продуктивність системи. У цьому розділі розглянуто основні методики створення програмних модулів для таких платформ.

Один із підходів передбачає створення універсального API або бібліотеки, яка дозволяє легко інтегрувати комунікаційні функції (відеозв'язок, чат, інтеграції) у існуючі корпоративні або навчальні системи. Цей метод забезпечує швидке впровадження нових можливостей, але може створити додаткові труднощі для розробників, які інтегруватимуть цей API у свої рішення. Основний недолік – необхідність адаптації під різні платформи, що ускладнює підтримку та може впливати на стабільність зв'язку.

Альтернативний підхід полягає у створенні окремого спеціалізованого сервісу, який працює як незалежна хмарна платформа для онлайн-комунікацій. Це

дозволяє легко інтегруватися з іншими сервісами через API, зменшуючи потребу в розробці з нуля. Таке рішення забезпечує гнучкість у виборі технологій, високу масштабованість та можливість використання штучного інтелекту для аналізу комунікацій (наприклад, розпізнавання голосу). Проте, створення такої хмарної платформи вимагає значних початкових інвестицій у розробку, інфраструктуру та забезпечення кібербезпеки.

Найбільш перспективним є підхід розробки повноцінної кросплатформної системи для онлайн-комунікацій, що включатиме всі ключові функції (чат, відеоконференції, інтернаціоналізація, інтеграція з іншими сервісами). Це дозволяє забезпечити повний контроль над функціональністю та безпекою, а також впроваджувати унікальні інновації (наприклад, автоматичні підсумки зустрічей або інтерактивні дошки для командної роботи). Основний недолік – значні витрати часу та ресурсів на розробку, тестування та підтримку, однак результат виправдовує ці інвестиції, оскільки продукт може легко масштабуватися та адаптуватися під різні потреби користувачів.

Після аналізу переваг та недоліків кожного з підходів було вирішено обрати стратегію розробки власної онлайн-платформи для комунікацій. Це дозволить створити ефективне рішення, яке відповідатиме потребам сучасних компаній та команд, забезпечуючи гнучкість у розширенні функціоналу та легкість у масштабуванні системи в майбутньому.

1.4 Постановка задач дослідження

Після аналізу переваг та недоліків існуючих платформ для онлайн-комунікацій було визначено ключові завдання для розробки платформи:

- визначити оптимальний метод організації комунікацій, що дозволить ефективно управляти зустрічами;

- створити інтуїтивно зрозумілий та легкий у використанні графічний інтерфейс націлений на вебплатформи, адаптований під різні типи пристроїв (мобільні телефони, планшети, ПК);
- розробити платформу, що інтегрує в себе всі необхідні функції для зручного зв’язку та співпраці, зокрема відеоконференції, чати та інші інструменти для командної роботи;
- провести всебічне тестування розробленої платформи, щоб забезпечити її надійність, ефективність та зручність використання в реальних умовах, включаючи високий рівень безпеки.

Ці завдання спрямовані на створення платформи, яка не лише задовольнить потреби сучасних користувачів у швидкому та зручному доступі до комунікаційних інструментів, але й надасть організаціям ефективні інструменти для оптимізації робочих процесів.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено аналіз існуючих онлайн-платформ для комунікацій, зокрема таких як Zoom, Tandem, Slack. Це дослідження дозволило виявити, що, хоча ці платформи є популярними завдяки їх зручності та функціональності, вони мають певні обмеження, які можуть бути усунені за допомогою розробки власного рішення. Зокрема, було відзначено відсутність можливості для інтернаціоналізації та інтеграції з іншими інструментами, а також деякі обмеження щодо взаємодії в реальному часі.

Порівняння з існуючими платформами підкреслило необхідність створення власної онлайн-системи, яка не лише інтегруватиме основні функції цих сервісів, але й внесе інновації, зокрема функції для співпраці в реальному часі, інтернаціоналізацію і підтримку функцій ШІ.

Загальний коефіцієнт для власної розробки, що дорівнює 5, підтверджує

значні переваги запропонованого рішення порівняно з іншими платформами, підкріплюючи його доцільність та інноваційний потенціал. Висновки з аналізу підтверджують, що розробка та імплементація власної платформи для онлайн-комунікацій має великий потенціал для вдосконалення якості роботи команд, покращення взаємодії користувачів і підвищення рівня задоволеності клієнтів. Це стане важливим кроком на шляху до оптимізації робочих процесів компаній та підвищення загальної ефективності їх діяльності.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМИ

2.1 Аналіз даних

Обробка та передача інформації є ключовими аспектами роботи сучасних онлайн-платформ. Платформа забезпечуватиме обмін повідомленнями, організацію відеозустрічей та інтеграцію із зовнішніми сервісами для зручного використання. Передача даних між користувачами здійснюється через захищене з'єднання, що забезпечує конфіденційність та безпеку комунікації.

Користувачі можуть створювати облікові записи, налаштовувати профілі та отримувати доступ до функцій платформи через вебінтерфейс або мобільний застосунок. Інформація про учасників спілкування, історію повідомлень та зустрічей зберігається у базі даних, що дозволяє швидко отримувати доступ до потрібних матеріалів та планувати взаємодію.

Модуль авторизації та реєстрації дозволяє користувачам створювати облікові записи та входити до системи. Вхідними даними є інформація, введена користувачем, така як ім'я, електронна пошта, пароль. Реалізовано можливість авторизації через сторонні сервіси, як-от Goole OAuth2. Користувачі можуть як реєструватись, так і авторизуватись, щоб отримати доступ до основних функцій платформи. Вихідними даними є графічний інтерфейс, що підтверджує успішну авторизацію, надає можливість персоналізації профілю та забезпечує доступ до функцій застосунку, таких як спілкування з друзями, зміна налаштувань профілю та перегляд історії спілкування.

Модуль обміну повідомленнями забезпечує обмін текстовими повідомленнями між користувачами у реальному часі. Після автентифікації користувач отримує доступ до списку своїх чатів, де може переглядати історію спілкування, надсилати повідомлення та отримувати відповіді від співрозмовників. Повідомлення передаються через протокол WebSockets, що

забезпечує їхню миттєву доставку. Повідомлення зберігаються в базі даних та синхронізуються між користувачами. Вихідними даними є оновлений чат із повідомленнями в реальному часі. Цей модуль підвищує ефективність взаємодії між користувачами, дозволяючи обмінюватись і отримувати повідомленнями як тільки вони були створені, без необхідності оновлювати застосунок.

У модулі для відео/аудіо зв'язку користувачі можуть проводити реальні часові комунікації з іншими учасниками системи. Цей модуль забезпечує безпосередню трансляцію відео- та аудіопотоків між учасниками, дозволяючи їм спілкуватися як наочно, так і голосово. Основними вхідними даними для цього модулю є відео- та аудіопотоки, які генеруються камерою та мікрофоном кожного учасника. Користувачі можуть активувати або деактивувати свою камеру та мікрофон за допомогою спеціальних кнопок у графічному інтерфейсі. Вихідними даними модулю є графічний інтерфейс, який відображає відео- та аудіопотоки учасників зв'язку.

У модулі для управління контактами користувачі можуть зберігати інформацію про свої контакти, з якими спілкуватимуться або планують звертатися в майбутньому. Вхідними даними є інформація про друзів, як-от їхня електронна пошта, якщо воно зареєстровані в системі і запрошення, надіслані іншими користувачами. Користувачі можуть ініціювати запит, щоб додавати друзів до списку через інтерфейс застосунку, та приймати або відхиляти запрошення через інтерфейс застосунку. Вихідними даними є графічний інтерфейс зі списком друзів і запрошень, які забезпечують швидкий доступ до важливої інформації та покращують організацію комунікації.

У модулі для перегляду історії спілкування користувачі можуть переглядати свої минулі бесіди та повідомлення. Вхідними даними є історія спілкування, збережена в базі даних. Користувачі можуть переглядати та шукати конкретні повідомлення через інтерфейс застосунку. Вихідними даними є графічний інтерфейс, що відображає історію спілкування, забезпечуючи зручний доступ до

важливої інформації та оптимізує процес аналізу бесід.

У модулі для зміни локалізації та мови користувачі можуть налаштовувати мову та регіональні параметри застосунку. Вхідними даними є вибір мови, який користувач робить через інтерфейс. Після введення даних вони зберігаються локально в налаштуваннях користувача. Вихідними даними є графічний інтерфейс, що відображає обрану мову та регіон, забезпечуючи комфортне користування застосунком. Цей модуль підвищує доступність платформи для ширшої аудиторії користувачів.

2.2 Розробка методу побудови вікон застосунку

Для реалізації графічного інтерфейсу користувача у вебзастосунку на базі React та Next.js було розроблено метод побудови вікон застосунку. React – популярна бібліотека для розробки інтерфейсів користувача, яка забезпечує високу ефективність та гнучкість. Next.js – фреймворк для React, який дозволяє створювати швидкі та SEO-оптимізовані вебдодатки з підтримкою серверного рендерингу та статичної генерації [10]. Метод охоплює кілька ключових етапів:

1. Вхідною точкою в застосунок є функція main(), яка ініціалізує додаток. У Next.js це реалізується через автоматичний запуск сервера та клієнтських компонентів. В цій ж функції також реалізовано інтерфейс Layout, щоб надати спільний макет для всього застосунку.

2. Для форматування локалізованих дат та текстового контенту використовується бібліотека next-intl. Локаль налаштовується на основі параметрів користувача, наприклад, «uk» для української мови.

3. Головний компонент App будує графічний інтерфейс застосунку. У Next.js це зазвичай реалізується через pages/_app.tsx, де можна налаштувати глобальні провайдери (наприклад, Redux Provider для глобального стану застосунку). Тут ініціалізуються загальні стилі та конфігурації, такі як теми та шрифти, а також метадані застосунку і глобальний менеджер помилок, щоб

користувач завжди був усвідомлений щодо статусу дій.

4. Для початкового завантаження даних використовуються методи getServerSideProps або getStaticProps, які забезпечують передачу даних з сервера на клієнт. Це дозволяє оптимізувати завантаження сторінок та покращити SEO-показники.

5. Першим екраном, який з’являється при запуску застосунку, є сторінка авторизації (AuthPage). Ця сторінка відображає форму для входу або реєстрації користувачів. Вона також містить посилання для відновлення паролю, що активує механізм скидання паролю через електронну пошту.

6. Для управління глобальним станом застосунку використовується Redux Toolkit. Через Provider з бібліотеки react-redux модель аутентифікації надається всім компонентам застосунку.

7. Тема застосунку налаштовується через Material UI, а також через глобальні стилі в styles/globals.scss. Це включає параметри для полів введення, кнопок, нижньої панелі навігації та інших елементів інтерфейсу. Користувачі можуть змінювати тему (світла/темна) через налаштування профілю.

8. Навігація між сторінками реалізується через маршрутизацію Next.js. Маршрути визначаються в теці pages, а для програмної навігації використовується хук useRouter з модуля next/router. Також реалізована обробка помилок 404 для неіснуючих маршрутів.

2.3 Розробка моделі системи

Для кращого розуміння архітектури розроблюваного застосунка побудовано модель системи у вигляді діаграми класів (див. рисунок 2.1) [11]. Вона відображає основні компоненти, їхні атрибути, методи та зв’язки між ними, що допомагає структуровано уявити взаємодію частин застосунку. На основі цієї діаграми можна оптимізувати код, спрощуючи його розширення та підтримку в майбутньому.

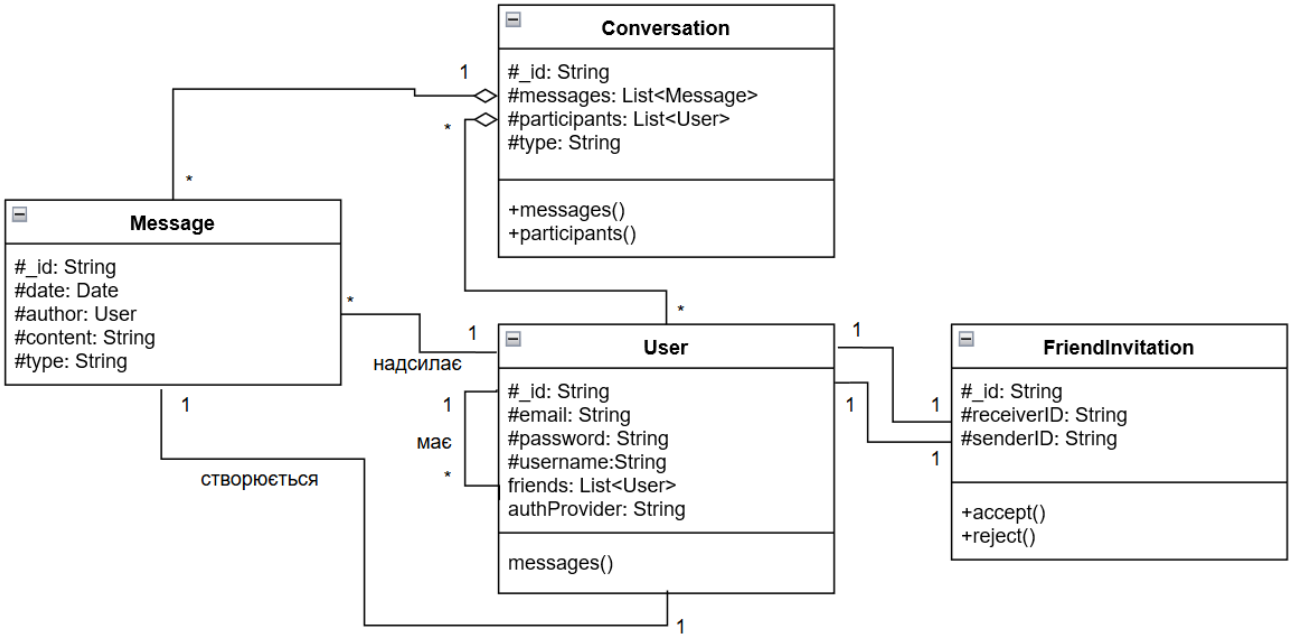

Рисунок 2.1 – Модель роботи системи з використанням діаграми класів

Основним класом у цій системі є ‘User’, який відповідає за зберігання даних користувачів. Поля цього класу включають такі атрибути, як ‘id’ (унікальний ідентифікатор), ‘email’ для електронної адреси, ‘password’ для пароля, який хешується і надійно шифрується на сервері під час реєстрації користувача, ‘username’ для нікнейму, ‘friends’ для списку контактів та ‘authProvider’, що вказує на провайдера автентифікації, і може набувати значення або ‘None’ або ‘Google’. Значення ‘None’ означає, що користувач зареєструвався використовуючи пошту і пароль, а якщо ‘Google’ – використовуючи Google OAuth.

Клас ‘Message’ має зв’язок багато до одного з класом ‘User’, що означає, що один користувач може мати декілька повідомлень. Також він пов’язаний з класом ‘Conversation’ через аналогічний тип зв’язку. Поля класу ‘Message’ включають ‘_id’ (унікальний ідентифікатор повідомлення), ‘date’ для дати створення, ‘author’ для збереження автора, ‘content’ для текстового вмісту та “type’, який визначає категорію повідомлення, які можуть бути ‘DIRECT’, для персонального (прямого), або ‘GROUP’ для групового повідомлення.

Клас ‘Conversation’ представляє собою об’єкт, де зберігаються дані про

бесіди між користувачами. Він має зв’язок один до багатьох з класом ‘User’ через поле ‘participants’. Це говорить про те, що одна бесіда може включати кількох учасників. Поля класу ‘Conversation’ містять такі атрибути, як ’_id’ (унікальний ідентифікатор бесіди), ‘messages’ для збереження списку повідомлень, ‘participants’ для переліку користувачів, що беруть участь, та ‘type’, який визначає тип бесіди, яка також може набувати значень ‘DIRECT’ і ‘GROUP’ з аналогічними призначеннями.

Клас ‘FriendInvitation’ призначений для управління запрошеннями до списку друзів. Він має зв’язок один до одного з класом ‘User’ через поля ‘receiverID’ та ‘senderID’. Поля класу ‘FriendInvitation’ включають ‘_id’ (унікальний ідентифікатор запрошення), ‘receiverID’ для отримувача та ‘senderID’ для відправника. Клас також надає методи ‘accept()’ для підтвердження запрошення та ‘reject()’ для його відхилення.

Ця діаграма класів демонструє складну взаємодію між різними компонентами системи, надаючи чітке уявлення про те, як користувачі спілкуються через повідомлення та бесіди. Вона також пояснює, як організовуються запрошення на дружбу, і показує механізми зберігання та обробки цих даних. Крім того, структура діаграми допомагає зрозуміти логіку взаємозв’язків між об’єктами, що забезпечує ефективне керування взаємодіями користувачів. Завдяки такому підходу, система залишається гнучкою для подальшого розширення функціоналу.

2.4 Розробка загального алгоритму роботи системи та діаграм станів

Основний алгоритм застосунку служить фундаментом для розробки програми. Він демонструє взаємодію між окремими модулями системи та визначає загальну логіку її функціонування (див рисунок 2.2).

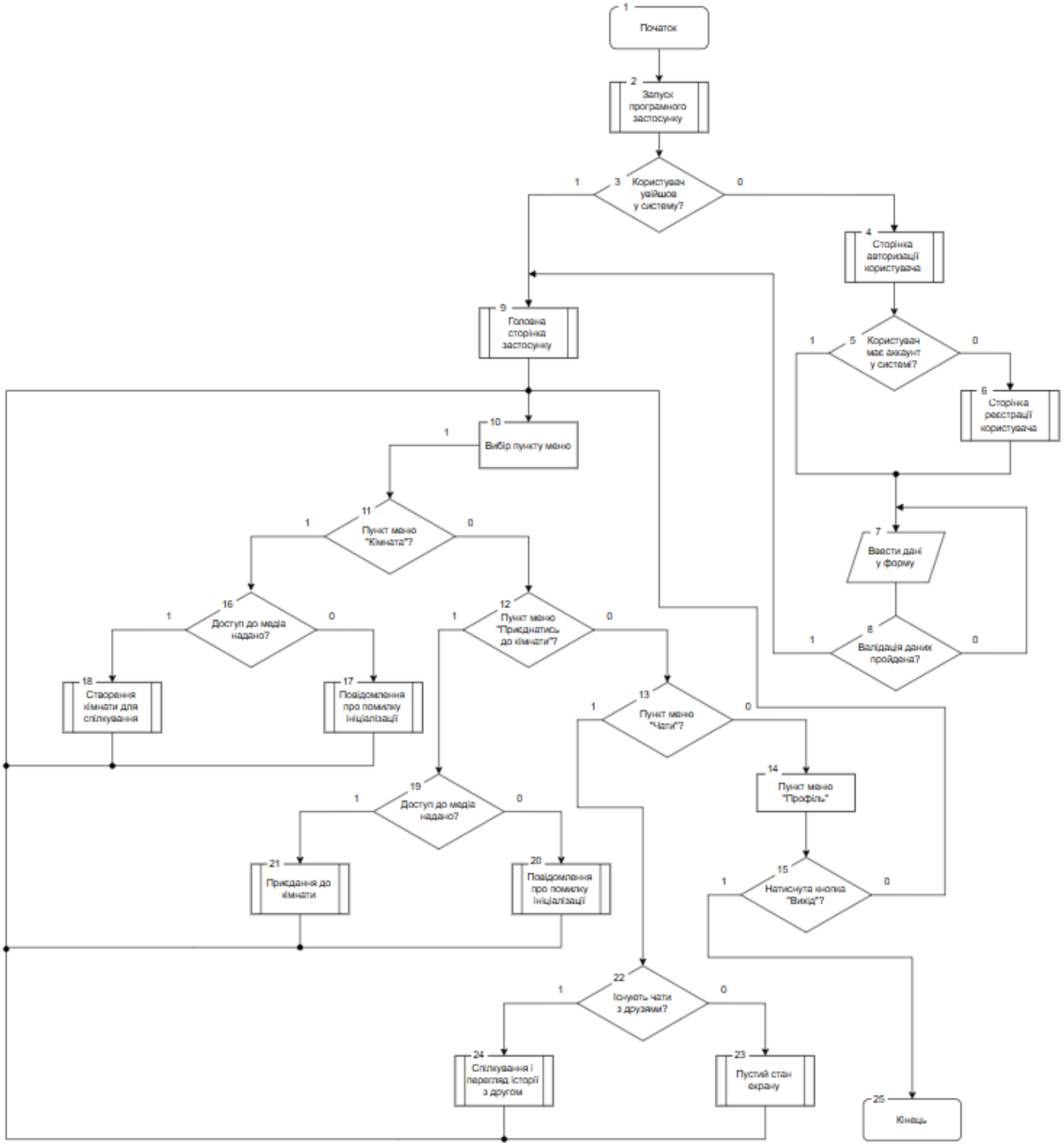

Рисунок 2.2 – Загальний алгоритм роботи системи

На початку користувач перевіряється на наявність входу в систему. Якщо користувач не авторизований, система пропонує перейти до сторінки авторизації або реєстрації. У випадку, коли користувач має обліковий запис, він вводить свої

дані у форму, яка проходить валідацію.

На головній сторінці користувачеві доступні різні пункти меню, серед яких можна обрати "Кімната", "Приєднатися до кімнати", "Профіль" чи "Чат". Залежно від вибору, відкривається відповідне функціональне вікно. Наприклад, при виборі пункту "Кімната" система перевіряє, чи має користувач доступ до медіа (відео/аудіо). Якщо доступ надано, створюється кімната для спілкування; інакше користувач отримує повідомлення про помилку.

Діаграми станів [12] є важливим інструментом моделювання, який описує поведінку системи шляхом визначення різних станів об’єкта та переходів між ними. Вони надають графічне представлення життєвого циклу об’єкта і демонструють реакцію на внутрішні або зовнішні події. Основними компонентами таких діаграм є: стани, події, дії та переходи.

Діаграма станів для модуля авторизації та реєстрації користувача представлена на рисунку 2.3.

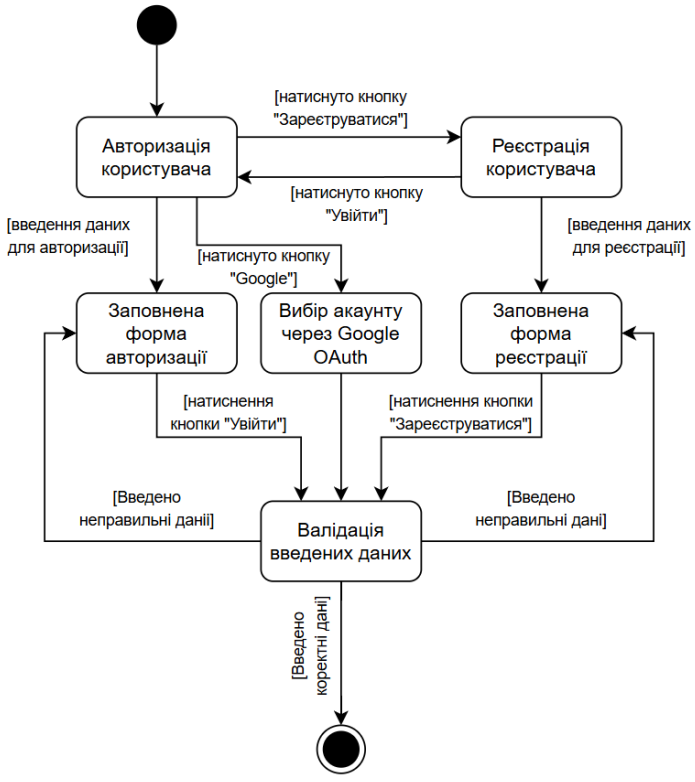

Рисунок 2.3 – Діаграма станів для модуля авторизації та реєстрації користувача

Діаграму станів модуля управління контактами наведено на рисунку 2.4. Вона ілюструє процес створення взаємозв’язку між користувачами, починаючи з пошти іншого користувача, і закінчуючи запитом з відповідною дією або обробкою помилок.

