Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська дипломна робота на тему:

«РОЗРОБКА АВТОМАТИЗОВАНОГО МОДУЛЯ ОБМІНУ ПОВІДОМЛЕННЯМИ І ФАЙЛАМИ ЕЛЕКТРОННОЇ ОСВІТНЬОЇ СИСТЕМИ НА ОСНОВІ ПЛАТФОРМИ FIREBASE»

Виконав: студент IV курсу, групи ЗАКІТ-20б
спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)
Поліщук Я. В..
(прізвище та ініціали)
Керівник: професор кафедри АІІТ
Паламарчук Є.А.
(прізвище та ініціали)
«11» 06 2024 р.
Рецензент: к.т.н., доцент кафедри ПЗ
Коваленко О. О.
(прізвище та ініціали)
«12» 06 2024 р.

Допущено до захисту
завідувач кафедри АІІТ
Дідкало О.В.
«13» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 15 – Автоматизація та приладобудування
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
д.т.н., проф. Бісікало О. В.
«13» 03 2024 року

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

Поліщуку Ярославу Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка автоматизованого модуля обміну повідомленнями і файлами електронної освітньої системи на основі платформи Firebase»
керівник роботи Паламарчук Євген Анатолійович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «11» березня 2024 року №80
2. Термін подання студентом роботи 10.06.2024 р.
3. Вихідні дані до роботи:
   - Можливість обміну текстовими повідомленнями між студентами та викладачами.
   - Підтримка відправки та отримання файлів у межах повідомлень.
   - Динамічне оновлення повідомлень у режимі реального часу.
   - Інтеграція з університетською системою JetIQ для синхронізації даних користувачів та забезпечення автентифікації.
4. Зміст текстової частини (перелік питань, які потрібно розробити) провести огляд готових рішень та існуючих засобів розробки, визначити структуру додатку, розробити алгоритм роботи та програмне забезпечення для модуля обміну інформацією.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) Uml діаграма, Use-case діаграма, Er діаграма.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	зав пр
1-4	Паламарчук Є.А., професор АІІТ		

7. Дата видачі завдання 12.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Термін виконання		Пр
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БДР	03.10.2023	09.10.2023	
2	Аналіз предметної області та опрацювання літературних джерел.	12.03.2024	29.03.2024	
3	Постановка задач для вирішення в БДР	01.04.2024	26.04.2024	
4	Вирішення поставлених задач	29.04.2024	10.05.2024	
5	Підтвердження достовірності отриманих результатів	13.05.2024	24.05.2024	
7	Оформлення пояснювальної записки та графічної частини	27.05.2024	31.05.2024	
8	Перевірка ПЗ на відповідність вимогам	03.06.2024	07.06.2024	
9	Попередній захист, доопрацювання, та рецензування БДР	10.06.2024	15.06.2024	
10	Захист БДР ЕК	17.06.2024	17.06.2024	

Студент _____ Поліщук Я. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Паламарчук Є.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 99 сторінок формату А4 в тому числі і додатків, на яких є 3 схеми, список використаних джерел містить 24 найменування.

У цій бакалаврській роботі представлено розробку автоматизованого модуля обміну повідомленнями та файлами для електронної освітньої системи на основі платформи Firebase. Основною метою роботи є створення зручного та ефективного інструменту для комунікації між студентами та викладачами в рамках навчального процесу. Проєкт реалізовано за допомогою фреймворку Flutter, який забезпечує швидкий та зручний процес розробки кросплатформенних додатків. Firebase використовується як бекенд для зберігання даних та забезпечення реального часу обміну повідомленнями та файлами. У роботі розглядаються основні аспекти архітектури програми, включаючи взаємодію з базою даних, авторизацію користувачів та забезпечення безпеки даних.

Ключові слова: Firebase, Flutter, Dart, автоматизований модуль, web, модуль, обмін повідомленнями, обмін файлами, JetIQ, BLoC, Firestore, інтеграція.

ANNOTATION

The bachelor's thesis consists of 99 pages of A4 format, including appendices, which contain 3 diagrams. The list of references includes 24 titles.