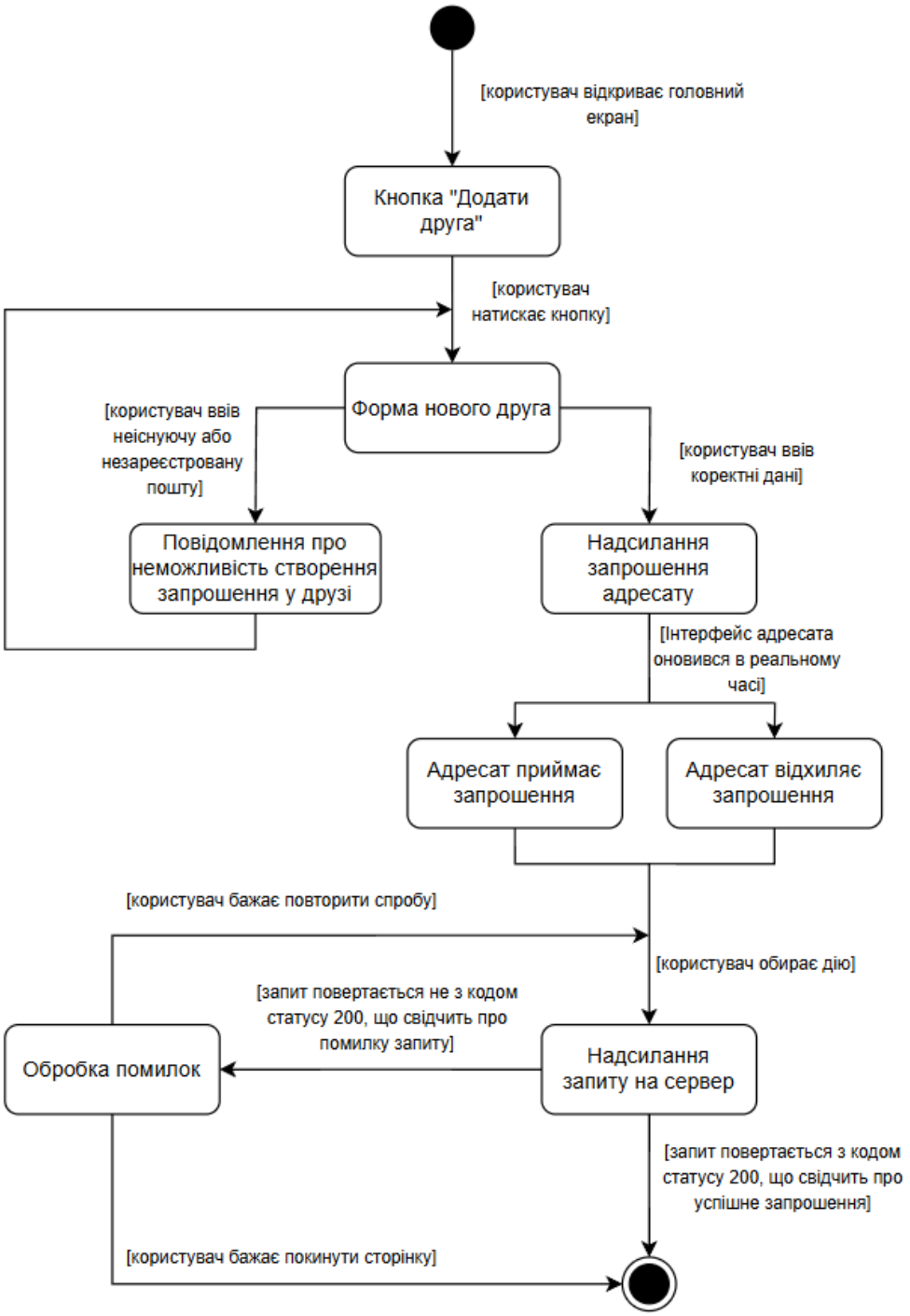

Рисунок 2.4 – Діаграма станів модуля управління контактами

Діаграму станів модуля інтернаціоналізації наведено на рисунку 2.5.

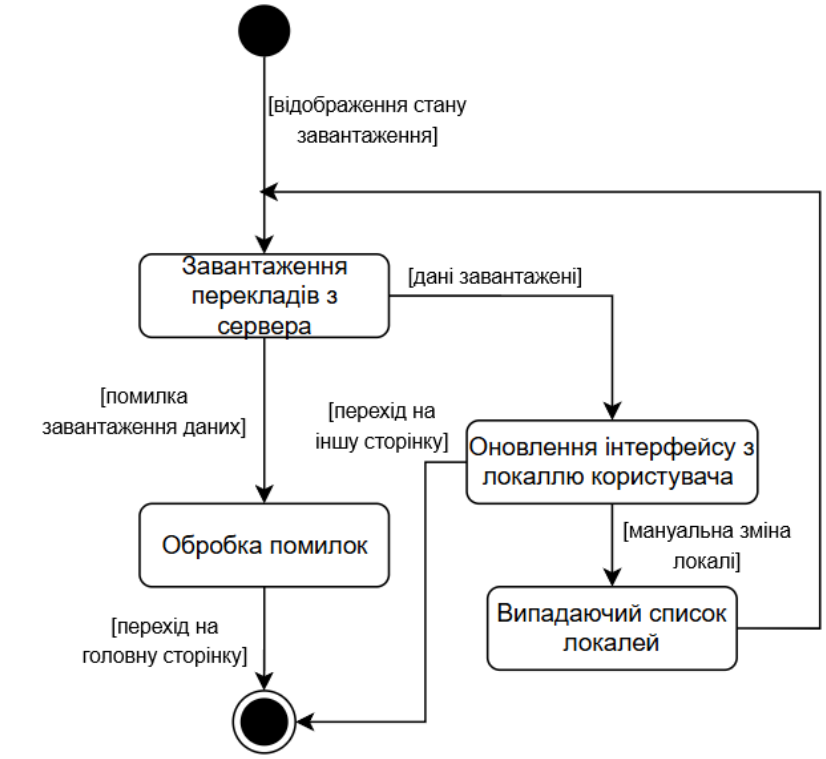

Рисунок 2.5 – Діаграма станів модуля інтернаціоналізації

Діаграму станів модуля профілю користувача наведено на рисунку 2.6.

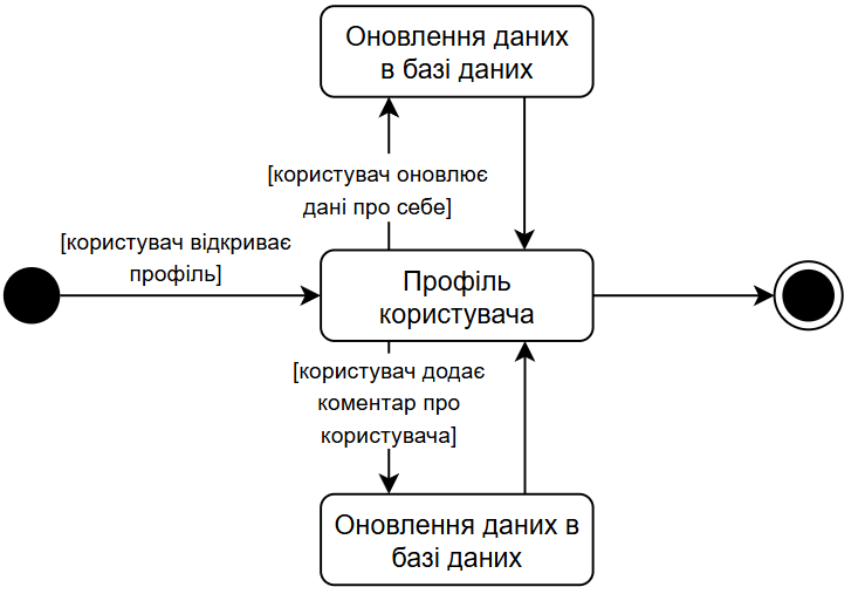

Рисунок 2.6 – Діаграма станів модуля профілю користувача

2.5 Висновки

У другому розділі детально розглянуто основні аспекти розробки програмного забезпечення для платформи онлайн-комунікацій, включаючи обробку та аналіз даних, зберігання інформації в базі даних, а також роботу з персональними даними користувачів для організації зв’язку. Особливу увагу приділено модулям авторизації, відеодзвінків, інтернаціоналізації, управління чатами, управління контактами/друзями.

Метод побудови вікон застосунку на базі React та Next.js продемонстровано через створення інтерфейсу користувача за допомогою декларативного підходу та широких можливостей фреймворку Next.js. Це дозволяє розробляти ефективний та оптимізований код, полегшуючи підтримку та масштабування застосунку. Для створення динамічних списків використовуються компоненти, використовуючи вбудований метод масивів map, а маршрутизація між сторінками реалізується через систему файлів у теці pages та хук useRouter з модуля next/router.

Загальний алгоритм роботи системи та діаграми станів чітко показують взаємозв’язки між модулями програми та їх поведінку. Діаграми станів для авторизації, реєстрації, створення чатів, перегляду профілів користувачів та дашборду користувача ілюструють, як об’єкти реагують на події, що полегшує розробку складних застосунків. Кожен модуль має чітко визначені стани та переходи, що забезпечує надійність та прозорість системи.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

JavaScript – це потужна і гнучка мова програмування, яка стала основним інструментом для розробки сучасних вебдодатків. З появою фреймворків, таких як React і Next.js, JavaScript отримав додаткові можливості для створення ефективних і швидких вебзастосунків. TypeScript, строго типізована версія JavaScript, ще більше розширює можливості, додаючи підтримку типів і покращуючи масштабованість коду. TS поєднує в собі простоту та розширені можливості, що робить його найкращим вибором для створення вебзастосунків [13].

C# – це об’єктно-орієнтована мова програмування, розроблена Microsoft. Вона широко використовується для створення вебдодатків за допомогою ASP.NET Core [14]. Хоча C# забезпечує високу продуктивність і добре інтегрується з іншими технологіями Microsoft, вона менш популярна у фронтенд-розробці через свою орієнтацію на серверну частину.

Java – це надійна і класична мова програмування, яка використовується для розробки широкого спектру додатків, включаючи вебсистеми, вебдодатки за допомогою фреймворків, таких як Spring Boot. Java відома своєю платформонезалежністю ("Write Once, Run Anywhere"), але її синтаксис, строга типізація, концепції і конфігурації VM можуть бути надто громіздкими для швидкої розробки вебінтерфейсів [15]. У порівнянні з JS/TS, Java вимагає більше часу та ресурсів для досягнення подібних результатів, особливо у контексті сучасних вебтехнологій.

PHP – це популярна мова програмування для веброзробки, яка використовується для створення серверних додатків [16]. PHP є досить простим для початківців, але через свою старішу архітектуру та меншу оптимізацію для сучасних вебдодатків, вона поступається JS/TS у питаннях продуктивності,

масштабованості та підтримки сучасних фреймворків. JS/TS пропонує більш сучасний підхід до веброзробки, що робить його кращим вибором для розробки сучасних вебзастосунків.

Порівняння функціональних характеристик мов програмування JavaScript/TypeScript, C#, Java та PHP наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння функціональних характеристик мов програмування

Характеристика	TS	C#	Java	PHP
Об’єктно-орієнтованість	1	1	1	1
null-безпека	1	0	1	1
Підтримка асинхронності	1	1	1	1
Платформонезалежність	1	0	1	0
Строга типізація	1	1	1	0
Динамічна індексація	1	1	0	1
Загальний коефіцієнт	6	4	5	4

TypeScript, маючи вищий сумарний коефіцієнт, опереджає своїх конкурентів у розробці сучасних вебдодатків. Саме тому TypeScript був обраний для розробки платформи онлайн-комунікацій, оскільки він забезпечує оптимальне поєднання продуктивності, гнучкості та простоти розробки.

Компанії-розробники програмного забезпечення пропонують широкий вибір сучасних середовищ розробки вебзастосунків. Розглянемо найпопулярніші з них: WebStorm, Eclipse, Visual Studio Code та Visual Studio.

WebStorm – це потужне інтегроване середовище розробки (IDE), розроблене компанією JetBrains, яке спеціалізується на веброзробці та підтримці сучасних мов програмування, таких як JavaScript, TypeScript, HTML, CSS та інших [17]. WebStorm є улюбленим інструментом для багатьох розробників завдяки своїй

високій продуктивності, автоматизації процесів та інтелектуальному аналізу коду. Однією з головних переваг WebStorm є його вбудована підтримка фреймворків, таких як React, Angular, Vue.js, а також можливість легко інтегруватися з інструментами, такими як Node.js, npm, Git, Docker та хмарними платформами. WebStorm також надає розширюваність через плагіни, що дозволяє адаптувати IDE під конкретні потреби проекту.

Eclipse – це безкоштовне інтегроване середовище розробки (IDE) з відкритим вихідним кодом, яке підтримує багато мов програмування, включаючи Java, JavaScript, Python, PHP та інші [18]. Eclipse має потужний набір інструментів для розробки програмного забезпечення, включаючи підтримку налагодження, рефакторингу, контролю версій тощо.

Eclipse можна використовувати для веброзробки за допомогою плагінів, таких як Eclipse Wild Web Developer. Однак, порівняно з іншими IDE, Eclipse може бути менш зручним у використанні та налаштуванні для веброзробки.

Visual Studio Code (VS Code) – це безкоштовне, легке та потужне інтегроване середовище розробки (IDE), розроблене компанією Microsoft [19]. VS Code підтримує велику кількість мов програмування та платформ, що робить його універсальним інструментом для розробки вебзастосунків. Однією з головних переваг VS Code є його розширюваність за допомогою численних плагінів, що дозволяє налаштувати IDE під конкретні потреби розробника.

VS Code забезпечує інтеграцію з різними інструментами та сервісами, такими як Git, Docker, хмарні платформи, а також має підтримку редагування коду у браузері через VS Code for Web.

Visual Studio – це потужне інтегроване середовище розробки (IDE), розроблене компанією Microsoft [20]. Воно підтримує велику кількість мов програмування, включаючи .NET, C++, Python, JavaScript та інші.

Visual Studio часто використовується для бекенд-розробки та створення складних корпоративних вебзастосунків. Воно має інструменти для тестування,

розгортання та інтеграції з Azure, що робить його потужним рішенням для великих команд розробників. Однак, у порівнянні з WebStorm чи VS Code, Visual Studio може бути більш ресурсозатратним та менш гнучким у налаштуванні.

Порівняння функціональних характеристик середовищ наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння функціональних характеристик середовищ програмування

Характеристика	VS Code	Eclipse	WebStorm	Visual Studio
Підтримка веброзробки	1	1	1	1
Вбудовані інструменти для JavaScript і TypeScript	1	0	1	1
Інтеграція з системами контролю версій	1	1	1	1
Можливість підключення сторонніх плагінів	1	1	1	0
Підтримка бекенд-розробки	1	1	1	1
Вбудовані інструменти для тестування	0	1	1	1
Загальний коефіцієнт	5	5	6	5

За таблицею, WebStorm є найкращими середовищем для фронтенд-розробки, оскільки має гнучкість, розширюваність та глибоку інтеграцію з JavaScript-екосистемою. Visual Studio і VS Code краще підходять для великих проєктів та бекенд-розробки, а Eclipse залишається гарним вибором для Java та

корпоративних рішень.

3.2 Розробка модуля для авторизації та реєстрації користувача

Призначенням модуля автентифікації є створення зручного та безпечного механізму входу й реєстрації користувачів у вебзастосунку. При відкритті сторінки автентифікації користувач отримує можливість обрати між входом та реєстрацією (див. рисунок 3.1).

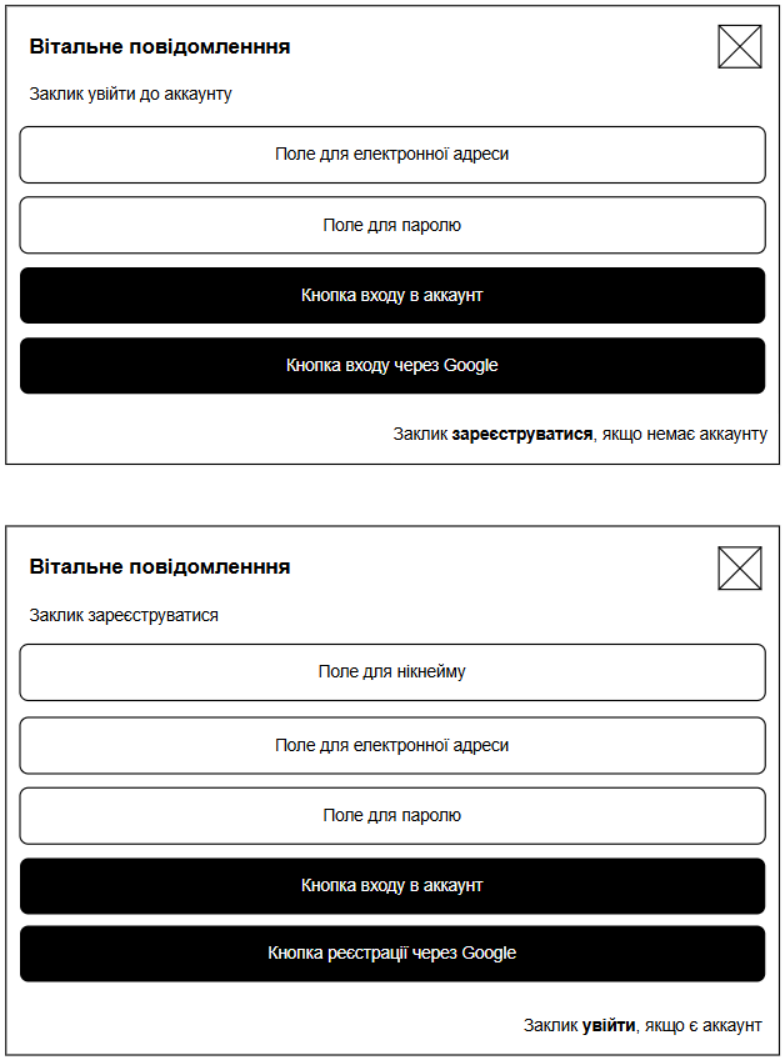

Рисунок 3.1 – Схема інтерфейсу сторінок авторизації та реєстрації

Для створення облікового запису або входу в існуючий, йому потрібно вибрати відповідну опцію, після чого система відображає відповідну форму , що реалізовано за допомогою динамічних інтерфейсів на основі React та Next.js.

Процес автентифікації реалізовано за допомогою JWT-токенів, які зберігаються у cookies для забезпечення безперервного сеансу користувача. При наявності токена система автоматично виконує запит на отримання інформації про користувача через Redux Thunk, що дозволяє забезпечити плавний перехід між сторінками без необхідності повторного введення облікових даних при кожному перезавантаженні застосунку.

Фронтенд застосунку використовує Material UI (MUI) для стилізації компонентів та створення адаптивного дизайну, що відповідає сучасним стандартам UX/UI. Форма автентифікації реалізована за допомогою react-hook-form та Yup для валідації даних на клієнтському боці. Також передбачена можливість входу через OAuth 2.0, зокрема через Google, використовуючи спеціально згенеровані URL-адреси для авторизації.

Бекенд працює на Node.js з використанням Express.js для обробки запитів. Реалізовано захищену авторизацію за допомогою хешування паролів через bcrypt та генерацію токенів через jsonwebtoken. Обмін даними з базою відбувається через MongoDB, з використанням Mongoose для управління моделями даних.

Основні компоненти модуля:

- LoginPage – сторінка входу, що містить форму для введення електронної пошти та пароля, а також кнопку для входу через Google.
- RegisterPage – сторінка реєстрації, що містить поля для введення імені користувача, електронної пошти та пароля.
- authSlice – Redux-редюсер, що керує станом авторизації та зберігає інформацію про користувача.
- getCurrentUser – асинхронна функція, що отримує поточний стан користувача при наявності токена.

- login та register – Thunk-дії для відправки запитів на сервер з метою входу або реєстрації нового користувача.

Ця архітектура забезпечує гнучкість, безпеку та високу продуктивність автентифікаційного модуля, що критично важливо для сучасних вебзастосунків із підтримкою real-time обміну даними через WebSockets.

Було побудовано блок-схему для взаємодії з головною сторінкою застосунку, зображену на рисунку 3.2. Вона описує взаємодію користувача з основними компонентами модуля автентифікації і головною сторінкою.

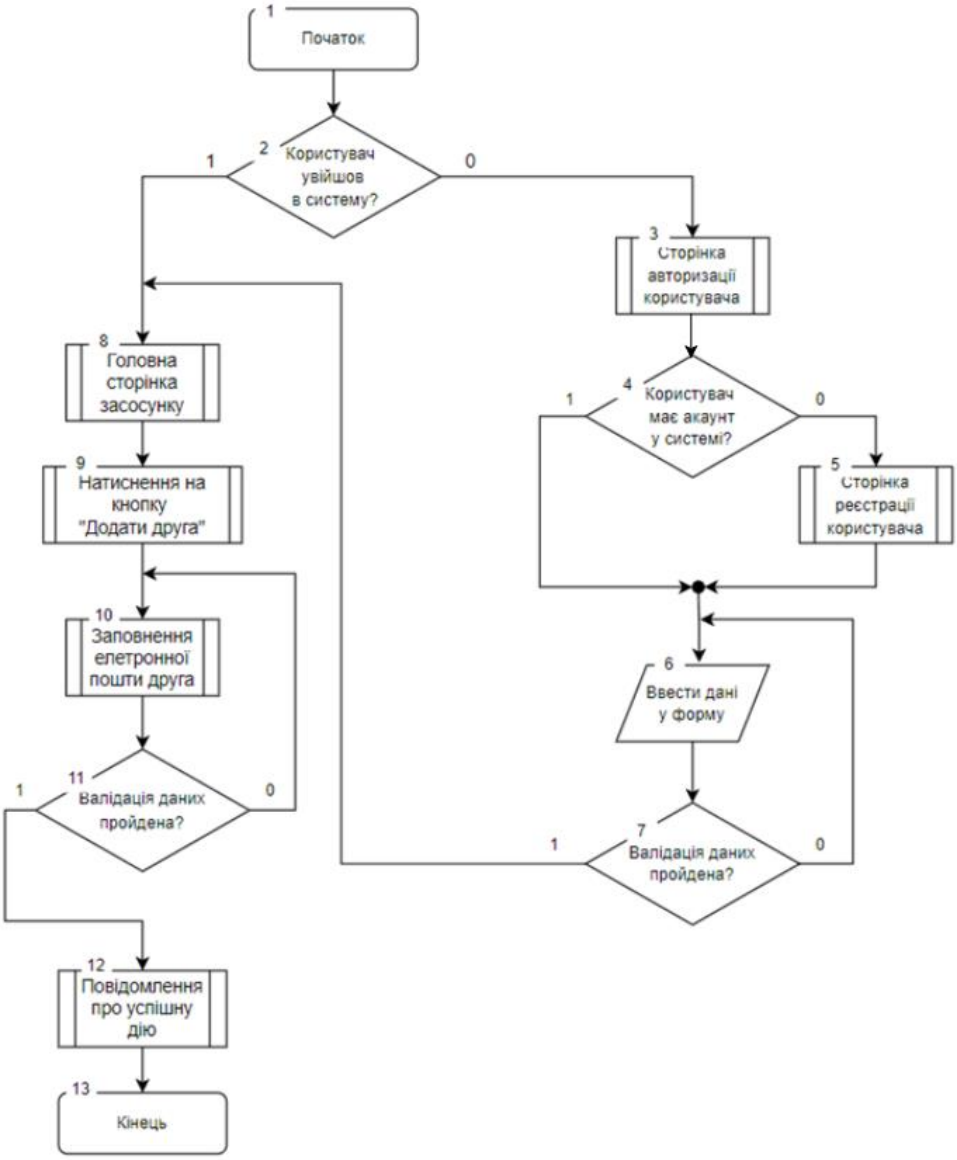

Рисунок 3.2 – Блок-схема взаємодії з головною сторінкою застосунку

3.3 Розробка модуля перегляду списку контактів

Цей модуль відповідає за створення інтерфейсу для відображення списку контактів у застосунку. Він реалізований з використанням компонентів та бібліотеки Material-UI для забезпечення швидкої взаємодії користувачів з інтерфейсом. Для оптимізації продуктивності та масштабованості використовується серверний рендеринг, що дозволяє ефективно завантажувати дані про контакти.

Кожен елемент списку представляє картку контакту з інформацією про нього, а також вказує, чи є цей контакт онлайн. Кожна картка контакту включає основну інформацію (наприклад, ім’я). Для реалізації списку контактів використовується компонент List з бібліотеки Material-UI, який генерується за допомогою методу map. Кожен елемент списку є окремим компонентом ContactCard, який відповідає за відображення інформації про контакт та взаємодію з ним. Графічна схема модуля наведена на рисунку 3.3.

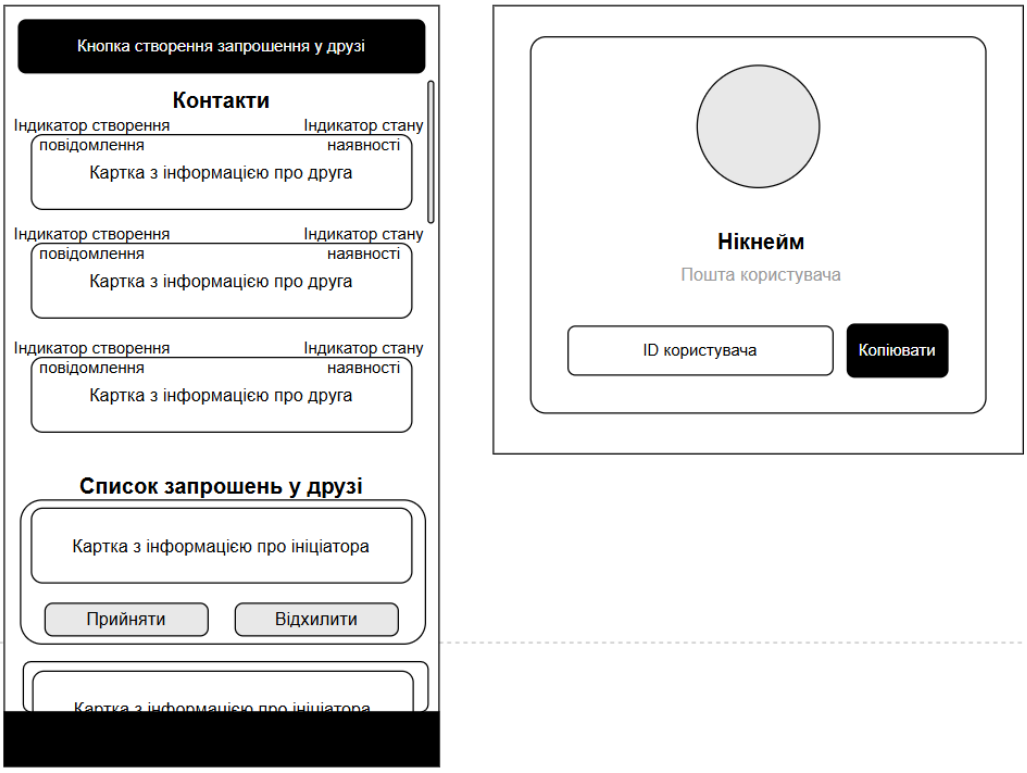

Рисунок 3.3 – Графічна схема модуля перегляду списку контактів

3.4 Розробка модуля для здійснення відеодзвінків

Призначення модуля полягає в наданні зручного та функціонального інтерфейсу для проведення відеодзвінків з друзями, що забезпечує комфортний досвід для користувачів. Алгоритм описаний у вигляді блок-схеми на рисунку 3.4 та відповідає за управління відеодзвінком, включаючи початок і завершення дзвінка, перемикання стану мікрофону та камери, а також можливість поширення екрану.

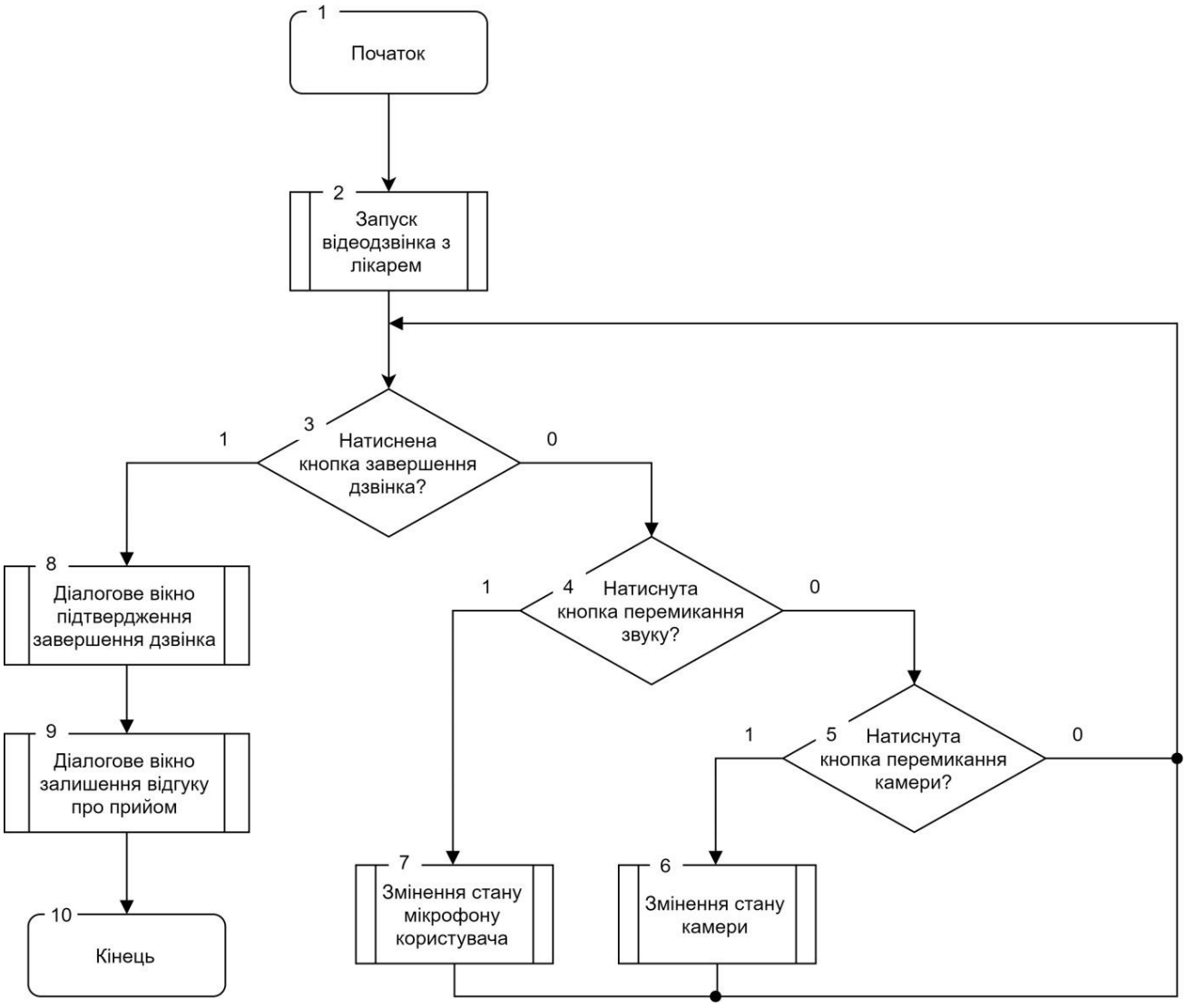

Рисунок 3.4 – Блок-схема алгоритму роботи модуля для здійснення відеодзвінків

Алгоритм починається з ініціалізації процесу відеодзвінка для онлайн-комунікації. На наступному етапі відбувається запуск відеодзвінка між користувачами, під час якого ініціалізуються камера та мікрофон для забезпечення якісного зв’язку.

Система постійно моніторить, чи була активована кнопка завершення дзвінка. У випадку її натискання на екрані з’являється діалогове вікно для підтвердження бажання завершити дзвінок. Якщо ж кнопка завершення не була натиснута, система проводить перевірку наявності взаємодії з кнопкою вимкнення/увімкнення звуку. При її активації стан мікрофону користувача змінюється.

Коли кнопка управління звуком також не була задіяна, система переходить до аналізу натискання кнопки перемикання камери. Якщо ця кнопка активована, здійснюється зміна статусу роботи камери.

У ситуації, коли жодна з кнопок не була натиснута, система повертається до початкового етапу – перевірки стану кнопки завершення дзвінка.

У разі підтвердження наміру завершити дзвінок через діалогове вікно, процес відеодзвінка офіційно завершується, і сеанс припиняється.

Важливим елементом інтерфейсу відеодзвінка є відображення кнопок управління і потоком контенту. Це дозволяє користувачу легко управляти інформацією, яка йому і іншим буде доступна під час дзвінку (див. рисунок 3.5).

```
<div className='fixed bottom-[10px] right-[10px] flex flex-col items-center
     style={isRoomMinimized ? minimizedRoomStyle : fullScreenRoomStyle}>
  <VideosContainer />
   <RoomButtons>
      <ResizeRoomButton
          isRoomMinimized={isRoomMinimized}
          onClick={handleRoomResize}
      />
   </RoomButtons>
</div>
```

Рисунок 3.5 – Графічний інтерфейс кімнати

Відображення кімнати здійснюється за допомогою компонента Room, який включає кнопки керування і безпосередньо сам потік.

3.5 Розробка моделі системи

При створенні програмного забезпечення для платформи онлайн-комунікацій важливо детально спланувати її архітектуру та логіку взаємодії користувачів. З метою візуалізації процесів була розроблена UML-діаграма діяльності, яка ілюструє сценарії використання системи (див. рисунок 3.6).

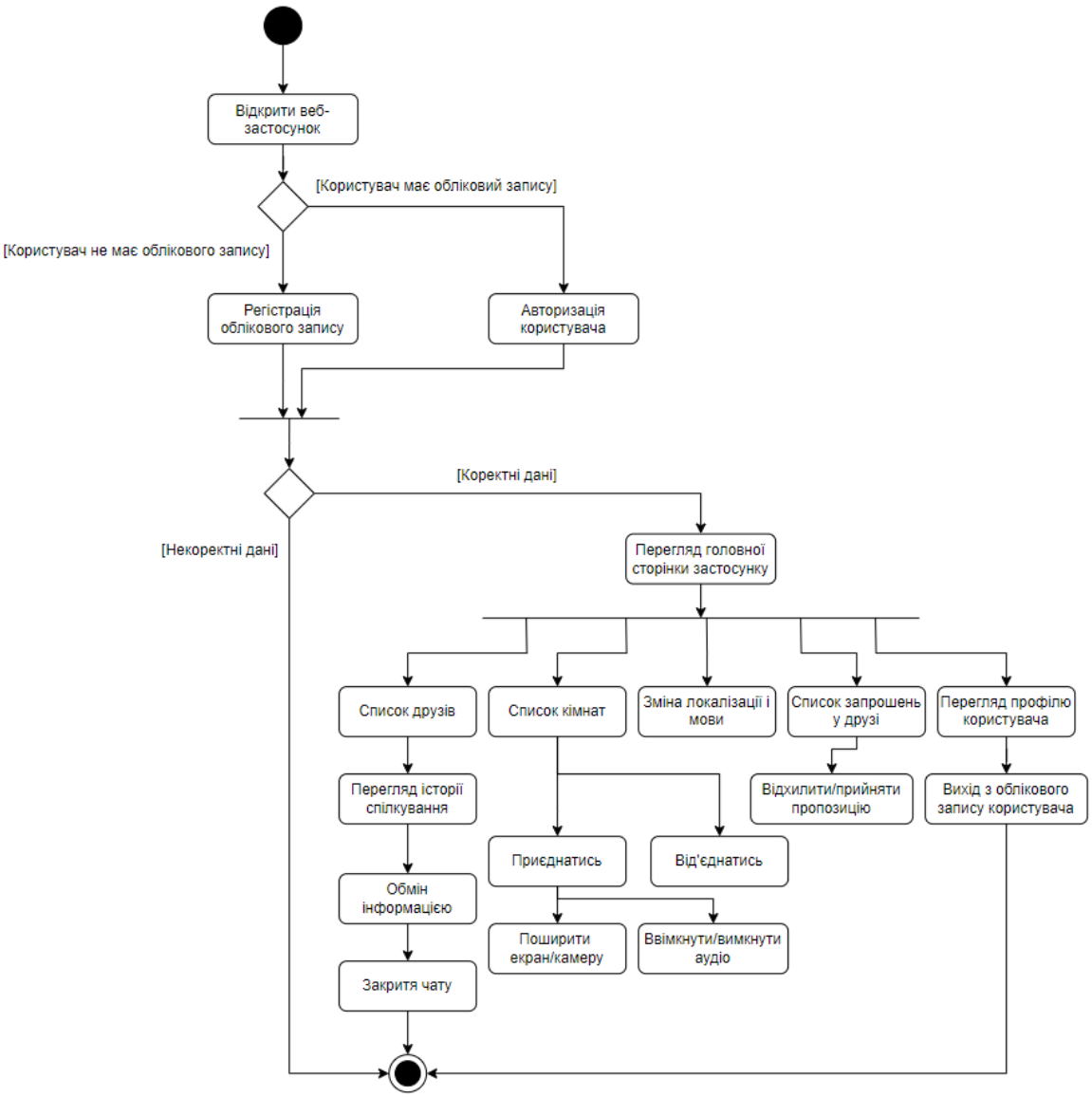

Рисунок 3.6 – Діаграма діяльності системи

На діаграмі діяльності представлено процес взаємодії користувача з вебзастосунком для онлайн-комунікацій. Процес починається з вибору активності: користувач може зареєструватися чи авторизуватися. Після входу в систему користувачеві доступні декілька опцій: перегляд списку друзів, кімнат і запрошень у друзі, а також можливість зміни локалізації і переглянути власний профіль.

3.6 Розробка структури інтерфейсу програмного застосунку

Оскільки застосунок розробляється для максимально широкої аудиторії і екранів, які можуть мати обмежену робочу область, важливо створити інтерфейс, який може адаптуватись під конкретний екран користувачів зі збереженням функціоналу. При цьому інтерфейс має забезпечувати комфортне користування програмою.

Для забезпечення зручної навігації у платформі реалізовані верхній навігаційний бар (хедер). Він забезпечує швидкий доступ до основних розділів застосунку та покращує користувацький досвід, при цьому займаючи мінімум робочого простору.

Оскільки платформа орієнтована на широку аудиторію, важливим аспектом розробки є забезпечення інтернаціоналізації. Користувачі з різних країн повинні мати можливість використовувати застосунок своєю рідною мовою, що підвищує зручність і доступність сервісу.

Для цього реалізовано функцію зміни мови без необхідності перезавантаження сторінки. У верхньому навігаційному барі передбачений спеціальний перемикач мов, який дозволяє швидко обирати потрібну локалізацію. Це не тільки покращує користувацький досвід, а й сприяє розширенню аудиторії платформи, роблячи її більш гнучкою для міжнародного використання.

Завдяки цьому підходу користувачі можуть комфортно взаємодіяти з платформою незалежно від їхнього мовного середовища, що є важливим фактором для сучасних онлайн-комунікаційних сервісів (див. рисунок 3.7). Крім того,

можливість легко адаптувати інтерфейс під різні мови робить сервіс привабливим для людей з різних культурних та географічних регіонів, що підкреслює його глобальну спрямованість.

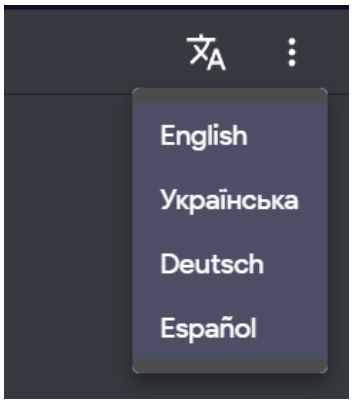

Рисунок 3.7 – Перемикач мов в хедері

Приклад перекладу головної сторінки на українську мову наведено на рисунку 3.8.

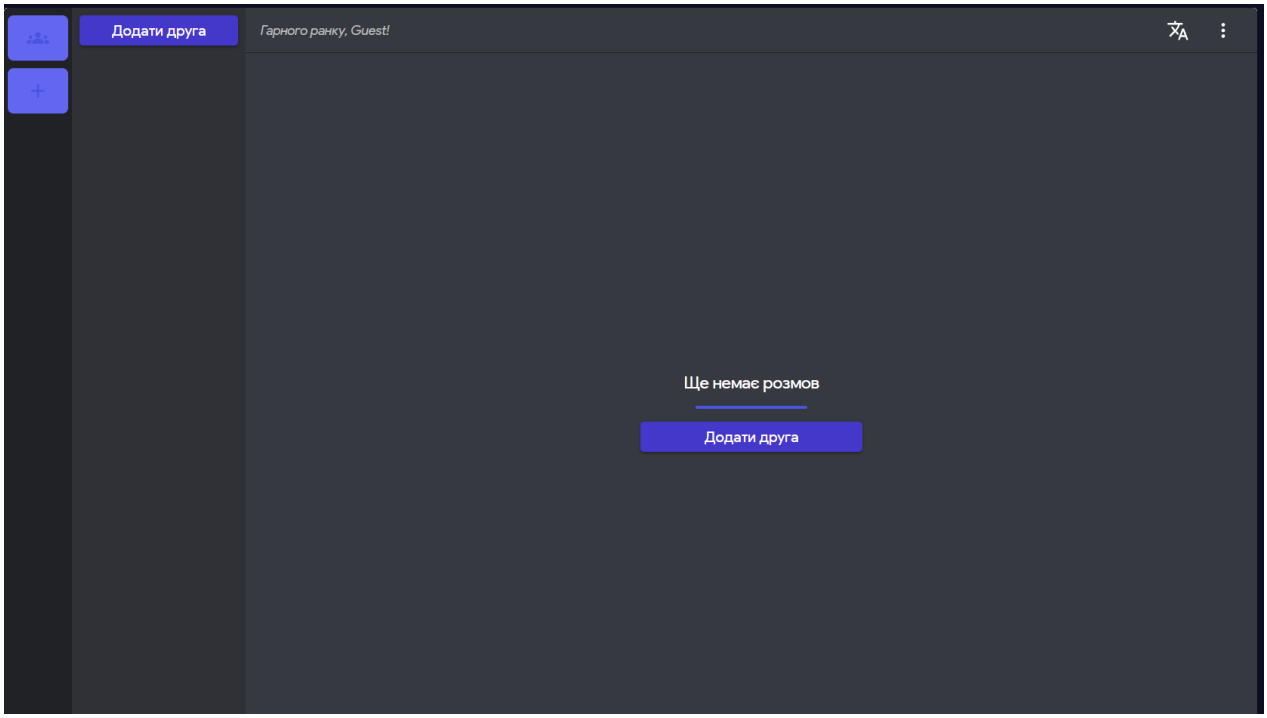

Рисунок 3.8 – Перекладена на українську мову головна сторінка

Перша сторінка, з яким зустрічається користувач – вікно авторизації. Користувачеві пропонується ввести електронну пошту та пароль для входу до облікового запису (див. рисунок 3.9). Для забезпечення надійного та захищеного процесу перевірки даних використовується сучасний підхід, що включає генерацію спеціальних ключів для підтвердження автентичності користувача. Це дозволяє платформі гарантувати безпечну взаємодію між користувачем та системою завдяки Node.js (Express) на бекенді, з використанням JWT (JSON Web Token).

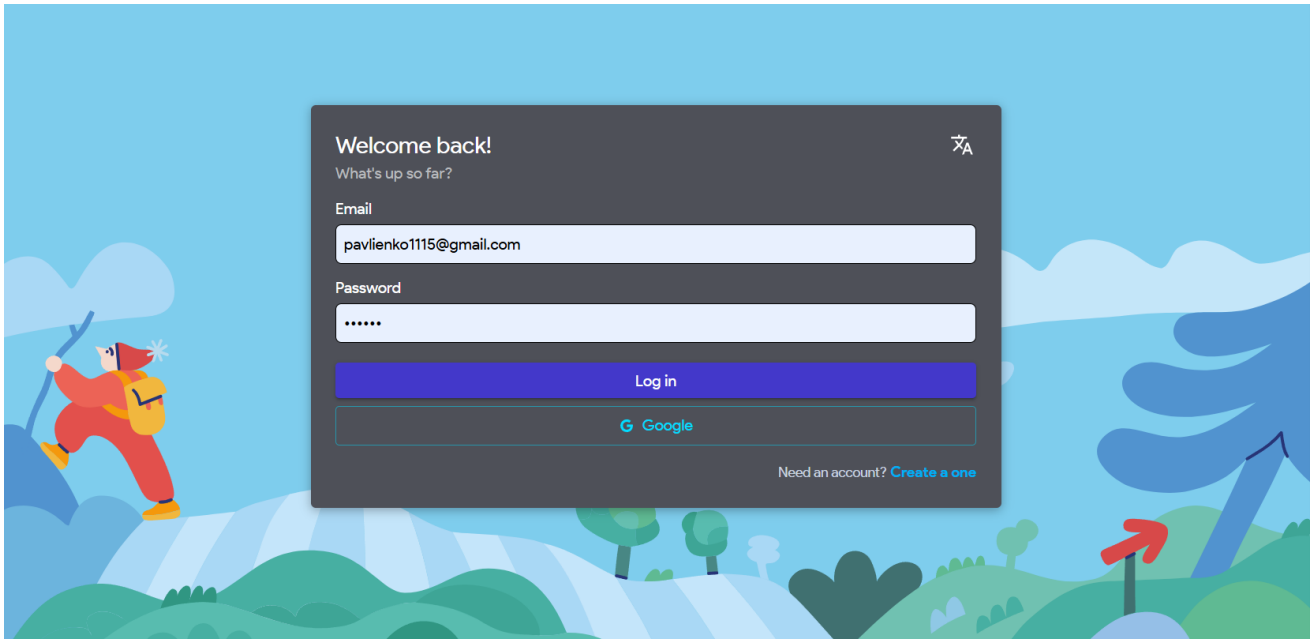

Рисунок 3.9 – Вікно авторизації

У випадку, коли користувач ще не має облікового запису, він може створити його, скориставшись кнопкою «Зареєструватися» (див. рисунок 3.10). На екрані реєстрації представлені поля для внесення імені, адреси електронної пошти та пароля, причому процес супроводжується перевіркою коректності введених даних як на рівні інтерфейсу, так і на рівні серверної частини системи. Такий підхід забезпечує надійність та безпеку при створенні нового облікового запису.

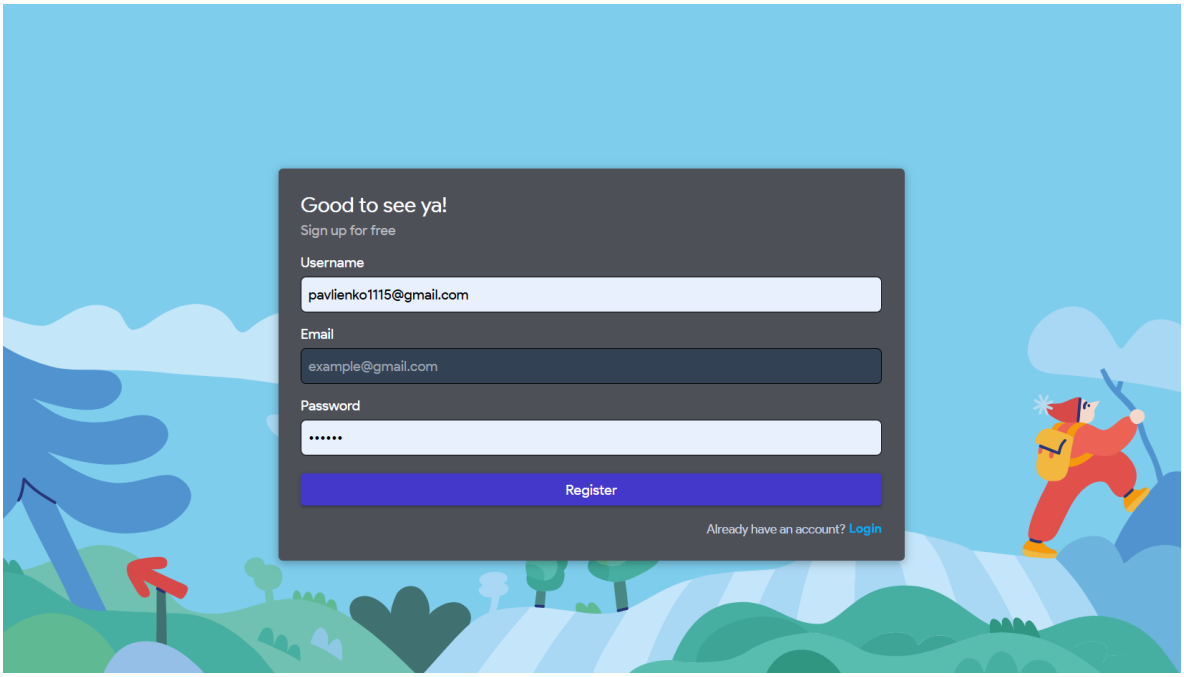

Рисунок 3.10 – Вікно реєстрації

У разі введення некоректних даних під час реєстрації або входу в систему, користувач отримує детальне повідомлення про помилку. Валідація введених даних реалізована як на фронтенді (через React Hook Form і Yup), так і на бекенді (з використанням Express Validator).

Крім того, подвійна валідація на рівні клієнта та сервера забезпечує додатковий захист від некоректних або шкідливих даних, що підвищує загальну безпеку системи. У разі виявлення помилки, користувачеві надаються чіткі інструкції щодо її виправлення, що робить процес взаємодії з платформою максимально зрозумілим і зручним.

Також при спробі входу з некоректними даними система надає відповідне повідомлення про помилку авторизації. Важливим елементом є обробка статусів помилок (наприклад, 401 Unauthorized або 403 Forbidden) на клієнтському рівні. Помилки оброблюються і повідомляються на глобальному рівні, щоб користувач завжди розумів стан системи.

Після успішної авторизації користувач потрапляє на головну сторінку

платформи, де доступні можливості текстового та відеоспілкування з іншими користувачами. Для забезпечення реального часу використовується WebSockets (socket.io) для миттєвого обміну повідомленнями та WebRTC для відеозв'язку безпосередньо між користувачами.

Функціонал запрошень є важливою складовою платформи, оскільки він дозволяє користувачам легко знаходити та додавати друзів для подальшої комунікацій. Процес запрошень складається з кількох ключових етапів, які забезпечують зручний процес надсилання, перегляду та керування запрошеннями. Користувач може ввести електронну адресу друга для відправлення запрошення (див. рисунок 3.11). Валідація введених даних гарантує, що електронна пошта вказана коректно. При успішному надсиланні користувач отримує підтвердження, а запрошений друг бачить запит у своєму списку.

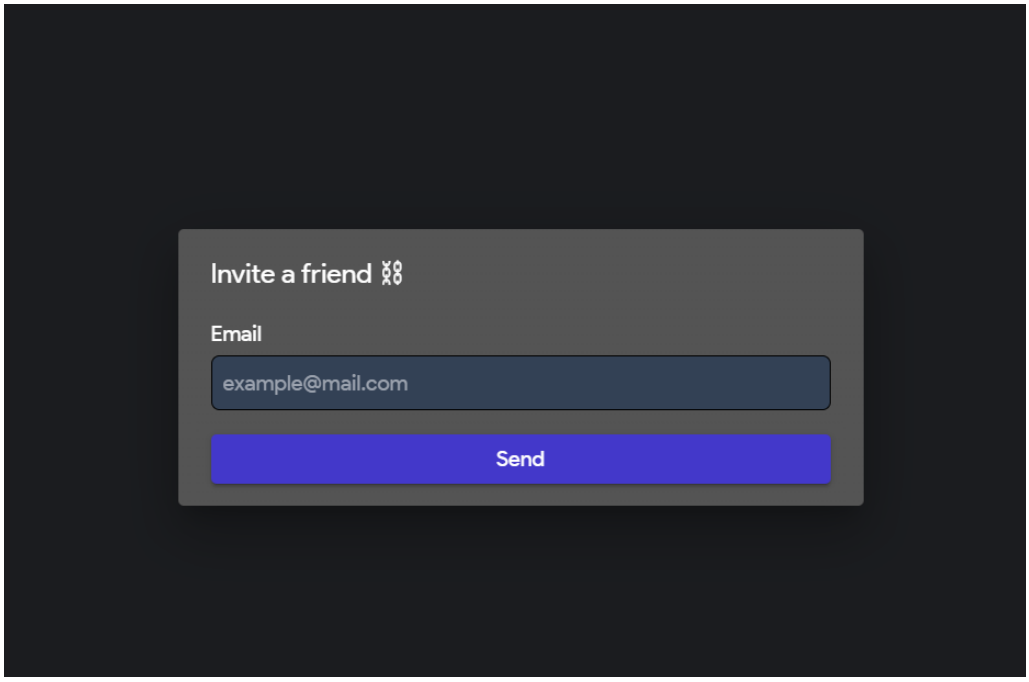

Рисунок 3.11 – Модальне вікно для запрошення друга

Всі отримані запрошення відображаються у вигляді списку. Якщо запрошень немає, цей блок ховається для збереження простору інтерфейсу. Кожне

запрошення містить ім’я відправника та його аватар, що покращує візуальну ідентифікацію.