This bachelor's thesis presents the development of an automated module for messaging and file exchange for an electronic educational system based on the Firebase platform. The main goal of the work is to create a convenient and efficient tool for communication between students and teachers within the educational process. The project is implemented using the Flutter framework, which ensures a fast and convenient process for developing cross-platform applications. Firebase is used as the backend for data storage and to provide real-time messaging and file exchange. The work examines the main aspects of the application's architecture, including interaction with the database, user authentication, and data security.

Keywords: Firebase, Flutter, Dart, automated module, web, module, messaging, file sharing, JetIQ, BLoC, Firestore, integration.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасні освітні процеси вимагають постійного розвитку та впровадження нових технологій для підвищення ефективності навчання та комунікації між учасниками освітнього процесу. Однією з актуальних проблем є забезпечення зручного та швидкого обміну інформацією між студентами та викладачами. В цьому контексті, розробка автоматизованих модулів для обміну повідомленнями та файлами є важливим кроком у вдосконаленні освітньої системи.

Мета і задачі роботи. розробка автоматизованого модуля для обміну повідомленнями та файлами, що забезпечить ефективну комунікацію між учасниками навчального процесу у Вінницькому національному технічному університеті (ВНТУ).

Розробка даного модуля спрямована на створення універсального інструменту, який дозволить студентам та викладачам оперативно обмінюватися інформацією та навчальними матеріалами, а також полегшить взаємодію з іншими елементами освітньої системи через інтеграцію з платформою JetIQ.

Для досягнення мети необхідко виконати наступні задачі:

1. Аналіз існуючих рішень для обміну повідомленнями та файлами в освітніх системах.
2. Визначення вимог до програмного модуля, враховуючи специфіку навчального процесу у ВНТУ.
3. Розробка архітектури модуля з урахуванням інтеграції з платформою JetIQ.
4. Реалізація модуля на основі Flutter та Firebase.
5. Тестування та оцінка ефективності роботи розробленого модуля.

Об'єкт дослідження - процес обміну повідомленнями та файлами в рамках електронної освітньої системи.

Предмет дослідження: програмний модуль обміну повідомленнями та файлами, розроблений на основі платформи Firebase з використанням фреймворку Flutter, з інтеграцією в університетську систему JetIQ.

Практична цінність роботи полягає у створенні автоматизованого модуля для обміну повідомленнями та файлами в електронній освітній системі, що базується на платформі Firebase. Цей модуль покращує комунікацію між учасниками освітнього процесу у Вінницькому національному технічному університеті, забезпечуючи зручний і ефективний спосіб обміну інформацією.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Розробка програмних рішень для підтримки освітніх процесів вимагає глибокого розуміння предметної області, до якої належить даний проєкт. Дослідження предметної області є важливим етапом у створенні ефективного та зручного інструменту для комунікації між учасниками навчального процесу. Основною метою цього розділу є аналіз існуючих технологій та рішень, які використовуються для забезпечення обміну інформацією в навчальних закладах, а також виявлення їх сильних та слабких сторін. Це дозволить визначити вимоги до розроблюваного модуля та обґрунтувати вибір технологій і підходів, які будуть використані при його створенні.

1.1 Обмін повідомленнями в JetIQ

У нашому університеті вже функціонує система для обміну повідомленнями між студентами та викладачами рис. 1.1. Ця система дозволяє студентам надсилати викладачам одиничні повідомлення з прикріпленими файлами до 64 Мб. Основні функціональні можливості включають вибір викладача зі списку, вибір типу роботи, опис файлу, вибір файлу для надсилання та перегляд історії відправлених файлів, яка містить деталі про дату відправлення, опис файлу, викладача, назву файлу з можливістю завантаження, розмір файлу та статус відправлення.