Користувач може прийняти або відхилити запрошення за допомогою відповідних кнопок. Натискання на кнопку миттєво оновлює список та виконує відповідний запит на сервер. Кнопки стають неактивними після виконання дії, щоб уникнути повторних запитів. Після прийняття запрошення, користувачі стають друзями і можуть безпосередньо спілкуватись. Якщо запрошення було відхилено, користувач зможе знову надіслати запрошення. Інтерфейс додатково має анімації для плавного оновлення списку запрошень, що робить взаємодію більш природною та комфортною.

Функціонал реалізовано інтуїтивно та мінімалістично, що робить процес додавання друзів швидким і зручним (див. рисунок 3.12).

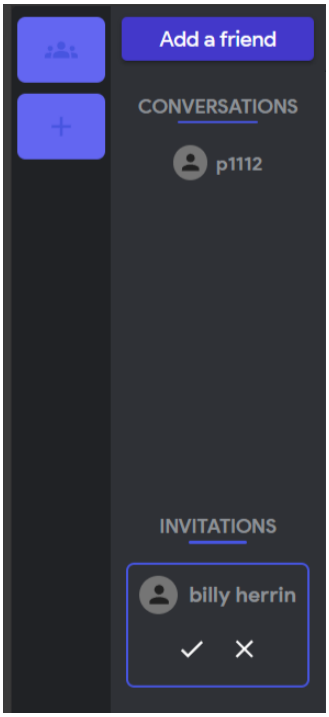

Рисунок 3.12 – Список запрошень в друзі

Екран чату є ключовим елементом платформи для онлайн-комунікацій, оскільки саме тут відбувається безпосередня взаємодія між користувачами. Його

дизайн розроблений таким чином, щоб забезпечити максимальну зручність і ефективність спілкування.

Повідомлення розміщуються в хронологічному порядку та автоматично прокручуються до останнього отриманого повідомлення. Для кращої візуальної орієнтації в історії чату додано роздільники за датами, які відображають початок нового дня спілкування.

Повідомлення від поточних користувачів та співрозмовників мають різний стиль оформлення, що дозволяє швидко орієнтуватися в розмові. Крім того, якщо повідомлення одного автора йдуть підряд, система групує їх для візуальної зручності.

Біля кожного повідомлення розміщується аватар автора та позначка часу, що відображає момент відправлення повідомлення. Це забезпечує кращий контекст у розмові та допомагає відстежувати її хронологію (див. рисунок 3.13).

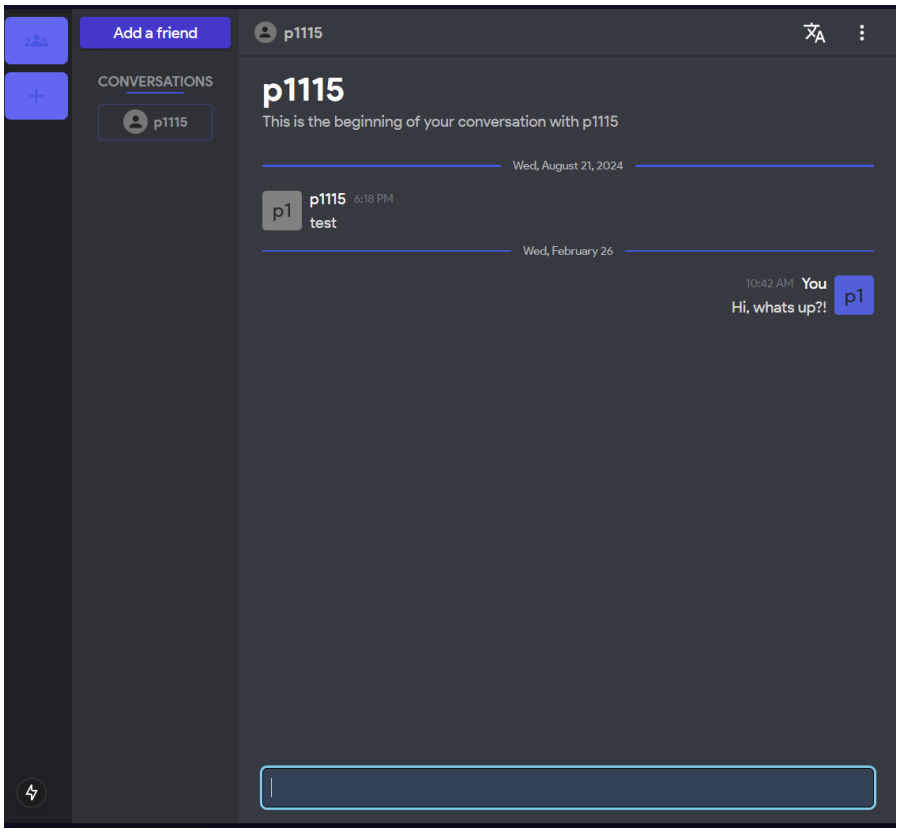

Рисунок 3.13 – Екран чату

Текстове поле для введення повідомлень є інтерактивним – під час введення тексту система надсилає сигнал про набір повідомлення, що допомагає уникнути ситуацій, коли користувачі пишуть одночасно або очікують відповіді, не знаючи, чи співрозмовник взагалі відповість. Якщо співрозмовник набирає повідомлення, система в реальному часі повідомляє про це, додаючи під текстовим полем відповідний індикатор. Це підвищує відчуття присутності та робить комунікацію більш природною. Також варто зазначати, що друзі, як зараз онлайн, мають зелену позначку біля їхніх аватарів (див. рисунок 3.14).

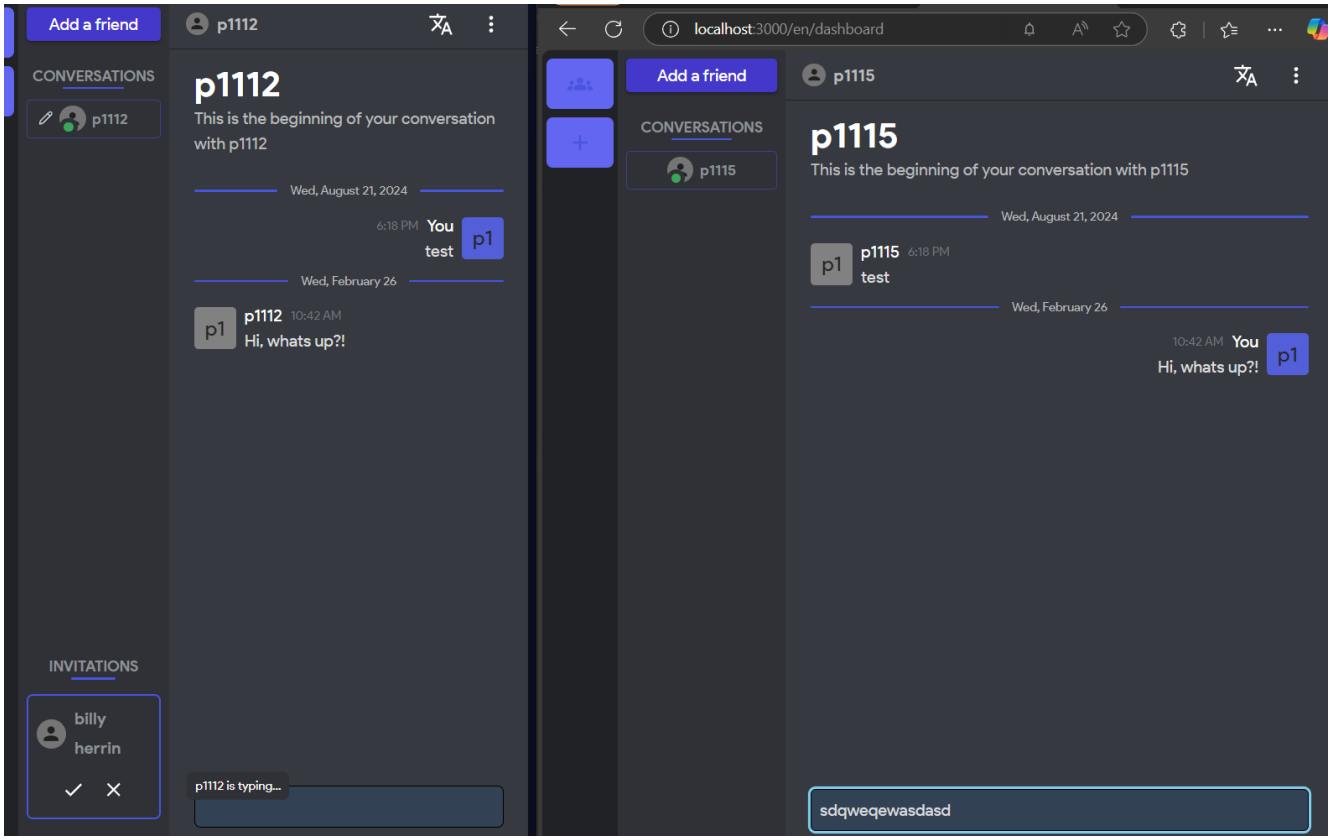

Рисунок 3.14 – Індикатори набору повідомлення і онлайн статусу

Функціонал відеочату та управління кімнатами забезпечує користувачам можливість створення, адміністрування та участі в онлайн-спілкуванні в режимі реального часу. Основна концепція передбачає організацію групових відеоконференцій, де кожен учасник може підключитися до кімнати, взаємодіяти з

іншими користувачами та керувати своїм відеопотоком.

Процес створення кімнати відбувається через інтерфейс бокової панелі, де користувач має можливість ініціювати нову сесію спілкування. При створенні кімнати він автоматично набуває статусу адміністратора, що дозволяє йому контролювати її перебіг. Кімнати мають обмеження за кількістю учасників, що запобігає надмірному навантаженню на систему. Якщо кімната вже містить максимальну кількість учасників, можливість приєднання нових користувачів блокується. Для кожної активної кімнати у списку візуалізується її творець, а також відображається кількість учасників, що спрощує навігацію та вибір відповідної кімнати для приєднання (див. рисунок 3.15).

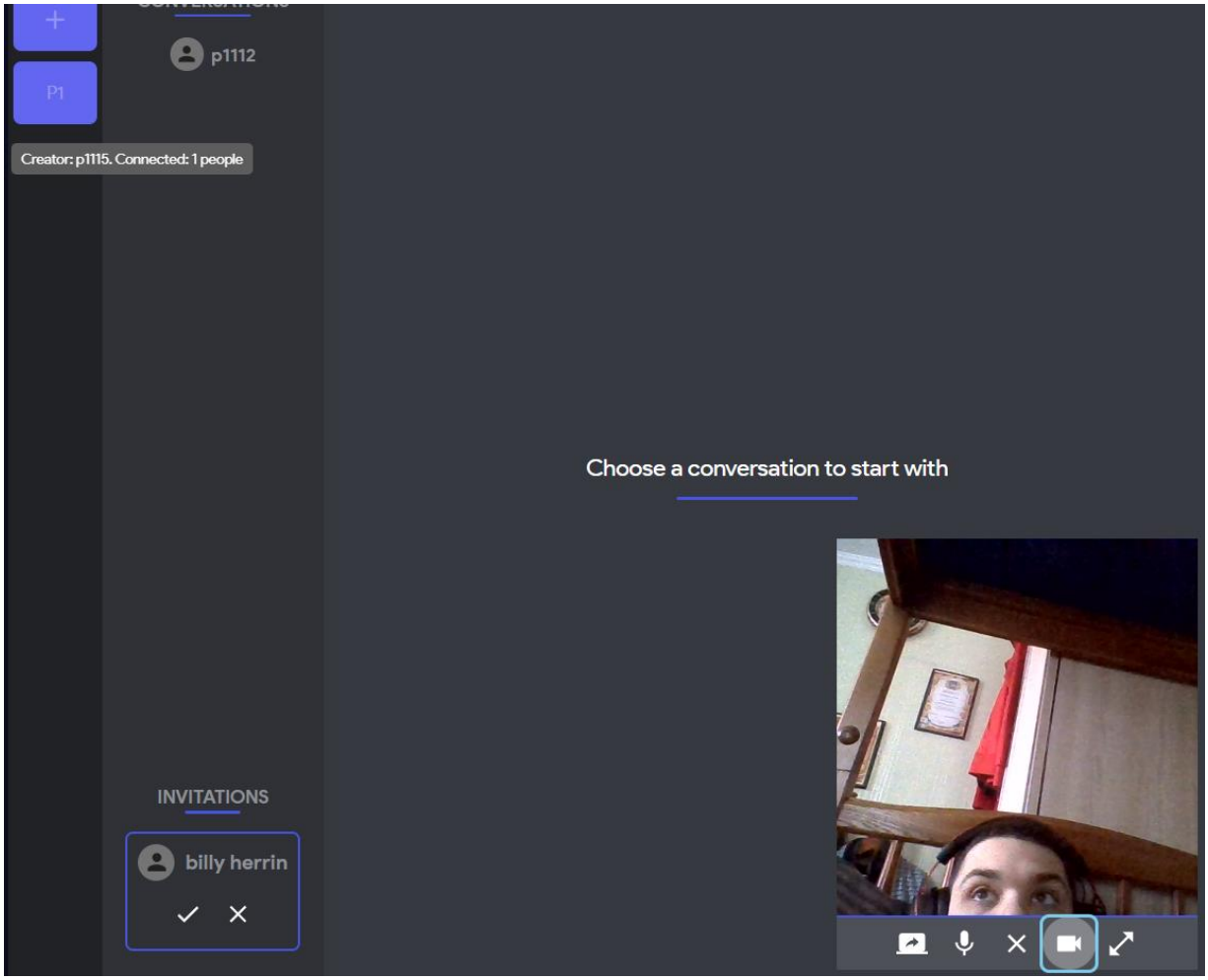

Рисунок 3.15 – Створена кімната з ввімкненою камерою

Функціонал зміни розміру відеочату дозволяє користувачеві перемикатися між мінімізованим та розгорнутим режимами, адаптуючи інтерфейс до своїх потреб. У мінімізованому стані вікно займає меншу частину екрану, що дає змогу користуватися іншими функціями платформи паралельно з відеозв'язком. При розгортанні відеочату на весь екран користувач отримує повноцінний перегляд відеопотоків, що сприяє зручнішій взаємодії під час конференцій. Додатково реалізована функція демонстрації екрану, що дозволяє змінювати вихідний потік відео для демонстрації робочого простору користувача (див. рисунок 3.16).

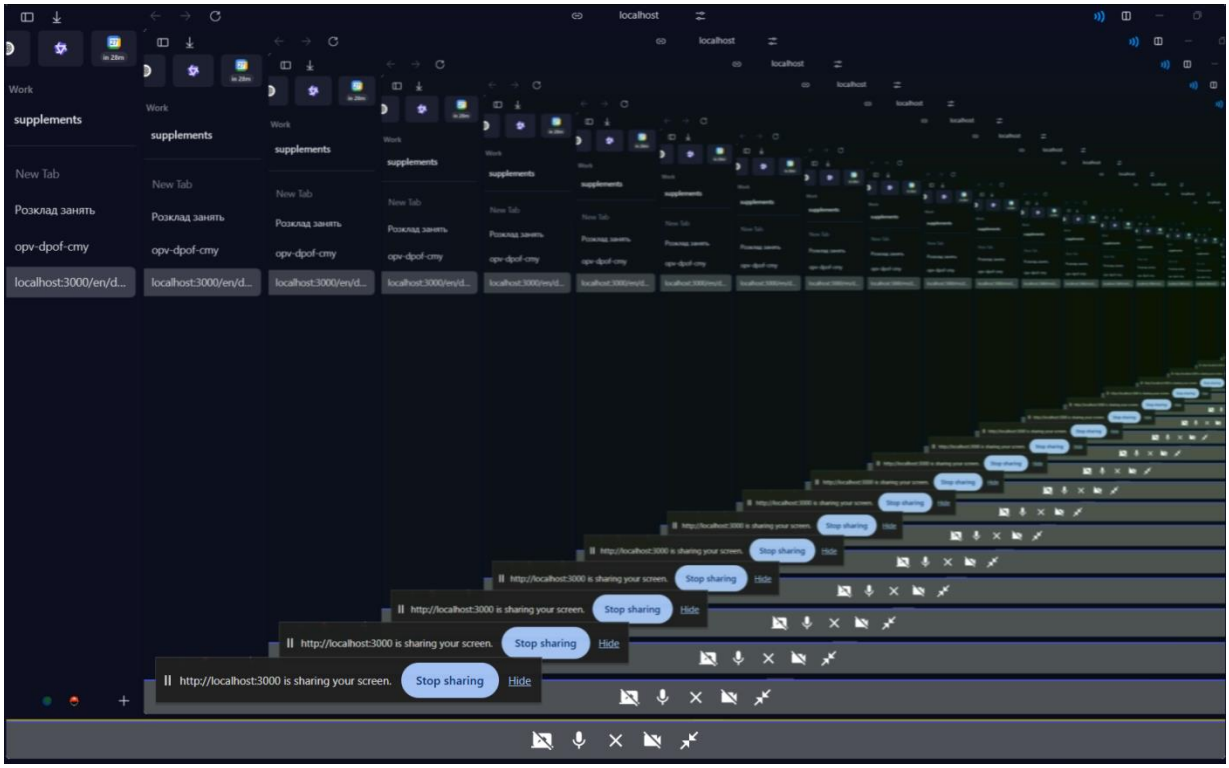

Рисунок 3.16 – Повноекранний режим з поширенням власного екрану

Система відеочату передбачає трансляцію як локального, так і віддалених потоків. Користувач може бачити себе та інших учасників, а також має змогу налаштовувати параметри своєї трансляції. Передбачена можливість вмикання та вимикання мікрофона й камери, що дає змогу контролювати аудіо- та відеопередачу. Інтерфейс містить зручні елементи керування, що дозволяють

мінімізувати або розгортати вікно відеочату, забезпечуючи адаптивність до потреб користувача.

При виході з кімнати система автоматично завершує активні з’єднання, припиняє трансляцію локального відеопотоку та очищує список віддалених потоків. Це гарантує коректне завершення сеансу без залишкових з’єднань, що сприяє оптимальному використанню ресурсів. Інтеграція WebRTC забезпечує ефективну передачу аудіо- та відеосигналу, тоді як WebSocket-з’єднання використовується для синхронізації стану кімнат між користувачами.

3.7 Висновки

У третьому розділі було проведено аналіз і вибір інструментів для розробки вебплатформи, зокрема мови програмування та середовища розробки. Було обґрунтовано доцільність використання мови TypeScript у поєднанні з фреймворками React та Next.js для створення платформи. Такий вибір забезпечує високу продуктивність, гнучкість та можливість ефективно керувати складними інтерфейсами. Як інтегроване середовище розробки було обрано WebStorm завдяки його потужним інструментам для роботи з TypeScript, підтримці сучасних технологій та зручному інтерфейсу.

Також у цьому розділі було розроблено графічні схеми, блок-схеми та діаграми для опису ключових модулів платформи, зокрема модулі авторизації та реєстрації користувачів, списку контактів, здійснення онлайн-спілкування та глобальної оброки помилок. Кожен модуль було детально описано, враховуючи його призначення, функціональні можливості та особливості реалізації. Структура інтерфейсу платформи забезпечує зручність та інтуїтивність у використанні, дозволяючи користувачам легко керувати своїми контактами та підтримувати ефективну онлайн-взаємодію.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення є важливим етапом розробки, який забезпечує її функціональність, зручність та надійність. Цей процес допомагає виявити можливі недоліки на різних етапах створення системи, що дозволяє покращити якість продукту перед його запуском. Основними завданнями тестування є перевірка коректності роботи програми, оцінка її швидкості та стабільності, а також мінімізація ризиків, пов’язаних з помилками, які можуть виникнути під час реального використання.

Методи тестування можна розділити за рівнями та підходами до перевірки. Зокрема, виділяють модульне тестування, яке перевіряє окремі частини системи незалежно одна від одної; інтеграційне тестування, спрямоване на аналіз взаємодії між компонентами; системне тестування, що оцінює роботу програмного забезпечення в цілому; та приймальне тестування, яке проводиться для підтвердження відповідності продукту очікуванням користувачів. Кожен із цих рівнів допомагає забезпечити комплексну перевірку системи [21].

Статичні методи тестування включають аналіз коду без його виконання. До таких методів належать огляд коду, коли інші розробники перевіряють написаний код на наявність помилок, та використання спеціальних інструментів для автоматичного аналізу. Також важливими є формальні перевірки документації та інших матеріалів, що супроводжують розробку.

Динамічні методи тестування передбачають перевірку системи під час її роботи. Серед них виділяють метод "білої скриньки", який вимагає знання внутрішньої структури програми; метод "чорної скриньки", що оцінює програмне забезпечення з точки зору користувача, не враховуючи деталей її реалізації; та метод "сірої скриньки", який поєднує обидва підходи. Ці методи дозволяють оцінити як внутрішню логіку системи, так і її зовнішню поведінку.

Однак тестування супроводжується певними викликами, такими як виявлення прихованих помилок, які проявляються лише за специфічних умов, а також витрати часу та ресурсів. Для подолання цих проблем можуть бути використані автоматизовані інструменти, які прискорюють процес тестування та підвищують його ефективність. Крім того, важливо розробляти стратегії тестування, які враховують різні методи для максимальної перевірки системи.

Аналіз методів тестування показує, що правильний вибір підходів, таких як метод "чорної скриньки", дозволяє оптимізувати процес перевірки та виявляти більшість недоліків до того, як програмне забезпечення потрапить до користувачів. Це забезпечує високу якість продукту та задоволення потреб кінцевих користувачів.

4.2 Тестування розробленого програмного застосунку

Тестування вебплатформи для онлайн-спілкування включає перевірку її функціональності, безпеки та коректності роботи у різних браузерах та на різних пристроях. Цей процес спрямований на забезпечення зручності взаємодії користувачів із системою, а також на підтвердження відповідності платформи встановленим вимогам. Крім того, тестування реєстрації включає перевірку безпеки передачі даних, зокрема шифрування паролів та захист від несанкціонованого доступ

Процес тестування починається з доступу до платформи через браузер. Після цього проводиться серія функціональних тестів, які оцінюють правильність обробки даних та відповідність логіці роботи системи. Особливу увагу приділено тестуванню сторінки авторизації користувачів. Якщо користувач не заповнює обов’язкові поля, наприклад ім’я або електронну адресу, система повинна вивести повідомлення про помилку та попросити заповнити необхідні дані. Результати такої перевірки представлені на рисунку 4.1.

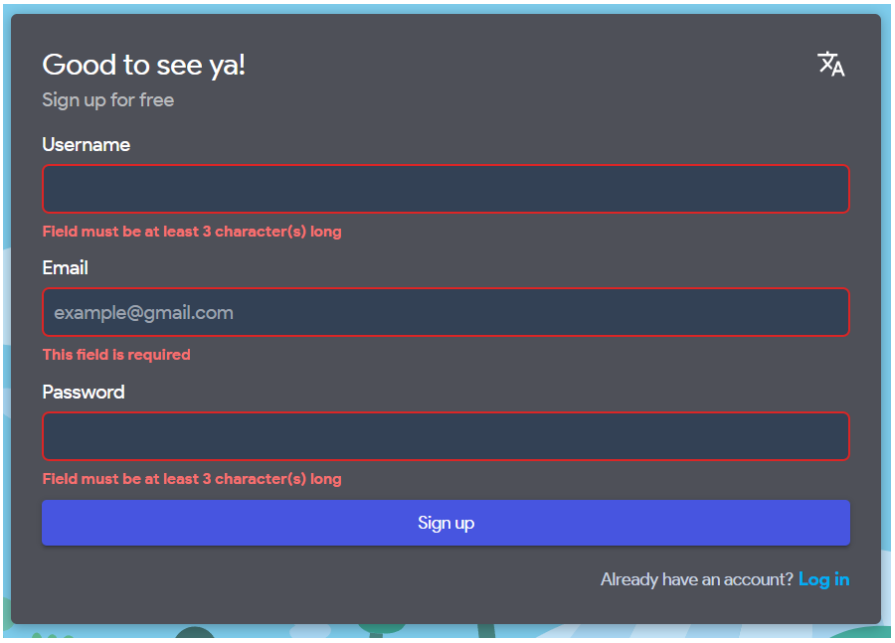

Рисунок 4.1 – Повідомлення про пропуск обов'язкових полів при реєстрації

Наступним етапом є перевірка правильності обробки даних, введених у поле пароля. Наприклад, система має відображати повідомлення про помилку, якщо пароль не відповідає встановленим критеріям безпеки, таким як мінімальна кількість символів або наявність спеціальних знаків. Результати цього тесту наведено на рисунку 4.2.

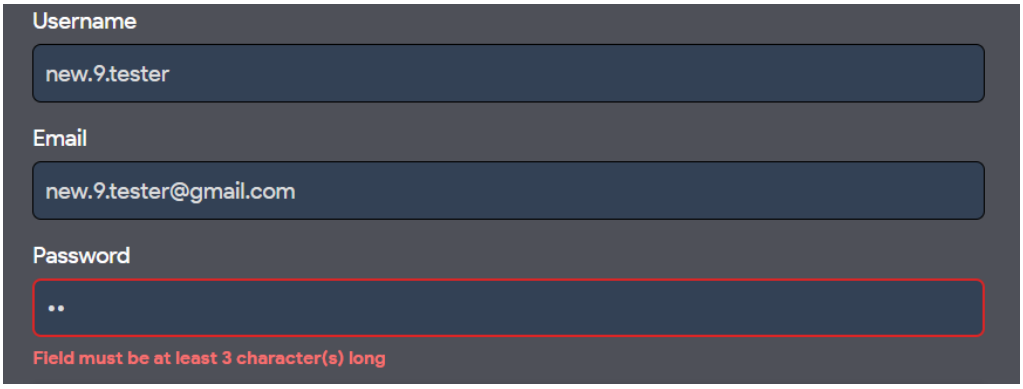

Рисунок 4.2 – Повідомлення про недостатню кількість символів у паролі

Для підтвердження успішної реєстрації необхідно перевірити, чи коректно зберігаються дані користувача у базі даних. Це включає перевірку правильності

збереження таких параметрів, як ім’я, електронна адреса та пароль. На рисунку 4.3 показано стан бази даних до моменту реєстрації нового користувача. Також важливо переконатися, що записи додаються до бази даних без затримок.

Рисунок 4.3 – База даних до реєстрації нового користувача

Після цього вводяться коректні дані (див. рисунок 4.4), і натискається кнопка «Зареєструватися». У разі успішної реєстрації дані користувача зберігаються у базі даних (див. рисунок 4.5), а користувач автоматично переадресовується на головну сторінку платформи (див. рисунок 4.6).

Рисунок 4.4 – Коректно введені дані при реєстрації

	_id ObjectId	email String	password String	username String	friends Array	__v Int32
1	ObjectId('66c59e3ccaf4747…	"pavlienko1115@gmail.com"	"$2a$10$WctQBd8X2DiJHHTD9…	"p1115"	[] 1 elements	2
2	ObjectId('66c59ff0caf4747…	"pavlienko1112@gmail.com"	"$2a$10$.X1jxMzv/SecqHAaO…	"p1112"	[] 1 elements	2
3	ObjectId('67bed80f5f1cb11…	"billyherg@gmail.com"	"$2a$10$LnkFrI7EMrgftnZ2A…	"billy herrin"	[] 0 elements	0
4	ObjectId('67ceb15fa46b1c7…	"new.9.tester@gmail.com"	No field	"new.9.tester"	[] 0 elements	0

Рисунок 4.5 – База даних після реєстрації нового користувача

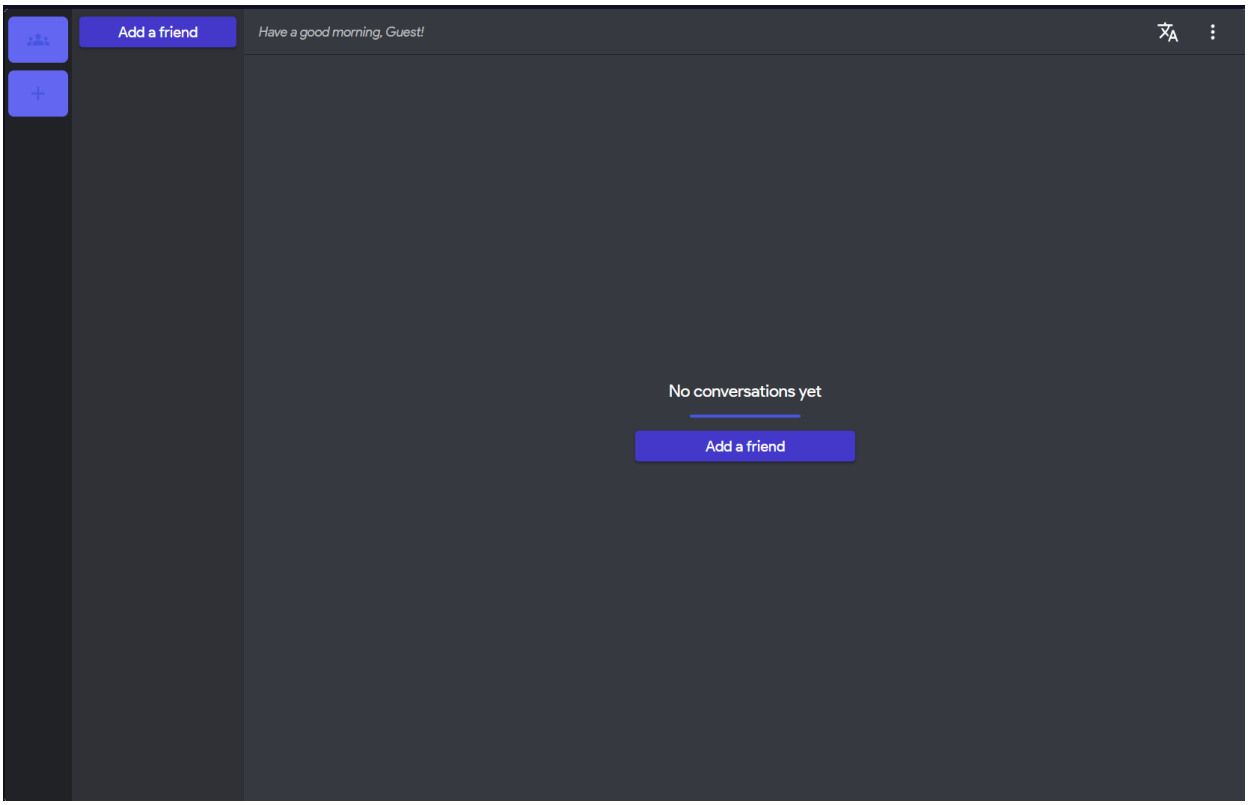

Рисунок 4.6 – Головна сторінка платформи після успішної авторизації

Такий підхід до тестування дозволяє забезпечити надійність та безпеку роботи платформи, а також гарантують комфортну взаємодію користувачів із системою.

Також важливим етапом тестування є перевірка коректності роботи процесу авторизації (див. рисунок 4.7). Перш за все, необхідно переконатися, що всі поля форми авторизації працюють належним чином, включаючи введення логіну та пароля. Потрібно протестувати обробку як правильних, так і неправильних даних, щоб гарантувати, що система адекватно реагує на кожен з випадків. Особливу

увагу слід приділити перевірці повідомлень про помилки, які мають бути зрозумілими та інформативними для користувача. Крім того, важливо забезпечити, щоб після успішної авторизації користувач автоматично переадресовувався на відповідну сторінку платформи.

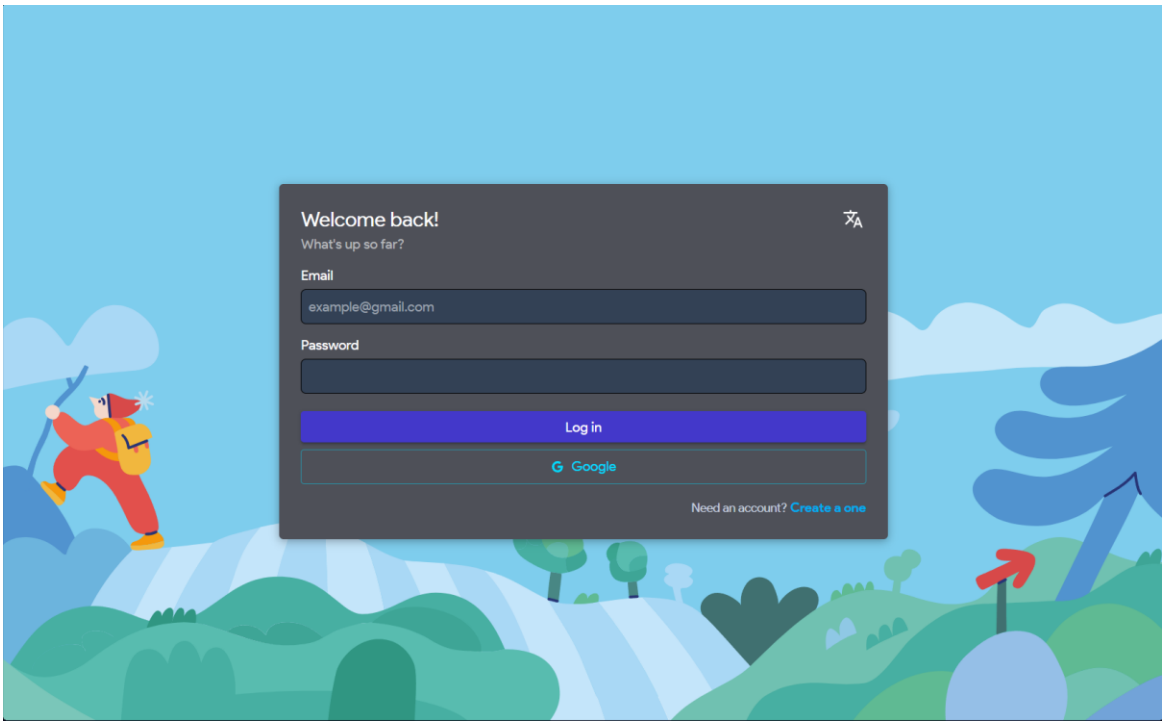

Рисунок 4.7 – Сторінка авторизації користувача

Наступним кроком перевірки є введення даних у відповідні поля, як показано на рисунку 4.8. Необхідно перевірити, чи коректно обробляються введені дані, зокрема правильність формату електронної пошти та пароля. Це допоможе виявити можливі проблеми на ранньому етапі.

Рисунок 4.8 – Заповнені поля форми

Якщо дані введені некоректно, при натисканні кнопки «Увійти» має з’явитися повідомлення про помилку (див. рисунок 4.9). Таке повідомлення має бути чітким і надавати користувачеві інформацію про причину невдалої спроби входу.

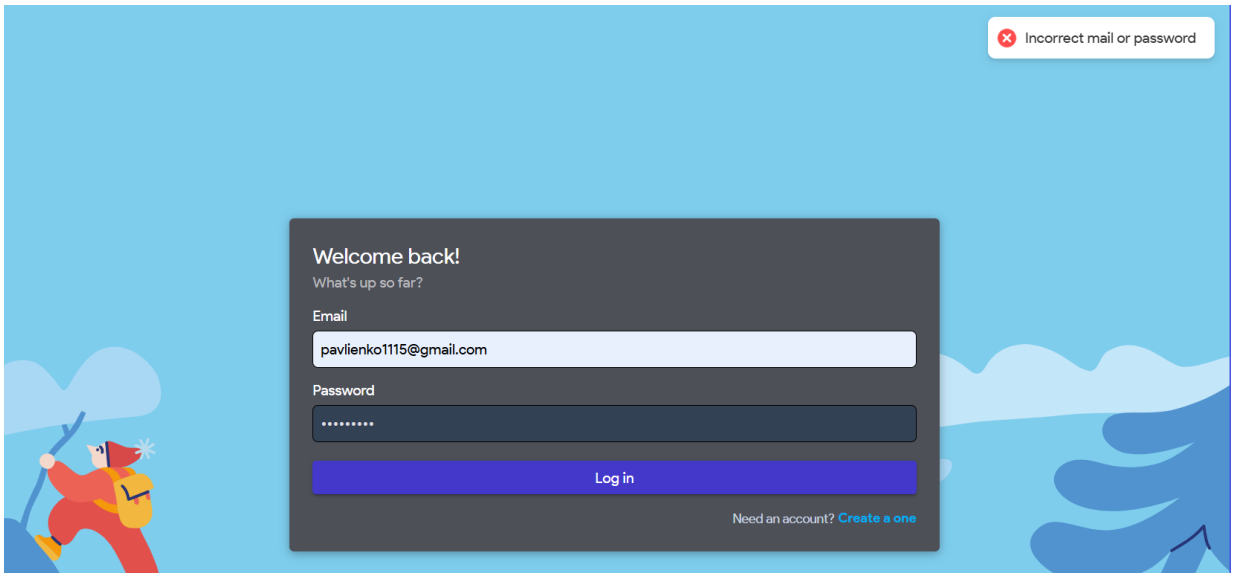

Рисунок 4.9 – Повідомлення про помилку

У разі виправлення помилок користувачем і повторного натискання кнопки «Увійти», система має завантажити головну сторінку (див. рисунок 4.6). На цій сторінці користувачеві демонструється персоналізований вітання з його ім’ям, що створює враження індивідуального підходу. Нижче розташована інформація про найближчі події або активності, що забезпечує зручний доступ до актуальних даних.

Цей комплексний підхід до тестування авторизації дозволяє забезпечити безперебійну роботу системи та комфортну взаємодію користувачів із платформою.

Сторінка чату демонструє особистий діалог із контактом, як показано на рисунку 3.14. В центальній частині розташована історія повідомлень між контактами. У нижній частині екрану розташоване поле для введення тексту та

кнопка для надсилання повідомлення.

Після натискання кнопки для ініціювання відеодзвінка відкривається екран, призначений для проведення онлайн-спілкування

Користувач має можливість керувати налаштуваннями відеодзвінка, вмикаючи або вимикаючи мікрофон та камеру (див. рисунок 3.16). Також доступна функція поширення екрану.

Якщо користувач бажає завершити дзвінок, він може це зробити, натиснувши відповідну кнопку «Х». Після завершення сесії користувач автоматично перенаправляється на головну сторінку платформи. Такий механізм управління сесіями сприяє збереженню безпеки користувачів і знижує ризик несанкціонованого доступу до системи.

4.3 Розробка інструкції користувача вебзастосунку

Для доступу до вебплатформи користувачам не потрібно встановлювати додаткове програмне забезпечення, оскільки всі функції доступні через браузер. Платформа сумісна з основними сучасними браузерами, такими як Google Chrome, Mozilla Firefox, Microsoft Edge та Safari. Користувачам достатньо відвідати офіційний вебсайт платформи, використовуючи посилання, що надається розробником.

Перед початком роботи з платформою важливо переконатися, що пристрій задовольняє мінімальні системні вимоги для коректного відображення інтерфейсу та забезпечення стабільної взаємодії. Інформація про рекомендовані параметри браузера та пристрою наведена в таблицях 4.1 та 4.2. Це допоможе уникнути технічних проблем під час використання.

Після перевірки системних вимог платформа автоматично адаптується до налаштувань браузера, і користувач може починати роботу без додаткових кроків інсталяції. Такий підхід робить платформу доступною та зручною для користувачів, незалежно від їхнього досвіду роботи з вебзастосунками.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	Intel i3-5000
Об’єм оперативної пам’яті	2 ГБ для 64-разрядної системи
Місце на жорсткому диску	1 ГБ
Операційна система	Windows 10, Linux, Mac

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	Intel i5-7000
Об’єм оперативної пам’яті	4 ГБ для 64-разрядної системи
Місце на жорсткому диску	3 ГБ
Операційна система	Windows 11, Linux, Mac

Перед тим як отримати доступ до функціоналу, система перевіряє, чи користувач авторизований. Якщо авторизація не була пройдена, користувачеві пропонується увійти в систему. Без входу в обліковий запис користувач не має можливості розпочати онлайн-спілкування або переглянути історію попередніх діалогів. Після успішної авторизації користувач отримує доступ до ключових функцій платформи, таких як початок нового діалогу, перегляд списку контактів.

У разі успішної авторизації користувач переадресовується на головну сторінку платформи, де може переглянути перелік контактів. На сторінці обраного контакту користувач має змогу ознайомитися з його профілем, включаючи додаткову інформацію.

4.4 Висновки

У четвертому розділі бакалаврської роботи було проведено детальне тестування методів та програмних засобів, розроблених для забезпечення онлайн-комунікацій на веб-платформі.

Крім того, було створено детальну інструкцію для користувачів, яка описує процеси реєстрації, налаштування та взаємодії з платформою через веб-інтерфейс. Інструкція також містить кроки для початку спілкування, управління налаштуваннями профілю та використання додаткових функцій платформи.

Результати тестування підтвердили, що розроблені методи та програмні засоби відповідають встановленим критеріям ефективності, безпеки та зручності використання. Таким чином, проект демонструє значний потенціал для покращення якості онлайн-спілкування та забезпечення комфортної взаємодії між користувачами через використання сучасних технологій комунікації.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено вебсистему для онлайн-комунікацій, яка включає такі функціональні модулі:

– реєстрація облікового запису і авторизація для користувача, в тому числі використовуючи сторонні сервіси;

– функціонал для спілкування онлайн в режимі реального часу: текстовий чат з збереженням історії та онлайн-комунікації з використанням камери і мікрофону користувачів;

– функціонал контактів, з можливістю запрошення.

– розробка локалізації і інтернаціоналізації інтерфейсу.

Для реалізації цієї системи було розроблено метод, модель, алгоритми її роботи. Використано мову програмування JS/TS у поєднанні з фреймворком React/Next для реалізації клієнта; для серверної частини використано Node та Express, а також середовище розробки Webstorm. Робота оформлена з урахуванням рекомендацій [22].

У першому розділі було проведено аналіз сучасних систем для онлайн-комунікацій. Було виявлено їх недоліки, що стало основою для розробки власного застосунку. Другий розділ охоплював аналіз даних, розробку методу і моделі роботи системи, а також побудову алгоритмів роботи застосунку та його окремих модулів. У третьому розділі було обґрунтовано вибір засобів для реалізації програмного забезпечення, проведено варіантний аналіз та розроблено програмні модулі й графічний інтерфейс користувача. Четвертий розділ описує тестування розробленої системи, яке підтвердило її працездатність і відповідність очікуванням. Крім того, визначено мінімальні та рекомендовані вимоги до пристроїв для використання застосунку, а також розроблено інструкцію зі встановлення програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маркевич, К. Цифровізація: переваги та шляхи подолання викликів [Електронний ресурс] – Режим доступу до ресурсу: https://razumkov.org.ua/statti/tsyfrovizatsiia-perevagy-ta-shliakhy-podolannia-vyklykiv.

2. Salvatore, L (2014). Real-Time Communication with WebRTC: Peer-to-Peer in the Browser.

3. Майданюк В.П. Розробка вебсистеми для онлайн-спілкування / В.П. Майданюк, М. І. Павленко / Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії. – Вінниця, 2025. - 384 с. – С. 67-69.

4. Технологія розробки real-time communication з WebRTC [Електронний ресурс] – Режим доступу до ресурсу: https://brander.ua/technologies/webrtc.

5. Сучасна комунікація: особливості мовлення в мережі інтернет [Електронний ресурс] – Режим доступу до ресурсу: http://dspace.nbuv.gov.ua/handle/123456789/178612.

6. Порівняння популярних корпоративних месенджерів [Електронний ресурс] – Режим доступу до ресурсу: https://hurma.work/blog/porivnyannya-populyarnih-korporativnih-mesendzheriv.

7. Zoom [Електронний ресурс] – Режим доступу до ресурсу: https://zoom.com.

8. Tandem [Електронний ресурс] – Режим доступу до ресурсу: https://tandem.chat.

9. Slack [Електронний ресурс] – Режим доступу до ресурсу: https://slack.com.

10. Banks, A. (2020). Learning React with Modern Patterns. Dawn Publishing Ltd.

11. Що таке діаграма класів [Електронний ресурс] – Режим доступу до ресурсу: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram.

12. All You Need to Know about State Diagrams [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/about-state-diagrams.

13. Baumgartner S. (2023). TypeScript Cookbook: Real World Type-Level Programming: O'Reilly Media.

14. Mueller, J. P. (2022). C# 10.0 All-in-One For Dummies. Велика Британія: Wiley.

15. Java Programming Language Explained [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/java.

16. What is PHP? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-is-php-the-php-programming-language-meaning-explained/.

17. WebStorm – NextGen IDE [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/webstorm.

18. Eclipse [Електронний ресурс] – Режим доступу до ресурсу: https://www.eclipse.org.

19. Visual Studio Code - Code Editing. Redefined Guide [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com.

20. What is Visual Studio? [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/visualstudio/get-started/visual-studio-ide

21. Рівні тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalearning.com.ua/theory/lectures/material/testing-levels.

22. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

ДОДАТОК А (обов'язковий). Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою ПЗ

д.т.н., проф.

_____ О. Н. Романюк

"24" березня 2025 р.

Технічне завдання

на бакалаврську кваліфікаційну роботу «Розробка програмного забезпечення платформи для онлайн-комунікацій» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:

_____ к.т.н., доцент В. П. Майданюк

"21" березня 2025 р.

Виконав:

_____ студент гр. 3ПІ-21б М. І. Павленко

"21" березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного забезпечення платформи для онлайн-комунікацій».

Галузь застосування – сфера онлайн-спілкування: освітні заклади, бізнес-організації та соціальні платформи.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою дослідження є покращення ефективності інтерактивного відео- та текстового спілкування і комунікаційних можливостей, яке забезпечить користувачам зручність використання та високий рівень безпеки.

Призначення роботи – розробка вебзастосунку для онлайн-комунікацій.

4. Вихідні дані для проведення НДР

1. Сучасна комунікація: особливості мовлення в мережі інтернет [Електронний ресурс] – Режим доступу до ресурсу: http://dspace.nbuv.gov.ua/handle/123456789/178612.

2. Baumgartner S. (2023). TypeScript Cookbook: Real World Type-Level Programming: O’Reilly Media.

3. Рівні тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalearning.com.ua/theory/lectures/material/testing-levels.

5. Технічні вимоги

Комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 8 ГБ, розмір жорсткого диску 256 ГБ. Операційна система Windows 10. ПК з будь-якою операційною системою та будь-який браузер.

6. Конструктивні вимоги

Вебзастосунок та повинен відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку програм для комунікацій в системах для онлайн-спілкування та постановка задач розробки	25.03.25 – 31.03.25
2	Розробка методів, моделі та алгоритмів роботи програми	01.04.25 – 14.04.25
3	Розробка програмного забезпечення	15.04.25 – 12.05.25
4	Тестування програми	13.05.25 – 15.05.25
5	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

ДОДАТОК В (довідниковий). Лістинг програми

```
import axios from “axios’
import {deleteCookie, getCookie} from “cookies-next’;
import {CookieKey} from “@/shared/models/cookie.model’;

export const apiClient = axios.create({
  baseURL: "http://localhost:5000/api",
  timeout: 2000,
});

apiClient.interceptors.request.use((config) => {
      const token = getCookie(CookieKey.Token)?.toString();

      if (token) {
        config.headers.Authorization = `Bearer ${token}`;
      }

      return config;
    },
    (err) => {
      return Promise.reject(err);
    }
);

apiClient.interceptors.response.use(config => config, async (error) => {
  if (!error.response || isUnauthorized(error.response.status)) {
     handleLogout();
     location.replace(“/login’);
  }
  return Promise.reject(error);
})
```

ДОДАТОК Б (обов'язковий). Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного забезпечення платформи для онлайн-комунікацій.

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Майданюк В. П.

Показники звіту подібності

Оригінальність	88 %
Загальна схожість	12 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Strikeplagiarism.

Автор роботи _____ Павленко М. І.

Керівник роботи _____ Майданюк В. П.

```
export function handleLogout() {
  deleteCookie(CookieKey.Token);
}

export const isUnauthorized = (responseCode: number) => {
  return (responseCode === 403);
};

import {io, Socket} from "socket.io-client";
import {DefaultEventsMap} from "@socket.io/component-emitter";
import {AppMiddlewareAPI} from "@/store/store";
import {getSocketHandlers} from "@/api/socket.io/handlers";
import {roomSlice} from "@/store/room";
import { SocketEvent } from "@/shared/models/socket.model";

export type IoSocket = Socket<DefaultEventsMap, DefaultEventsMap>;
let socket: IoSocket = null!;

export const connectSocketServer = ({jwtToken, state}: { jwtToken: string, state: AppMiddlewareAPI
}) => {
  const socketHandlers = getSocketHandlers(state);
  socket ??= io("localhost:5000', {
    auth: {token: jwtToken,}
  });

  socket.on("connect', () => {
    console.log("connected with socket.io', socket)
  });

  Object.entries(socketHandlers).forEach(([propName, handler]) => {
    socket.on(propName, handler(socket))
  });
```

```
  return socket;
}

export function sendDirectMessage(data: {message: { receiverID: string, content: string }}) {
  socket.emit(SocketEvent.SendNewDirectMessage, data);
}

export function initialUpdateDirectChatHistory(data: { friendID: string }) {
  socket.emit(SocketEvent.UpdateDirectChatHistory, data);
}

export function emitUserTypingMessage(data: { friendID: string }) {
  socket.emit(SocketEvent.UserTyping, data);
}

export function leaveRoom(roomID: string) {
  socket.emit(SocketEvent.LeaveRoom, {roomID});
}

export function createNewRoom() {
  socket.emit(SocketEvent.CreateRoom);
}

export function joinRoom(roomData: { roomID: string }) {
  socket.emit(SocketEvent.JoinRoom, roomData);
}

export function signalPeerData<T extends {}>(signalData: T) {
  socket.emit(SocketEvent.RTCSignal, signalData);
}

import {chatActions} from "@/store/chat";
```

```
import {AppMiddlewareAPI} from "@/store/store";
import {friendsActions} from "@/store/friends";
import {roomSlice} from "@/store/room";
import {UserData} from "@/store/room/types";
import {Friend, FriendInvitation} from "@/store/friends/types";
import {CHAT_TYPES, Message} from "@/store/chat/types";
import {getPeerConfiguration, handleParticipantLeftRoom, handleRTCFulfilledSignal,
prepareNewRTCConnection} from "@/api/webRTC";
import {IoSocket} from "@/api/socket.io/connection";
import Peer from "simple-peer";
import { SocketEvent } from "@/shared/models/socket.model";
import toast from “react-hot-toast’;
import {UserResponse} from “@/store/user/types’;
import {getTranslation} from “@/i18n/routing’;

export type Room = {
  roomCreator: UserData,
  participants: UserData[],
  roomID: string,
}

export type SignalWithSocketID = { signal: Peer.SignalData, connectingUserSocketID: string }
type CurrentUserSocket = { currentlyConnectingUserSocketID: string };

export function getSocketHandlers({dispatch, getState}: AppMiddlewareAPI) {

  return {
    [SocketEvent.GetDirectChatHistory]: () => ({messages, participants, isInitial}: {
      messages: (Message & { author: { _id: string } })[],
      participants: any[],
      isInitial: boolean
    }) => {
```

```
    const {content, author} = messages.at(-1)!;
    const {username, _id} = author;
    const currentlyLoggedUser = getState().auth.user!;
    const shouldShowNotification = !document.hasFocus() && !isInitial &&
      currentlyLoggedUser.username !== username;

    dispatch({
      type: chatActions.SET_MESSAGES.type,
      payload: messages,
    })

    shouldShowNotification && Notification.requestPermission().then(() => {
      createNotification(`New message from ` + username, {
        body: content.length >= 50 ? content.slice(0, 50) + "...' : content,
        icon: "next.svg'
      }, () => {
        console.log("Notification clicked', author);
        dispatch({
          type: chatActions.SET_CHAT_DETAILS.type,
          payload: {type: CHAT_TYPES.DIRECT, friend: {username, userID: _id}}
        });
      });
    });
  },

  [SocketEvent.UpdateFriendInvitations]: () => ({friendInvitations}: { friendInvitations:
FriendInvitation[] }) => {
    dispatch({
      type: friendsActions.SET_FRIEND_INVITATIONS.type,
      payload: friendInvitations
    })
  },
```

```
    [SocketEvent.UpdateFriends]: () => ({friends}: { friends: Friend[] }) => {
      dispatch({
        type: friendsActions.SET_FRIENDS.type,
        payload: friends
      })
    },

    [SocketEvent.UpdateOnlineUsers]: () => ({users}: { users: Record<string, Friend> }) => {
      dispatch({
        type: friendsActions.SET_ONLINE_USERS.type,
        payload: users
      })
    },

    [SocketEvent.UserTyping]: () => ({username, isTyping}: { username: string, isTyping: boolean })
=> {

      dispatch({
        type: chatActions.SET_CURRENTLY_TYPING_DATA.type,
        payload: {
          isTyping,
          username
        }
      })
    },

    [SocketEvent.CreateRoom]: () => ({roomDetails}: { roomDetails: { roomCreator: UserData,
participants: UserData[], roomID: string, } }) => {
      dispatch({
        type: roomSlice.actions.SET_ROOM_DETAILS.type,
        payload: roomDetails,
      })
    },
```

```
    [SocketEvent.UpdateActiveRooms]: () => ({activeRooms}: { activeRooms: Room[] }) => {
      const {friends} = getState().friends;
      // const shouldSetActiveRooms = activeRooms.some(({roomCreator}) => {
      //    return friends.some(friend => roomCreator.userID === friend.userID);
      // });

      dispatch({
        type:  roomSlice.actions.SET_ACTIVE_ROOMS.type,
        payload: activeRooms,
      });
    },
    [SocketEvent.PrepareRTC]: (socket) => ({currentlyConnectingUserSocketID}:
CurrentUserSocket) => {
      prepareNewRTCConnection({currentlyConnectingUserSocketID, peerConfig: {
          initiator: false,
          config: getPeerConfiguration(),
          stream: getState().room.localStream!,
        }});
      socket.emit(SocketEvent.InitRTC, {currentlyConnectingUserSocketID});
    },
    [SocketEvent.RTCInitResponse]: () => ({currentlyConnectingUserSocketID}: CurrentUserSocket)
=> {
      prepareNewRTCConnection({currentlyConnectingUserSocketID, peerConfig: {
          initiator: true,
          config: getPeerConfiguration(),
          stream: getState().room.localStream!,
        }});
    },
    [SocketEvent.RTCSignalFulfilled]: () => (data: SignalWithSocketID) => {
      handleRTCFulfilledSignal(data);
    },
    [SocketEvent.LeaveRoom]: () => ({leftParticipantSocketID}: { leftParticipantSocketID: string })
```

```
=> {
        handleParticipantLeftRoom(leftParticipantSocketID);
      },
      [SocketEvent.NotifyFriendRoomCreated]: () => async ({user}: {user: UserResponse}) => {
        const t = await getTranslation("Notification.Success')
        debugger
        toast.success(t("friendRoomCreated', {
          username: user.username
        }), {
          icon: "🚪',
        });
      }
    } satisfies Partial<Record<SocketEvent, (socket: IoSocket) => (...args: any[]) => void>>
}

function createNotification(title: string, options: NotificationOptions, onClick: () => void) {
  const notification = new Notification(title, options);

  notification.onclick = () => {
    onClick();
    window.focus();
  }
}

import {store} from "@/store/store";
import {roomSlice} from "@/store/room";
import Peer, {Options} from "simple-peer';
import {signalPeerData} from "@/api/socket.io/connection";
import {SignalWithSocketID} from "@/api/socket.io/handlers";
import toast from "react-hot-toast';
import {getTranslation} from "@/i18n/routing';
```

```
export type StreamHandler = (stream: MediaStream) => void

const peers: Record<string, Peer.Instance> = {};

const onlyAudioConstraints = {
  audio: true,
  video: false,
};

const defaultConstraints = {
  video: true,
  audio: true,
};

export const getPeerConfiguration = (): RTCConfiguration => {
  const turnIceServer = null;

  if (turnIceServer) {
    return {}
  } else {
    return {
      iceServers: [
        {
          urls: "stun:stun.l.google.com:19302',
        }
      ]
    }
  }
}

export function handleParticipantLeftRoom(leftParticipantSocketID: string) {
  destroyPeer(leftParticipantSocketID);
  store.dispatch({type: roomSlice.actions.FILTER_REMOTE_STREAMS.type, payload:
```

```
leftParticipantSocketID});
}

function destroyPeer(userSocketID: string) {
  const peer = peers[userSocketID];
  if(!peer) return;
  peer.destroy();
  delete peers[userSocketID];
}

export const closeAllConnections = () => {
  Object.entries(peers).forEach(([userSocketId]) => {
    destroyPeer(userSocketId)
  });
};

export function prepareNewRTCConnection({peerConfig, currentlyConnectingUserSocketID}: {
  currentlyConnectingUserSocketID: string,
  peerConfig: Options
}) {
  peers[currentlyConnectingUserSocketID] = new Peer(peerConfig);

  peers[currentlyConnectingUserSocketID].on("signal', (signal) => {
    const signalData = {
      signal,
      connectingUserSocketID: currentlyConnectingUserSocketID,
    };
    signalPeerData(signalData);
  })

  peers[currentlyConnectingUserSocketID].on("stream', (remoteStream) => {
    console.log("new stream event', remoteStream);
    const payload = {...remoteStream, currentlyConnectingUserSocketID};
```

```
    store.dispatch({type: roomSlice.actions.PUSH_REMOTE_STREAM.type, payload});
  });
}

export const getLocalStreamPreview = (onlyAudio = false, onSuccess?: StreamHandler) => {
  const constraints = onlyAudio ? onlyAudioConstraints : defaultConstraints;

  navigator.mediaDevices.getUserMedia(constraints)
    .then((stream) => {
      if (Object.entries(stream.getTracks()).length === 0) return;
      onSuccess && onSuccess(stream);
    })
    .catch(async (err) => {
      console.log(err, "err");
      const t = await getTranslation("Notification.Error");
      toast.error(t("localStream"), err);
    });
};

export const switchOutgoingTracks = (stream: MediaStream) => {
  for (let socketId in peers) {
    const peerStream = peers[socketId].streams[0];

    for (let peerIndex in peerStream.getTracks()) {
      const peerTrack = peerStream.getTracks()[peerIndex];

      for (let streamIndex in stream.getTracks()) {
        const streamTrack = stream.getTracks()[streamIndex];

        if (peerTrack.kind === streamTrack.kind) {
          peers[socketId].replaceTrack(peerTrack, streamTrack, peerStream);
          break;
        }
```

```
        }
      }
    }
};

export function handleRTCFulfilledSignal({signal, connectingUserSocketID}: SignalWithSocketID) {
  console.log("peers', peers, {signal, connectingUserSocketID}, peers[connectingUserSocketID])
  if (!peers[connectingUserSocketID]) return;
  peers[connectingUserSocketID].signal(signal);
}

import { getRequestConfig } from "next-intl/server';

export enum Locale {
  English = "en',
  Ukrainian = "uk',
  Deutsch = "de',
  Spanisch = "es',
}

export const DEFAULT_LOCALE = Locale.English;
export const LOCALE_OPTIONS = Object.values(Locale);

export default getRequestConfig(async ({ requestLocale }) => {
  let locale = await requestLocale;
  if (!locale || !LOCALE_OPTIONS.includes(locale as Locale)) locale = DEFAULT_LOCALE;

  return {
    messages: (await import(`../../messages/${locale}.json`)).default,
    locale,
  };
});
```

```
import { DEFAULT_LOCALE, LOCALE_OPTIONS, Locale } from “@/i18n/request’;
import { CookieKey } from “@/shared/models/cookie.model’;
import { getCookie } from “cookies-next’;
import { createTranslator } from “next-intl’;
import { createNavigation } from “next-intl/navigation’;
import { defineRouting } from “next-intl/routing’;

export const routing = defineRouting({
  locales: LOCALE_OPTIONS,
  defaultLocale: Locale.English,
});

// Lightweight wrappers around Next.js’ navigation APIs
// that will consider the routing configuration
export const { Link, redirect, usePathname, useRouter, getPathname } = createNavigation(routing);

async function loadMessages(locale: string) {
  try {
    return await import(`../../messages/${locale}.json`);
  } catch (error) {
    console.error(`Error loading translations for locale: ${locale}`, error);
    return {};
  }
}

export async function getTranslation(namespace?: string) {
  const locale = (await getCookie(CookieKey.Locale)) || DEFAULT_LOCALE;
  const messages = await loadMessages(locale);
  return createTranslator({ locale, messages, namespace });
}

@tailwind base;
@tailwind components;
```

```
@tailwind utilities;

@import url("https://fonts.cdnfonts.com/css/google-sans');

:root {
  --foreground-rgb: 0, 0, 0;
  --background-start-rgb: 214, 219, 220;
  --background-end-rgb: 255, 255, 255;
  --light-gray: #4e5058;
  --dark-gray: 43, 45, 49;
  --violet-rgb: 71, 85, 224;
  --cyan-rgb: 126, 205, 237;
  --orange-rgb: 252, 168, 120;
}

@media (prefers-color-scheme: dark) {
  :root {
    --foreground-rgb: 255, 255, 255;
    --background-start-rgb: 0, 0, 0;
    --background-end-rgb: 0, 0, 0;
  }
}

body {
  color: rgb(var(--foreground-rgb));
  background-color: var(--light-gray);
  font-family: "Product Sans', "Helvetica Neue', sans-serif;
}

*::selection {
  background-color: rgb(var(--violet-rgb));
  color: rgb(230, 230, 230);
}
```

```
* {
  transition: .2s all ease-in-out, .4s max-height ease-in-out;

  &:focus {
    outline: none;
    box-shadow: 0 0 0 3px transparent;
  }

  &:focus:focus-visible {
    box-shadow: 0 0 0 2.5px rgb(var(--cyan-rgb));
    border-radius: 5px;
  }
}

.css-1t1j96h-MuiPaper-root-MuiDialog-paper {
  background-color: var(--light-gray);
}

::-webkit-scrollbar {
  width: 4px;
}

::-webkit-scrollbar-thumb {
  border-radius: 7px;
  background: rgb(var(--violet-rgb));
}

.MuiButton-root {
  text-transform: initial !important;
}

import * as Yup from “yup’;
```

```
import {Dictionary} from “@/shared/types/i18n’;

const email = (t: Dictionary) =>
Yup.string().email(t(“FieldError.invalidEmail’)).required(t(“FieldError.required’));
const password = (t: Dictionary) => Yup.string()
    .min(3, t(“FieldError.minLength’, {
       minLength: 3,
    }))
    .required(t(“FieldError.required’));

export const SCHEMAS = (t: Dictionary) => ({
  email: email(t),
  password: password(t),
  sendFriendInvitation: Yup.object({
    email: email(t),
  }),
  sendMessage: Yup.object({
    message: Yup.string()
      .min(1, t(“FieldError.minLength’, {
        minLength: 1,
      }))
      .max(500, t(“FieldError.maxLength’, {
        maxLength: 500,
      }))
      .required(t(“FieldError.required’)),
  }),
  register: Yup.object({
    username: Yup.string()
      .min(3, t(“FieldError.minLength’, {
        minLength: 3,
      }))
      .max(20, t(“FieldError.maxLength’, {
        maxLength: 20,
```

```
        }))
        .required(t("FieldError.required")),
      email: email(t),
      password: password(t),
    }),
    login: Yup.object({
      email: email(t),
      password: password(t)
    }),
    generateMessage: Yup.object({
      messageLimit: Yup.number()
        .min(0, t("FieldError.min", {min: 0}))
        .max(50, t("FieldError.max", {max: 50})),
      instructions: Yup.string(),
      generatedMessage: Yup.string(),
    })
} satisfies Record<string, Yup.Schema>);

import { DEFAULT_LOCALE } from "@/i18n/request";
import { routing } from "@/i18n/routing";
import { extractLocaleAndPathname } from "@/shared/utils/url";
import createMiddleware from "next-intl/middleware";
import { NextRequest, NextResponse } from "next/server";
import { CookieKey } from "./shared/models/cookie.model";
import {cookies} from "next/headers";

const intlMiddleware = createMiddleware(routing);

const PRIVATE_ROUTES = ["/dashboard", "/profile", "/settings", ];
const AUTH_ROUTES = ["/login", "/register", ];

export const config = {
  matcher: [
```

```
    "/", // Root
    "/((?!api|_next|favicon.ico|robots.txt|sitemap.xml).*)", // Exclude API and other static assets
  ],
};

export default async function middleware(request: NextRequest): Promise<NextResponse | undefined>
{
  const { pathname } = request.nextUrl;
  const cookieStore = await cookies();
  const accessToken = cookieStore.get(CookieKey.Token)?.value;
  let { pathnameWithoutLocale, locale } = extractLocaleAndPathname(pathname);
  const savedLocale = cookieStore.get(CookieKey.Locale)?.value;
  let redirectURL = "";

  if (!locale || !routing.locales.includes(locale as never)) return
NextResponse.redirect(`http://localhost:3000/${DEFAULT_LOCALE}${pathnameWithoutLocale}`);
  locale = locale || savedLocale || DEFAULT_LOCALE;

  const isPrivateRoute = PRIVATE_ROUTES.some(route =>
pathnameWithoutLocale.startsWith(route));

  if ((isPrivateRoute && !accessToken)) redirectURL = `/login`;
  if (redirectURL) return NextResponse.redirect(`http://localhost:3000/${locale}${redirectURL}`);

  return intlMiddleware(request);
}

import {createSlice, PayloadAction} from "@reduxjs/toolkit";

import {Friend} from "@/store/friends/types";
import {CHAT_TYPES, Message} from "@/store/chat/types";

export type ChatState = {
```

```
    messages: Message[],
    chat: {
        type: CHAT_TYPES,
        friend: Friend | null,
        currentlyTypingData: {
            isTyping: boolean,
            username: string,
        }
    }
}

const initialState: ChatState = {
    messages: [],
    chat: {
        friend: null,
        type: CHAT_TYPES.NONE,
        currentlyTypingData: {
            isTyping: false,
            username: “”,
        }
    }
}

const chatSlice = createSlice({
    name: “chat’,
    initialState,
    reducers: {
        SET_CHAT_DETAILS: (state, action: PayloadAction<Partial<ChatState[“chat’]>>) => {
            state.chat = {...state.chat, ...action.payload};
        },
        SET_MESSAGES: (state, action: PayloadAction<Message[]>) => {
            state.messages = action.payload;
        },
```

```
        SET_CURRENTLY_TYPING_DATA: (state, action:
PayloadAction<ChatState[“chat’][“currentlyTypingData’]>) => {
            state.chat.currentlyTypingData = action.payload;
        },
    }
})

export default chatSlice.reducer;
export const chatActions = chatSlice.actions;

import {createSlice, PayloadAction} from "@reduxjs/toolkit";
import {Friend, FriendInvitation} from "@/store/friends/types";

export type FriendsState = {
    friends: Friend[],
    friendInvitations: FriendInvitation[],
    onlineUsers: Record<string, Friend>
}

const initialState: FriendsState = {
    friends: [],
    friendInvitations: [],
    onlineUsers: {},
}

const friendsSlice = createSlice({
    name: “friends’,
    initialState,
    reducers: {
        SET_FRIENDS: (state, action: PayloadAction<Friend[]>) => {
            state.friends = action.payload
        },
```

```
        SET_FRIEND_INVITATIONS: (state, action: PayloadAction<FriendInvitation[]>) => {
            state.friendInvitations = action.payload
        },
        APPEND_PENDING_FRIEND_INVITATIONS: (state, action:
PayloadAction<FriendInvitation[]>) => {
            state.friendInvitations.push(...action.payload)
        },
        SET_ONLINE_USERS: (state, action: PayloadAction<Record<string, Friend>>) => {
            state.onlineUsers = action.payload;
        },
    },

})

export default friendsSlice.reducer;
export const friendsActions = friendsSlice.actions;

import {Middleware} from "redux";
import {apiClient, } from "@/api/axios";
import {RootState} from "@/store/store";
import {authActions} from "@/store/user";
import {connectSocketServer} from "@/api/socket.io/connection";
import {User, UserResponse} from "@/store/user/types';
import {setCookie} from "cookies-next';
import {CookieKey} from "@/shared/models/cookie.model';

interface AuthMiddlewareAction {
    type: string;
    payload?: {
            user: UserResponse;
            token: string;
        }
        |
```

```
    {
      data: {
        error: string;
      };
      notification: string;
      status: number;
    };
}

function isAuthPayloadSuccess(payload: Record<string, any>): payload is { user: UserResponse } {
  return typeof payload === “object’ && “user’ in payload;
}

export const authMiddleware: Middleware<{}, RootState> = (state) => next => (action:
AuthMiddlewareAction) => {
  const {payload} = action;
  if (!payload) return next(action);

  const isAuthSuccessful = /auth\/.+\/fulfilled/.test(action.type);
  if (isAuthPayloadSuccess(payload) && isAuthSuccessful) {
    const {user, token} = payload;
    setCookie(CookieKey.Token, token);
    apiClient.defaults.headers.common[“Authorization’] = `Bearer ${token}`;
    state.dispatch({type: authActions.SET_USER.type, payload: user});
    connectSocketServer({jwtToken: token, state});
    Notification.requestPermission();
  }

  return next(action);
};

import {createSlice, PayloadAction} from "@reduxjs/toolkit";
import {createNewRoom, joinRoom, leaveRoom} from “@/api/socket.io/connection’;
```

```
import {closeAllConnections, getLocalStreamPreview, StreamHandler} from "@/api/webRTC";
import {Room} from "@/api/socket.io/handlers";
import {NullableMediaStream, RoomDetails} from "@/store/room/types";
import {store} from “@/store/store’;

export type RoomState = {
  user: {
    isInRoom: boolean,
    isRoomCreator: boolean,
    isWithOnlyAudio: boolean,
  },
  roomDetails: RoomDetails | null,
  activeRooms: Room[],
  localStream: NullableMediaStream,
  remoteStreams: MediaStream[],
  audioOnly: boolean,
  isScreenShareActive: boolean,
  screenShareStream: NullableMediaStream,
}

const initialState: RoomState = {
  user: {
    isInRoom: false,
    isRoomCreator: false,
    isWithOnlyAudio: false,
  },
  roomDetails: null,
  activeRooms: [],
  localStream: null,
  remoteStreams: [],
  audioOnly: false,
  isScreenShareActive: false,
  screenShareStream: null
```

```
}

export const roomSlice = createSlice({
  name: “room’,
  initialState,
  reducers: {
    JOIN_ROOM: (state, {payload: {roomID}}: PayloadAction<{ roomID: string }>) => {
      state.roomDetails = {...state.roomDetails!, roomID};
      state.user = {...state.user, isInRoom: true, isRoomCreator: false}

      getLocalStreamPreview(state.audioOnly, (stream) => {
        store.dispatch({type: roomSlice.actions.SET_LOCAL_STREAM.type, payload: stream});
        joinRoom({roomID});
      });
    },
    LEAVE_ROOM: (state) => {
      const {localStream, roomDetails, screenShareStream} = state;
      console.log(state.roomDetails, “state.roomDetails’);

      leaveRoom(state.roomDetails!.roomID);

      if (localStream) {
        localStream.getTracks().forEach((track) => track.stop());
        state.localStream = initialState.localStream;
      }

      if (screenShareStream) {
        screenShareStream.getTracks().forEach((track) => track.stop());
        state.screenShareStream = initialState.screenShareStream;
      }

      state.remoteStreams = initialState.remoteStreams;
      closeAllConnections();
```

```
      if(!roomDetails) return;

      state.roomDetails = initialState.roomDetails;
      state.user = initialState.user;
    },
    CREATE_ROOM: (state, action: PayloadAction<Partial<RoomState[“user’]>>) => {
      const {audioOnly} = state;
      state.user = {...state.user, ...action.payload};

      getLocalStreamPreview(audioOnly, stream => {
        store.dispatch({type: roomSlice.actions.SET_LOCAL_STREAM.type, payload: stream});
        createNewRoom();
      });
    },
    SET_ACTIVE_ROOMS: (state, action: PayloadAction<RoomState[“activeRooms’]>) => {
      state.activeRooms = action.payload;
    },
    SET_LOCAL_STREAM: (state, action: PayloadAction<MediaStream>) => {
      state.localStream = action.payload;
    },
    SET_ROOM_DETAILS: (state, action: PayloadAction<RoomState[“roomDetails’]>) => {
      state.roomDetails = action.payload;
    },
    PUSH_REMOTE_STREAM: (state, action:
PayloadAction<RoomState[“remoteStreams’][number]>) => {
      state.remoteStreams.push(action.payload);
    },
    FILTER_REMOTE_STREAMS: (state, {payload}: PayloadAction<string>) => {
      // @ts-ignore
      state.remoteStreams  = state.remoteStreams.filter(stream => stream.connectingUserSocketID !==
payload);
    },
```

```
    SET_AUDIO_ONLY: (state, action: PayloadAction<boolean>) => {
      state.audioOnly = action.payload;
    },
    SET_SCREEN_SHARE_STREAM: (state, action: PayloadAction<NullableMediaStream>) => {
      state.screenShareStream = action.payload;
      state.isScreenShareActive = !!action.payload;
    }
  },
})

import {createSlice, PayloadAction} from "@reduxjs/toolkit";
import {login} from "@/store/user/actions/thunks/login";
import {register} from "@/store/user/actions/thunks/register";
import {handleLogout} from "@/api/axios";
import {UserResponse} from "@/store/user/types";
import {getCurrentUser} from "@/store/user/actions/thunks/current-user";

export type AuthState = {
  user: UserResponse | null,
  isLoading: boolean,
}

const initialState: AuthState = {
  user: null,
  isLoading: false,
};

const authSlice = createSlice({
  name: "user",
  initialState,
  reducers: {
    SET_USER: (state, action: PayloadAction<AuthState["user"]>) => {
      state.user = action.payload;
```

```
  },
  LOG_OUT: (state) => {
    handleLogout();
    state.user = null;
  },
},
extraReducers: (builder) => {
  builder
    .addCase(login.fulfilled, (state, action) => {
      state.isLoading = false;
      state.user = action.payload.user;
    })
    .addCase(login.pending, (state) => {
      state.isLoading = true;
    })
    .addCase(login.rejected, (state) => {
      state.isLoading = false;
    })

    .addCase(register.fulfilled, (state, action) => {
      state.isLoading = false;
      state.user = action.payload.user;
    })
    .addCase(register.pending, (state) => {
      state.isLoading = true;
    })
    .addCase(register.rejected, (state) => {
      state.isLoading = false;
    })

    .addCase(getCurrentUser.fulfilled, (state) => {
      state.isLoading = false;
    });
```

```
  },
});

export default authSlice.reducer;
export const authActions = authSlice.actions;

import {bindActionCreators, combineReducers, configureStore} from "@reduxjs/toolkit";
import authReducer from "../store/user";
import chatReducer from "../store/chat";
import friendsReducer, {friendsActions} from "../store/friends";
import {TypedUseSelectorHook, useDispatch, useSelector} from "react-redux";
import thunk from "redux-thunk";
import {authActions} from "@/store/user";
import {chatActions} from "@/store/chat";
import {authMiddleware} from "@/store/middlewares/auth.middleware";
import {actionLogMiddleware} from "@/store/middlewares/action-log.middleware";
import {invitationsApi} from "@/code/features/invitations/services/invitations.service";
import {CombinedState, Dispatch, MiddlewareAPI} from "redux";
import {roomSlice} from "@/store/room";

const rootReducer = combineReducers({
  auth: authReducer,
  friends: friendsReducer,
  chat: chatReducer,
  room: roomSlice.reducer,
  [invitationsApi.reducerPath]: invitationsApi.reducer
});

const middlewares = [
  thunk,
  authMiddleware,
  invitationsApi.middleware,
];
```

```
if (process.env.NODE_ENV === “development’) { middlewares.push(actionLogMiddleware); }

export const store = configureStore({
  reducer: rootReducer,
  middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(...middlewares)
});

export type RootState = ReturnType<typeof rootReducer>
export type AppStore = typeof store;
export type AppDispatch = AppStore[“dispatch’]

export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector;

export const ALL_ACTIONS = {
  ...authActions,
  ...friendsActions,
  ...chatActions,
  ...roomSlice.actions,
};
export const useActions = () => {
  return bindActionCreators(ALL_ACTIONS, useAppDispatch());
};
export type AppMiddlewareAPI = MiddlewareAPI<Dispatch, CombinedState<RootState>>;

import {Middleware} from "redux";
import {RootState} from "@/store/store";
import toast from “react-hot-toast’;

export const actionLogMiddleware: Middleware<{}, RootState> = ({getState}) => next => (action) =>
{
   console.log(“Dispatching action:’, action, “\n - Next state:’, getState());
   if (action.type.includes(“rejected’)) {
```

```
        toast.error(action.payload?.error ?? action.payload?.data.error, {
            duration: 4000,
        })
    }
    if (action.type.includes(“fulfilled’) && action.payload?.notification) {
        toast.success(action.payload?.notification, {
            duration: 3000,
        })
    }
    return next(action);
};

export enum SocketEvent {
    UpdateFriendInvitations = “updateFriendInvitations’,
    UpdateFriends = “updateFriends’,
    UpdateOnlineUsers = “updateOnlineUsers’,
    GetDirectChatHistory = “getDirectChatHistory’,
    SendNewDirectMessage = “sendNewDirectMessage’,
    UpdateDirectChatHistory = “updateDirectChatHistory’,
    UserTyping = “userTypingMessage’,
    LeaveRoom = “leaveRoom’,
    CreateRoom = “createRoom’,
    UpdateActiveRooms = “updateActiveRooms’,
    JoinRoom = “joinRoom’,
    RTCSignal = “RTCSignal’,
    PrepareRTC = “prepareRTC’,
    InitRTC = “initRTC’,
    RTCInitResponse = “RTCInitResponse’,
    RTCSignalFulfilled = “RTCSignalFulfilled’,
    NotifyFriendRoomCreated = “notifyFriendRoomCreated’,
}

import {PaletteMode} from "@mui/material";
```

```
import {grey} from "@mui/material/colors";
import {ThemeOptions} from "@mui/material/styles/createTheme";

export default (mode: PaletteMode): ThemeOptions => ({
  typography: {
    fontFamily: “"Product Sans", sans-serif’,
  },
  palette: {
    mode,
    contrastThreshold: 3,
    tonalOffset: 0.2,
    ...(mode === “light’
      ? {
        primary: {
          main: “#4755e0’,
        },
        secondary: {
          main: “#5C76B7’
        },
        error: {
          main: “#F40B27’
        },

      }
      : {
        background: {
          default: grey[900],
          paper: “#333’,
        },
        primary: {
          main: “#4755e0’,
        },
        secondary: {
```

```
            main: “#03dcff’
          },
          success: {
            main: “#fff’,
            dark: “#57d249’,
          },
          error: {
            main: “#ff5b5b’
          },
          text: {
            primary: “#fff’,
            secondary: grey[500],
          },
        }),
  },
});

export function extractLocaleAndPathname(url: string): { locale: string | null; pathnameWithoutLocale:
string } {
 const match = url.match(/^\/([a-z]{2,4})(\/.*)?$/);
 if (match) {
  const locale = match[1];
  const pathnameWithoutLocale = match[2] || “/’;
  return { locale, pathnameWithoutLocale };
 }
 return { locale: null, pathnameWithoutLocale: url };
}

export function capitalize(string: string): string {
 return string.charAt(0).toUpperCase() + string.slice(1);
}

“use client’;
```

```
import {FC, MouseEventHandler, PropsWithChildren, ReactNode, useState} from “react’;
import {IconButton, Menu} from "@mui/material";

type Props = {
  menuIcon: ReactNode
};

const MenuDropdown: FC<PropsWithChildren<Props>> = ({children, menuIcon }) => {
  const [anchorEl, setAnchorEl] = useState<HTMLButtonElement | null>(null);
  const isOpen = Boolean(anchorEl);

  const handleMenuOpen: MouseEventHandler = (event) => {
    setAnchorEl(event.currentTarget as HTMLButtonElement);
  };
  const handleMenuClose = () => {
    setAnchorEl(null);
  };

  return (
     <div>
       <IconButton onClick={handleMenuOpen} style={{color: "white"}}>
         {menuIcon}
       </IconButton>

       <Menu
         id="basic-menu"
         anchorEl={anchorEl}
         open={isOpen}
         onClose={handleMenuClose}
       >
         {children}
       </Menu>
```

```
      </div>
  );
};

export default MenuDropdown;

“use client’

import {PropsWithChildren, useEffect} from “react’;
import {createTheme, ThemeProvider} from “@mui/material’;
import {useAppDispatch} from “@/store/store’;
import {CookieKey} from “@/shared/models/cookie.model’;
import designTokens from “@/shared/utils/design-tokens’;
import { getCookie } from “cookies-next/client’;
import {getCurrentUser} from “@/store/user/actions/thunks/current-user’;

type Props = {};

const MainLayout = ({children}: PropsWithChildren<Props>) => {
  const dispatch = useAppDispatch();
  const theme = createTheme(designTokens(“dark’));

  useEffect(() => {
    const token = getCookie(CookieKey.Token)
    token && dispatch(getCurrentUser());
  }, []);


  return (
    <div style={{minHeight: “100vh’}}>
      <ThemeProvider theme={theme}>
        {children}
      </ThemeProvider>
```

```
      </div>
  );
};

export default MainLayout;

import React, {FC} from “react’
import GroupsIcon from "@mui/icons-material/Groups";
import AddIcon from "@mui/icons-material/Add";
import {useActions, useAppSelector} from "@/store/store";
import SidebarButton from "@/shared/ui/SidebarButton";
import {CHAT_TYPES} from "@/store/chat/types";
import ActiveRoomButton from “@/code/features/rtc/components/buttons/ActiveRoomButton’;

type Props = {};

const SideBar: FC<Props> = ({}) => {
  const {SET_CHAT_DETAILS, CREATE_ROOM} = useActions();
  const {activeRooms, user: {isInRoom}} = useAppSelector(state => state.room)

  function handleGroupsClicked() {
    SET_CHAT_DETAILS({
      type: CHAT_TYPES.NONE,
      friend: null,
    });
  }

  function handleNewRoomClicked() {
    CREATE_ROOM({
      isRoomCreator: true,
      isInRoom: true,
    });
  }
```

```
  return (
    <div className="w-[72px] px-3 py-2 flex gap-2 flex-col items-center bg-[#202225]">
      <SidebarButton onClick={handleGroupsClicked}>
        <GroupsIcon/>
      </SidebarButton>
      <SidebarButton disabled={isInRoom} onClick={handleNewRoomClicked}>
        <AddIcon/>
      </SidebarButton>
      {activeRooms.map(room => (
        <ActiveRoomButton key={room.roomID} room={room}/>
      ))}
    </div>
  );
};

export default SideBar;

"use client";

import {Locale} from "@/i18n/request";
import {usePathname, useRouter} from "@/i18n/routing";
import {useLocale} from "next-intl";
import {useParams} from "next/navigation";
import * as React from "react";
import {useState} from "react";
import {MenuItem} from "@mui/material";
import TranslateIcon from "@mui/icons-material/Translate";
import MenuDropdown from "@/shared/ui/MenuDropdown";

const LanguageSwitch = () => {
  const router = useRouter();
  const locale = useLocale();
```

```
  const pathname = usePathname();
  const params = useParams();
  const [currentLocale, setCurrentLocale] = useState(locale);

  const changeLanguage = (newLocale: Locale) => {
    if (newLocale === locale) return;
    setCurrentLocale(newLocale);

    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    router.replace({pathname, params}, {locale: newLocale});
  };


  return (
     <MenuDropdown menuIcon={<TranslateIcon/>}>
       <MenuItem selected={currentLocale === Locale.English} onClick={() =>
changeLanguage(Locale.English)}>
         English
       </MenuItem>
       <MenuItem selected={currentLocale === Locale.Ukrainian} onClick={() =>
changeLanguage(Locale.Ukrainian)}>
         Українська
       </MenuItem>
       <MenuItem selected={currentLocale === Locale.Deutsch} onClick={() =>
changeLanguage(Locale.Deutsch)}>
         Deutsch
       </MenuItem>
       <MenuItem selected={currentLocale === Locale.Spanisch} onClick={() =>
changeLanguage(Locale.Spanisch)}>
         Español
       </MenuItem>
     </MenuDropdown>
```

```
  );
};

export default LanguageSwitch;

import {DEFAULT_LOCALE} from "@/i18n/request';
import {getCookie, setCookie} from "cookies-next';
import {CookieKey} from "@/shared/models/cookie.model';
import {NextApiRequest} from "next';
import {NextResponse} from "next/server';

export async function GET(req: NextApiRequest) {
  const {searchParams} = new URL(req.url!);

  const code = searchParams.get("code');
  const lang = (await getCookie(CookieKey.Locale)) || DEFAULT_LOCALE;

  if (!code) {
    return NextResponse.redirect(`http://localhost:3000/${lang}/auth-status`, {
      status: 401,
    });
  }

  try {
    const res = NextResponse.redirect(`http://localhost:3000/${lang}/auth-status`, {
      status: 301,
    });

    res.cookies.set(CookieKey.GoogleCode, code, {
      path: "/',
      maxAge: 60
    })
    setCookie(CookieKey.GoogleCode, code, {
```

```
        path: “/’,
        maxAge: 60
      });

      return res;
    } catch (error: unknown) {
      console.log(error, “error’);
      return new Response((error as Error)?.message ?? ““, {
        status: 500,
      });
    }
}

“use client’;

import {FC,} from “react’;
import Messager from “@/code/features/chat/components/messager’;
import SideBar from “@/shared/ui/layout/SideBar’;
import NavBar from “@/shared/ui/layout/NavBar’;
import FriendsSidebar from “@/code/features/friends/ui/friends-sidebar’;
import {useAppSelector} from “@/store/store’;
import Room from “@/code/features/rtc/components/Room’;

type Props = {};

const DashboardPage: FC<Props> = ({}) => {
  const {user: {isInRoom}} = useAppSelector(state => state.room);

  return (
     <div className="w-full min-h-screen flex">
       <SideBar/>
       <FriendsSidebar/>
       <div className="flex flex-col w-full">
```

```
            <NavBar/>
            <Messager/>
          </div>
          {isInRoom && <Room/>}
        </div>
    );
};

export default DashboardPage;
```

```
“use client’;

import Input from “@/shared/ui/Input’;
import PageSizedForm from “@/shared/ui/PageSizedForm’;
import FormHeader from “@/shared/ui/FormHeader’;
import PrimaryButton from “@/shared/ui/PrimaryButton’;
import Redirect from “@/code/features/auth/components/redirect’;
import {useAppDispatch, useAppSelector} from “@/store/store’;
import HeroImage from “@/shared/ui/HeroImage’;
import {SCHEMAS} from “@/validation/schemas’;
import {useForm} from “react-hook-form’;
import {yupResolver} from “@hookform/resolvers/yup’;
import {useRouter} from “@/i18n/routing’;
import {useTranslations} from “next-intl’;
import {Button} from “@mui/material’;
import {IoLogoGoogle} from “react-icons/io’;
import {generateGoogleAuthUrl} from “@/shared/utils/auth’;
import {login} from “@/store/user/actions/thunks/login’;

const LoginPage = ({}) => {
  const router = useRouter();
  const dispatch = useAppDispatch();
  const t = useTranslations();
```

```
  const tLogin = useTranslations(“Pages.Login’);
  const {isLoading} = useAppSelector(state => state.auth);


  const {control, handleSubmit} = useForm({
    defaultValues: {
      email: ““,
      password: ““,
    },
    resolver: yupResolver(SCHEMAS(t).login)
  });

  const onSubmit = handleSubmit(async data => {
    try {
      await dispatch(login(data));
      router.push(`/dashboard`);
    } catch (error) {
    }
  });

  return (
     <div className="relative">
       <PageSizedForm>
         <FormHeader title={tLogin(“title’)} description={tLogin(“description’)}/>
         <form onSubmit={onSubmit} className="flex-col mb-4 flex gap-2 mt-3">
           <Input placeholder="example@gmail.com" name="email" label={t(“Common.email’)}
control={control}/>
           <Input name="password" label={t(“Common.password’)} type="password"
control={control}/>
           <PrimaryButton className="mt-2" disabled={isLoading} type="submit">
             {tLogin(“submit’)}
           </PrimaryButton>
           <Button sx={{textTransform: “none’, fontSize: 16}}
```

```
                color="secondary" variant="outlined"
                onClick={() => {
                  const authWindow = window.open(generateGoogleAuthUrl(), "_self");
                  if (authWindow) authWindow.onbeforeunload = () => {
                    console.log("closed");
                  };
                }}>
              <IoLogoGoogle className="mr-2"/>
              Google
            </Button>
          </form>
          <Redirect href="/register" description={tLogin("needAnAccount")}>
            {tLogin("createOne")}
          </Redirect>
        </PageSizedForm>
        <HeroImage outerWrapperClassName="bottom-0 left-0" innerWrapperClassName="lg:h-[60vh]
h-[75vh]" imageProps={{
          src: "https://resume-io-helpscout.s3.amazonaws.com/1644317987059/assets/hero-image.svg",
          alt: "bg"
        }}/>
      </div>
  );
};

export default LoginPage;



"use client";

import {FC} from "react";
import PageSizedForm from "@/shared/ui/PageSizedForm";
import FormHeader from "@/shared/ui/FormHeader";
import PrimaryButton from "@/shared/ui/PrimaryButton";
```

```
import Redirect from “@/code/features/auth/components/redirect’;
import Input from “@/shared/ui/Input’;
import { useAppDispatch, useAppSelector} from “@/store/store’;
import HeroImage from “@/shared/ui/HeroImage’;
import {RegisterForm} from “@/shared/types/auth’;
import {useForm} from “react-hook-form’;
import {SCHEMAS} from “@/validation/schemas’;
import {yupResolver} from “@hookform/resolvers/yup’;
import {useRouter} from “@/i18n/routing’;
import {useTranslations} from “next-intl’;
import {register} from “@/store/user/actions/thunks/register’;

type Props = {};

const RegisterPage: FC<Props> = ({}) => {
  const dispatch = useAppDispatch();
  const router = useRouter();
  const tRegister = useTranslations(“Pages.Register’);
  const t = useTranslations(““);
  const {isLoading} = useAppSelector(state => state.auth);

  const {control, handleSubmit} = useForm<RegisterForm>({
    resolver: yupResolver(SCHEMAS(t).register),
    defaultValues: {
      username: ““,
      email: ““,
      password: ““,
    },
  });

  const onSubmit = handleSubmit(async data => {
    try {
      await dispatch(register(data)).unwrap();
```

```
        router.push(“/dashboard’);
      } catch (error) {

      }
    });

  return (
      <div className="relative">
        <PageSizedForm>
          <FormHeader title={tRegister(“title’)} description={tRegister(“description’)}/>
          <form onSubmit={onSubmit} className="flex-col mb-4 flex gap-2 mt-3">
            <Input name="username" label={t(“Common.username’)} control={control}/>
            <Input placeholder="example@gmail.com" name="email" label={t(“Common.email’)}
control={control}/>
            <Input name="password" label={t(“Common.password’)} type="password"
control={control}/>
            <PrimaryButton disabled={isLoading} className="mt-2" type="submit">
              {tRegister(“submit’)}
            </PrimaryButton>
          </form>
          <Redirect href="/login" description={tRegister(“alreadyHaveAccount’)}>
            {tRegister(“logIn’)}
          </Redirect>
        </PageSizedForm>
        <HeroImage outerWrapperClassName="bottom-0 left-0" innerWrapperClassName="lg:h-[60vh]
h-[75vh]" imageProps={{
          src: “https://resume-io-helpscout.s3.amazonaws.com/1644317987059/assets/hero-image.svg’,
          className: “scale-x-[-1]’,
          alt: “bg’
        }}/>
      </div>
  );
};
```

```
export default RegisterPage;

"use client";

import { useEffect, useState } from "react";
import { Avatar, CircularProgress } from "@mui/material";
import { useAppSelector } from "@/store/store";
import {useTranslations} from “next-intl’;
import toast from “react-hot-toast’;

export default function ProfilePage() {
  const { user, isLoading } = useAppSelector((state) => state.auth);
  const t = useTranslations();

  if (isLoading) {
    return (
      <div className="flex justify-center items-center h-screen bg-gray-900">
        <CircularProgress />
      </div>
    );
  }

  if (!user) return null;

  const handleCopyID = () => {
    navigator.clipboard.writeText(user.userID).then(() => {
      toast.success(t(“Notification.Success.copyID’));
    });
  };

  return (
    <div className="min-h-screen bg-gray-800 text-white flex flex-col items-center p-8">
```

```
        <div className="bg-gray-700 p-6 rounded-lg shadow-lg w-full max-w-md">
          <div className="flex flex-col items-center">
            <Avatar
              src={user?.avatar}
              sx={{ width: 80, height: 80, bgcolor: "#5865F2" }}
              className="mb-4"
            >
              {user.username.charAt(0).toUpperCase()}
            </Avatar>

            <h1 className="text-2xl font-semibold">{t(“Common.username’)}: {user.username}</h1>
            <p className="text-gray-400 text-sm">{user.email}</p>
          </div>

          <div className="mt-6 bg-gray-600 p-2 rounded-lg flex items-center justify-between">
            <span className="text-gray-300 text-sm">ID: {user.userID}</span>
            <button
              onClick={handleCopyID}
              className="text-blue-400 hover:text-blue-300 text-sm"
            >
              Copy
            </button>
          </div>
        </div>
      </div>
    );
}

“use client’;

import {useEffect, useState} from “react’;
import {useRouter} from “@/i18n/routing’;
import toast from “react-hot-toast’;
```

```
import {connectSocketServer} from “@/api/socket.io/connection’;
import {store, useActions} from “@/store/store’;
import PrimaryButton from “@/shared/ui/PrimaryButton’;
import {useTranslations} from “next-intl’;
import {getCookie} from “cookies-next/client’;
import {CookieKey} from “@/shared/models/cookie.model’;
import {UserResponse} from “@/store/user/types’;
import axios from “axios’;
import {setCookie} from “cookies-next’;

const AuthStatusPage = () => {
  const router = useRouter();
  const [googleCode, setGoogleCode] = useState<null | string>(null);
  const t = useTranslations();
  const {SET_USER} = useActions();

  useEffect(() => {
    const code = getCookie(CookieKey.GoogleCode) ?? ““;
    setGoogleCode(code);
    (async function () {
      if (!code) {
        router.push(“/login’);
        return toast.error(“Google authentication failed. Try again...’);
      }
      const {data} = await axios.post(`http://localhost:5000/api/auth/google`, {
        code,
      });

      if (!data) throw new Error((data?.message ?? ““) as string);

      const {token, user} = data as { token: string; user: UserResponse };
      if (!user) {
        router.push(“/login’);
```

```
            return toast.error("Error occurred during authentication');
          } else {
            SET_USER(user);
            setCookie(CookieKey.Token, token);
            connectSocketServer({jwtToken: token, state: store});
            await Notification.requestPermission();
            router.push("/dashboard');
          }
        })();
      }, []);

      if (googleCode === null) return <div>Loading...</div>;
      else if (!googleCode) {
        return (
          <div className="text-red-700 grid gap-5 p-5">
            Authentication failed. Try again...
            <PrimaryButton onClick={() =>
router.push("/login')}>{t("Pages.Register.logIn')}</PrimaryButton>
          </div>
        );
      } else {
        return <div>Authenticating...</div>;
      }
};

export default AuthStatusPage;
```

ДОДАТОК Г
(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПЛАТФОРМИ ДЛЯ ОНЛАЙН-КОМУНІКАЦІЙ

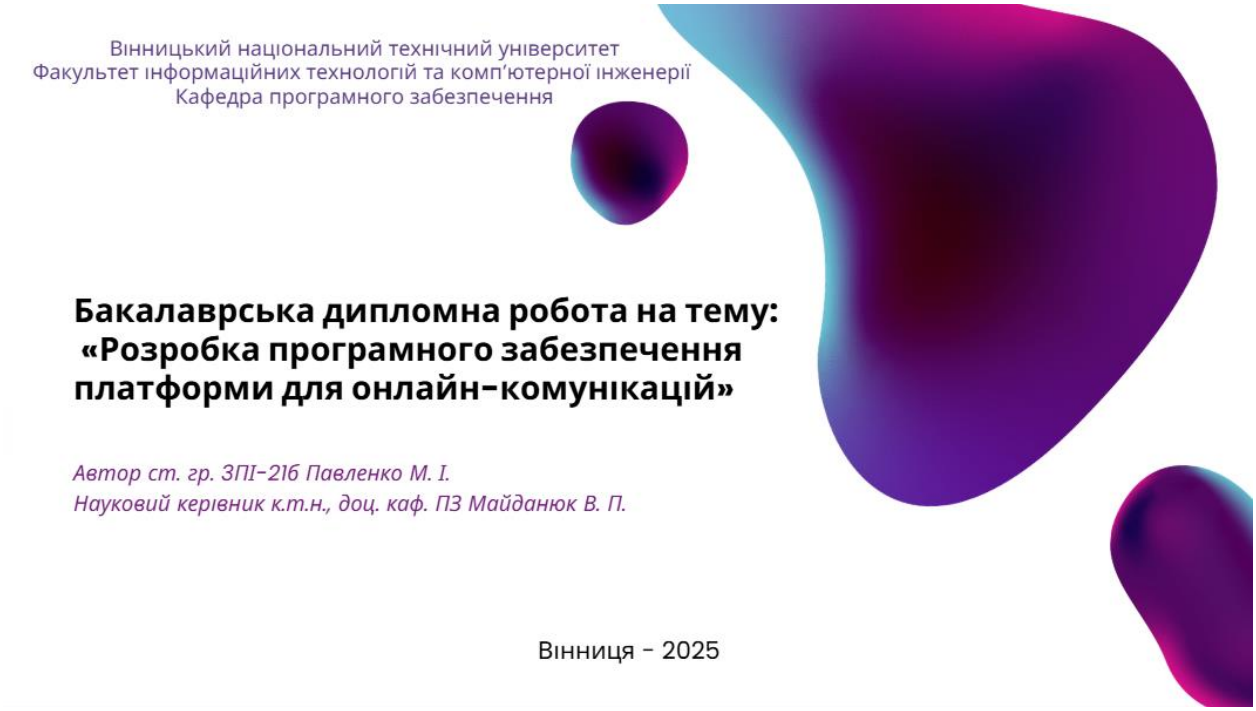

Рисунок Г.1 – Титульний слайд

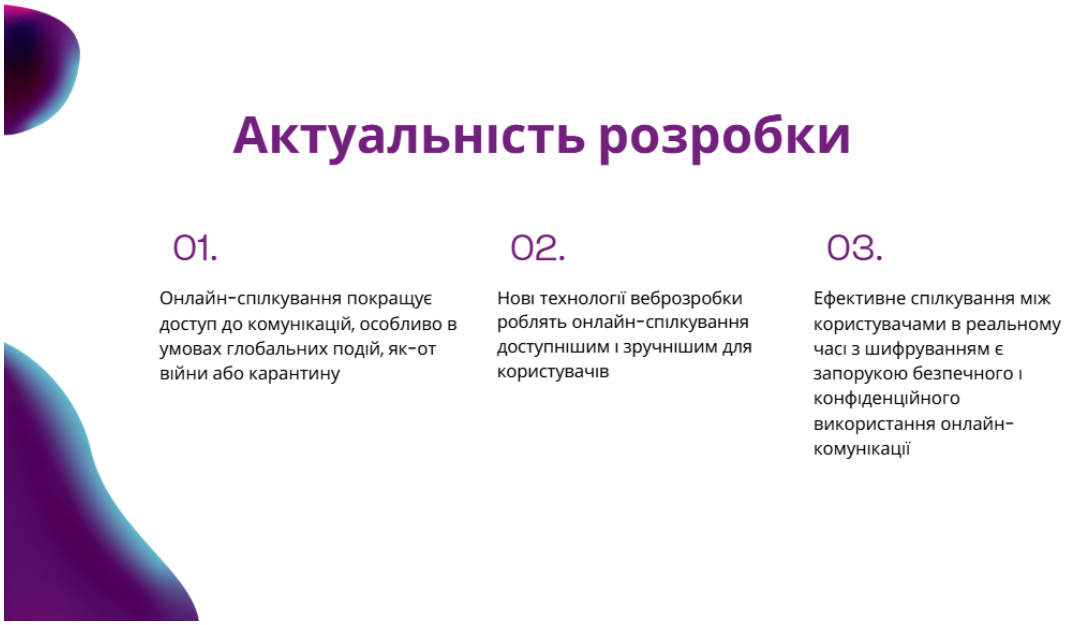

Рисунок Г.2 – Актуальність розробки

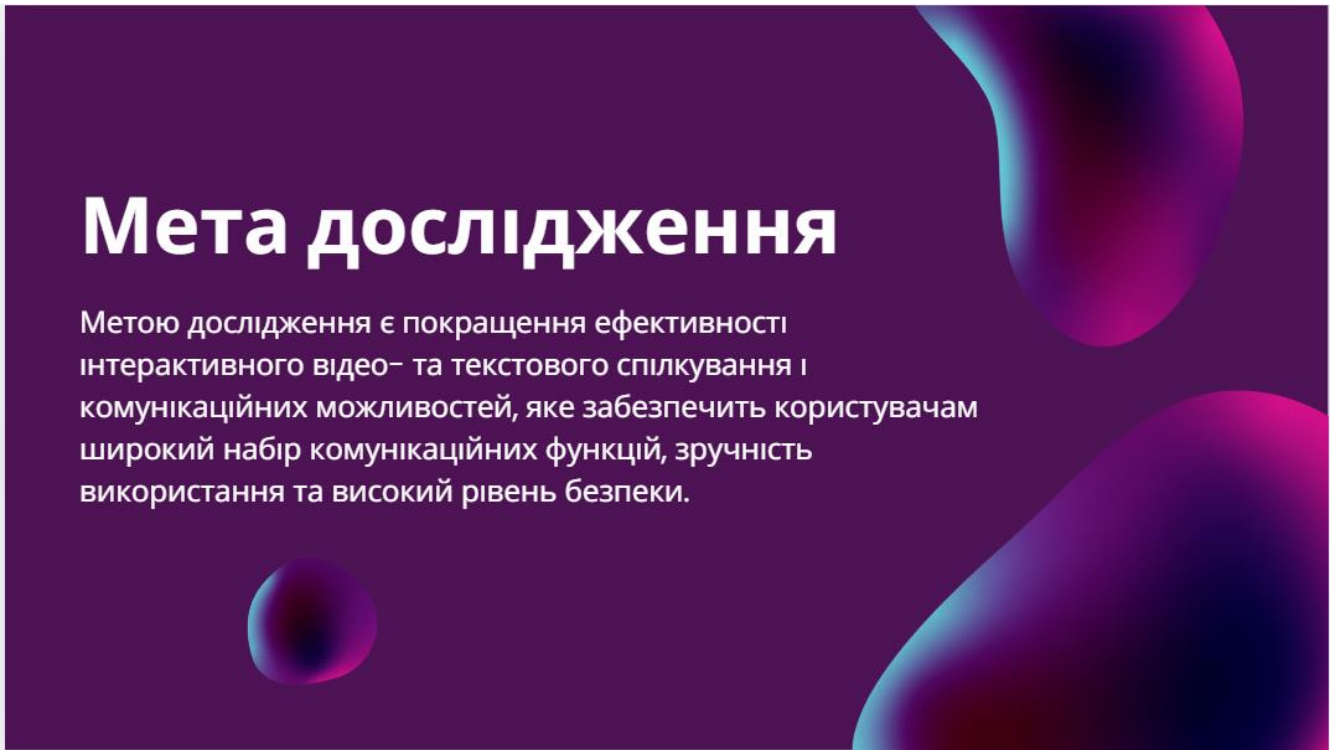

Рисунок Г.3 – Мета дослідження

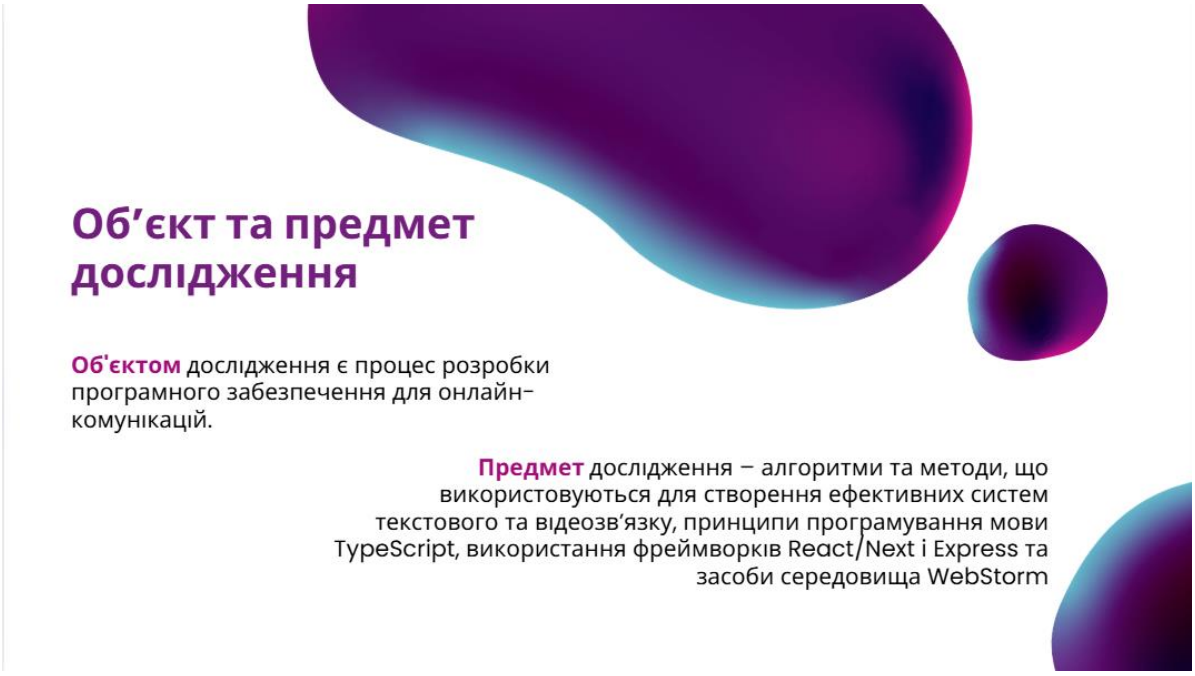

Рисунок Г.4 – Об’єкт та предмет дослідження

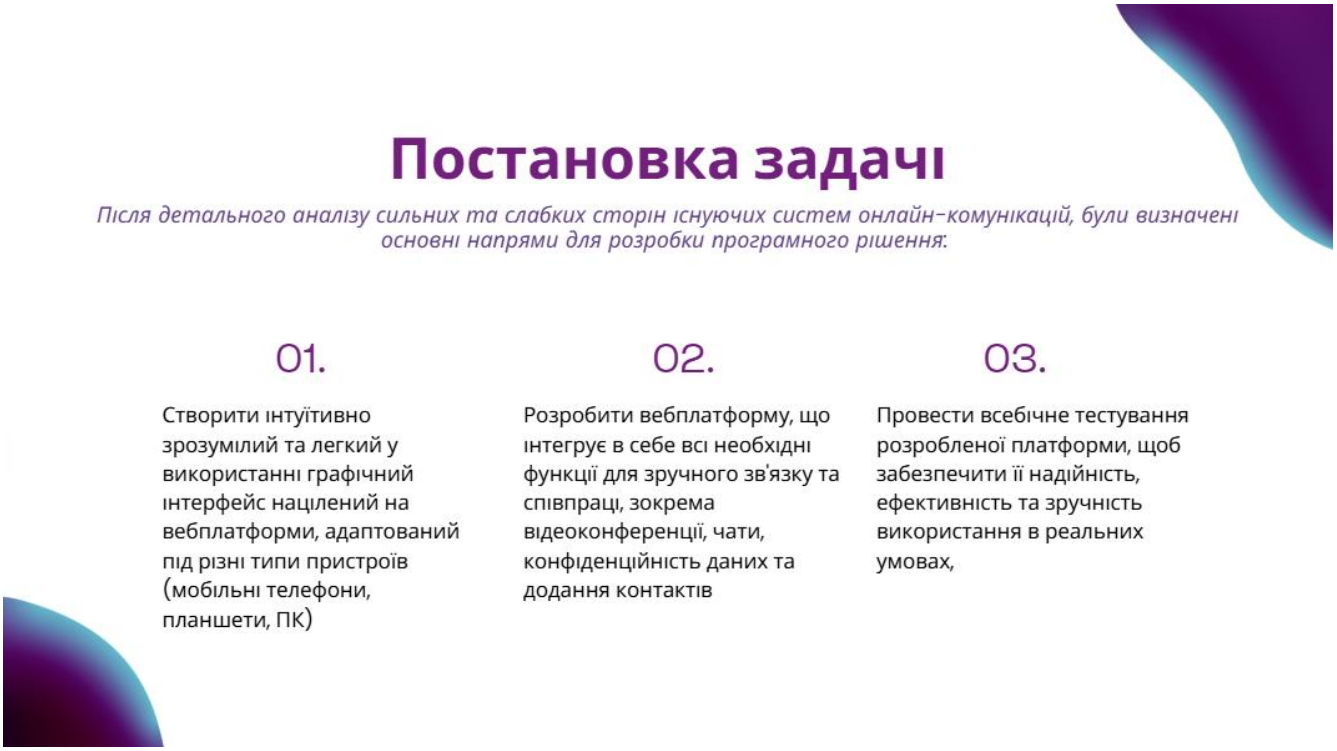

Рисунок Г.5 – Постановка задачі

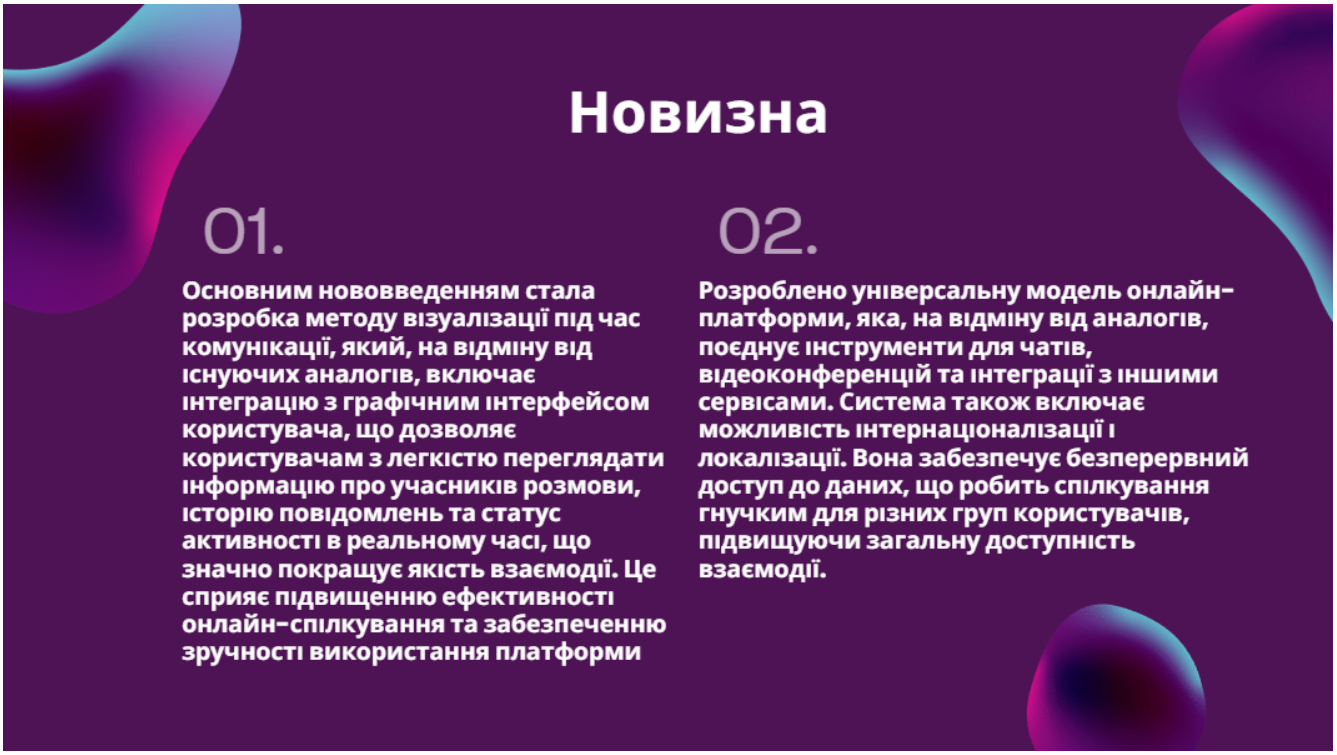

Рисунок Г.6 – Новизна розробленого застосунку

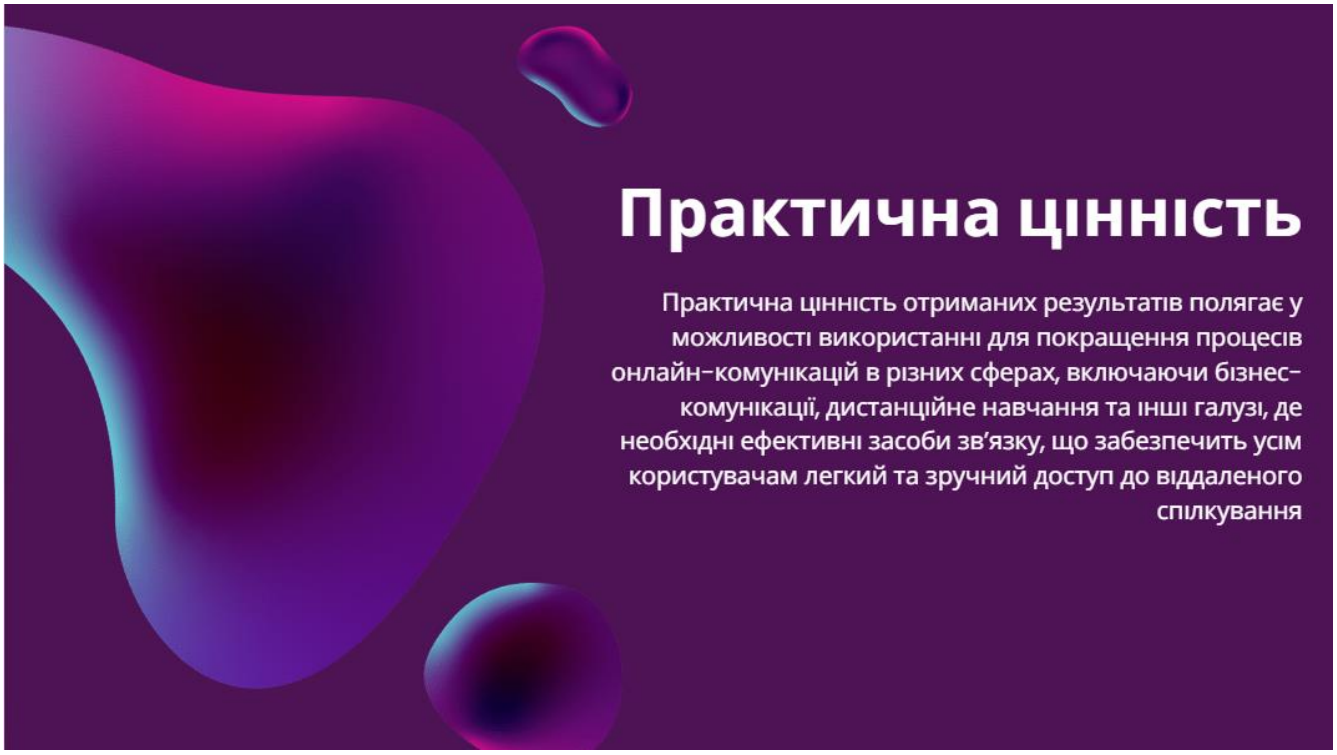

Рисунок Г.7 – Практична цінність

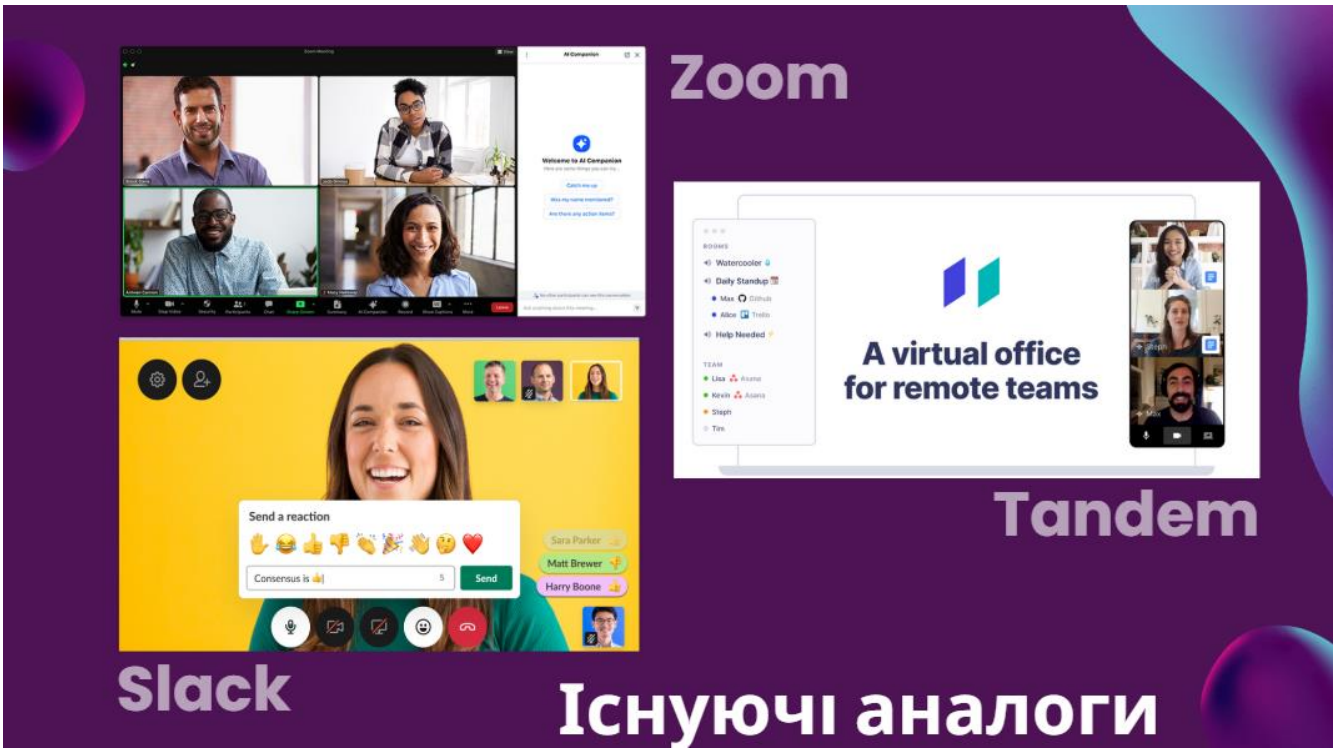

Рисунок Г.8 – Існуючі аналоги

Аналіз аналогів

Порівняльна характеристика	Zoom	Tandem	Slack	Власна розробка
Аутентифікація	1	1	1	1
Оновлення інформації в реальному часі	1	1	0	1
Підтримка технологій ШІ	1	0	1	1
Інтернаціоналізація	1	0	0	1
Інтеграція зі сторонніми сервісами	0	1	1	1
Сумарний коефіцієнт	**4**	**4**	**3**	**5**

Рисунок Г.9 – Аналіз аналогів

Модель роботи системи

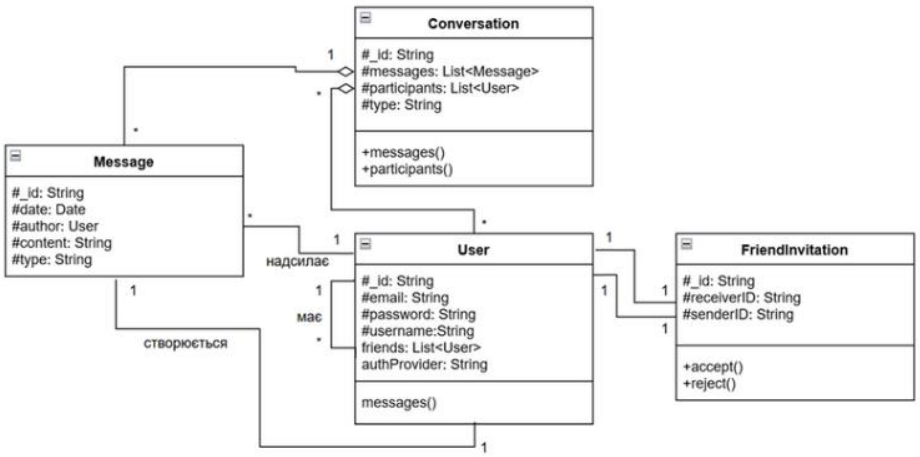

Рисунок Г.10 – Модель роботи системи

Рисунок Г.11 – Загальний алгоритм роботи системи

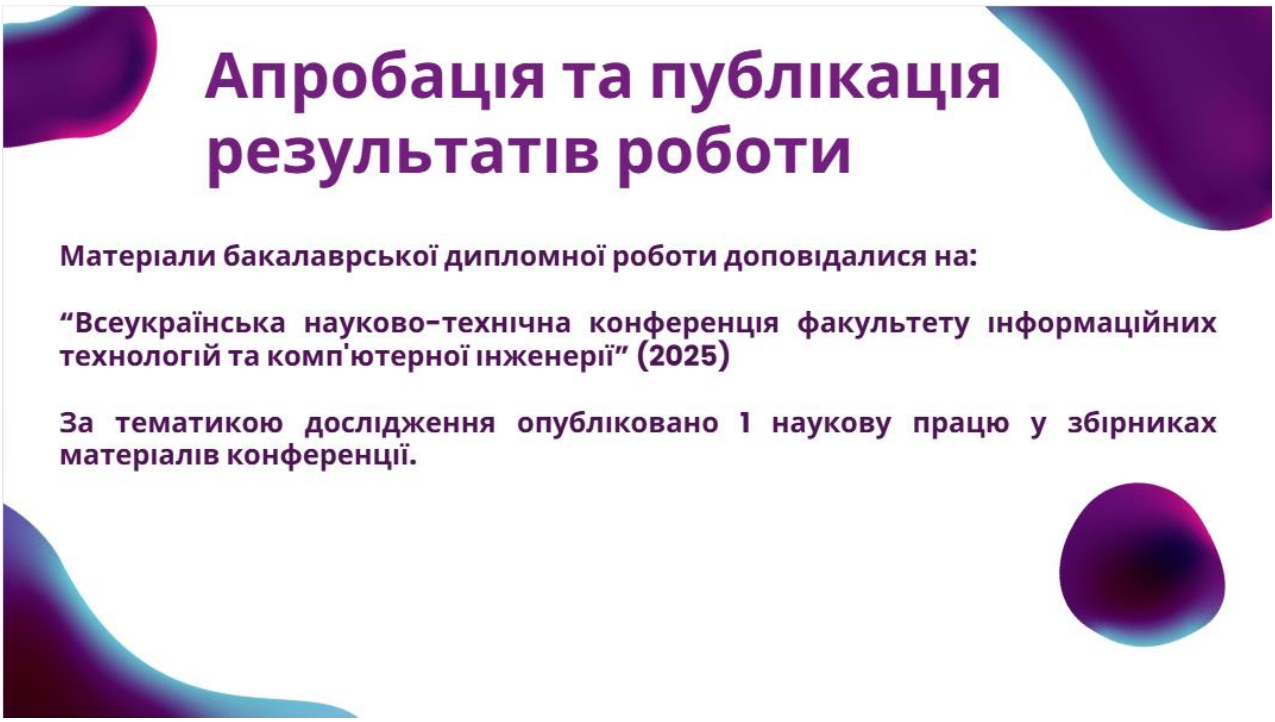

Рисунок Г.12 – Апробація та публікація результатів роботи

Рисунок Г.13 – Висновки

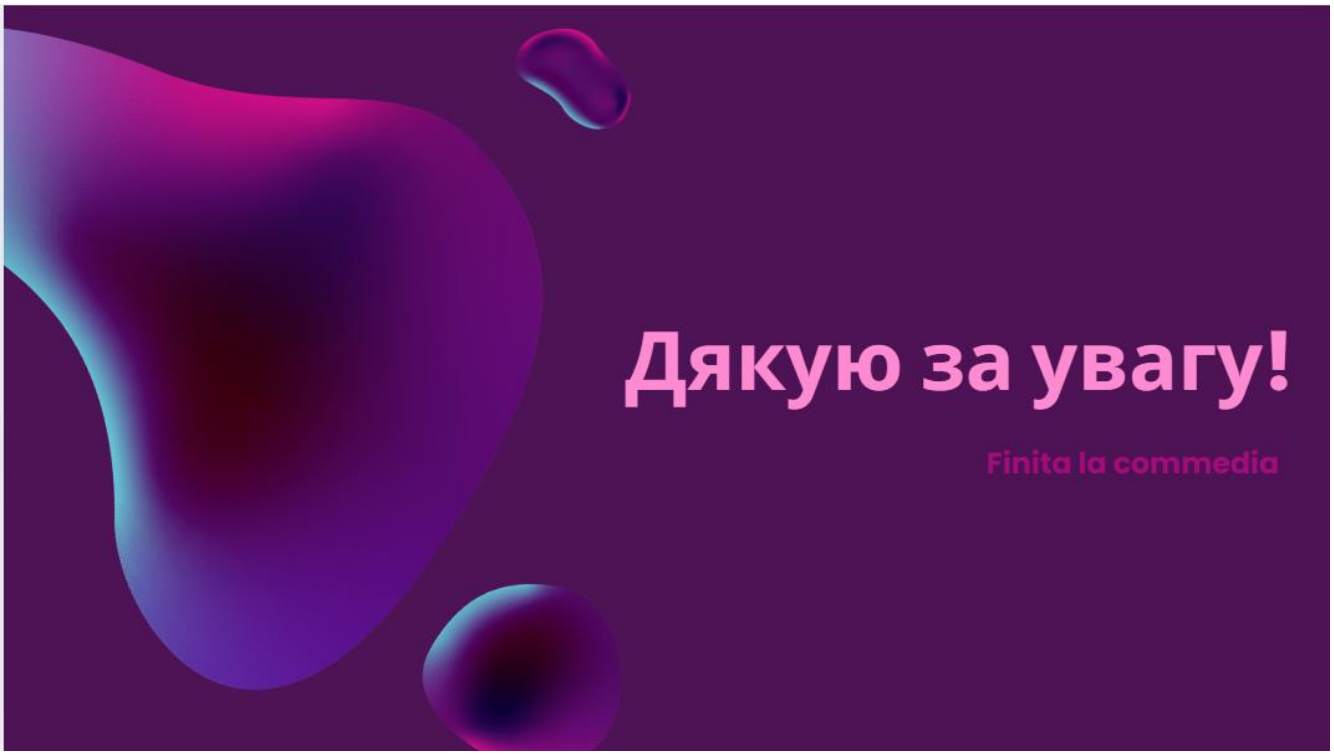

Рисунок Г.14 – Фінальний слайд