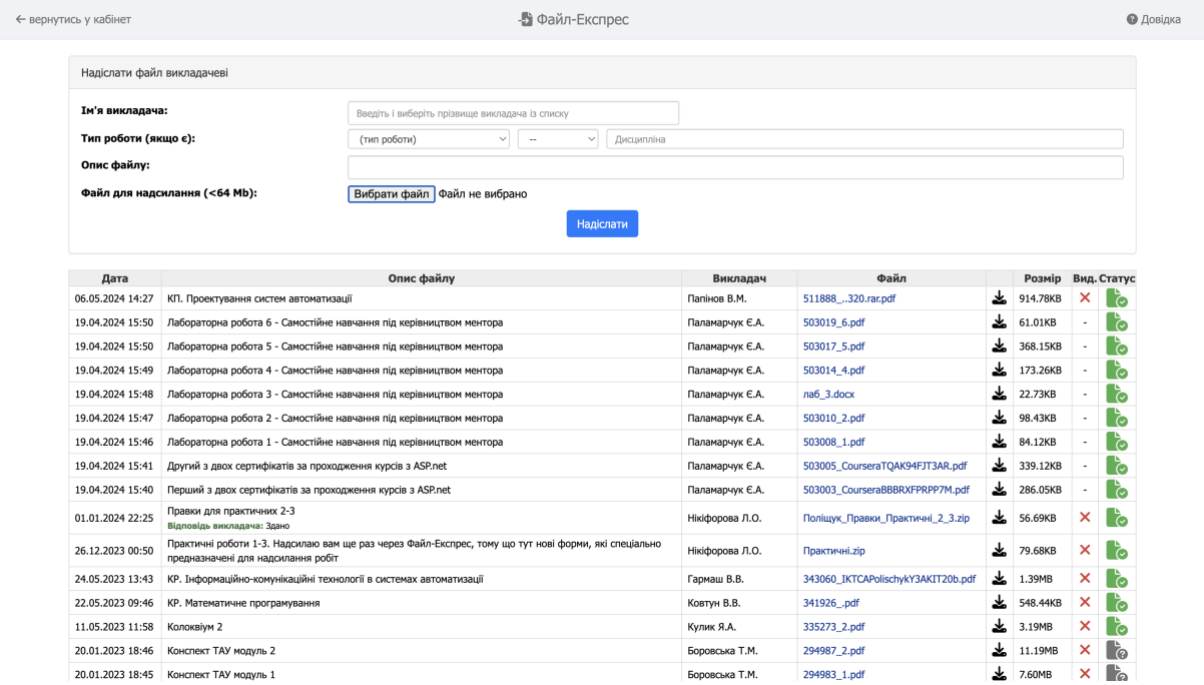

Рисунок 1.1 – Інтерфейс системи обміну повідомленнями в системі JetIQ

Система має певні переваги. Вона відрізняється простим і зрозумілим інтерфейсом, що забезпечує зручність використання. Можливість вибору типу роботи і опису файлу спрощує ідентифікацію відправлених матеріалів. Також історія відправлень надає користувачам доступ до всієї інформації про відправлені файли, включаючи можливість їх повторного завантаження.

Однак, ця система має й недоліки. Основним обмеженням є обмежений функціонал обміну повідомленнями. Відсутність підтримки динамічного оновлення повідомлень, це унеможливлює спілкування між студентами та викладачами у реальному часі. Крім того, система не підтримує надсилання декількох файлів одночасно, а максимальний розмір файлу обмежений до 64 Мб. Інтерфейс, хоча і є функціональним, може бути більш інтерактивним і сучасним, а також не підтримує додаткові функції, такі як групові чати та сповіщення про нові повідомлення.

Таким чином, аналіз існуючої системи обміну повідомленнями у ВНТУ показує, що вона виконує свої основні функції, проте має ряд обмежень, які знижують її ефективність у порівнянні з сучасними месенджерами. Для покращення комунікації у навчальному процесі доцільно розглянути можливість розширення функціоналу системи, додавання динамічного оновлення повідомлень, підтримки діалогів на більше ніж одне повідомлення та одну відповідь та групових чатів.

1.2 Аналіз стану реалізації комунікацій у корпоративних системах

Розробка системи обміну повідомленнями для освітніх закладів є важливим завданням, яке вже вирішували багато існуючих платформ. Розглянемо деякі з найбільш популярних та функціональних аналогів, які можуть бути використані для порівняння та врахування кращих практик при створенні нової системи.

1.2.1 Google Classroom

Google Classroom (рис. 1.2) є однією з найпопулярніших платформ для управління навчальним процесом. Вона пропонує інтегровану систему обміну повідомленнями, яка дозволяє викладачам та студентам обмінюватися повідомленнями, завданнями та файлами. Основними перевагами Google Classroom є інтеграція з іншими сервісами Google (Google Drive, Google Docs, Google Calendar), підтримка реального часу сповіщень та можливість створення групових дискусій [1]. Недоліками є залежність від інтернет-з'єднання та обмежені можливості кастомізації для специфічних потреб окремих навчальних закладів.

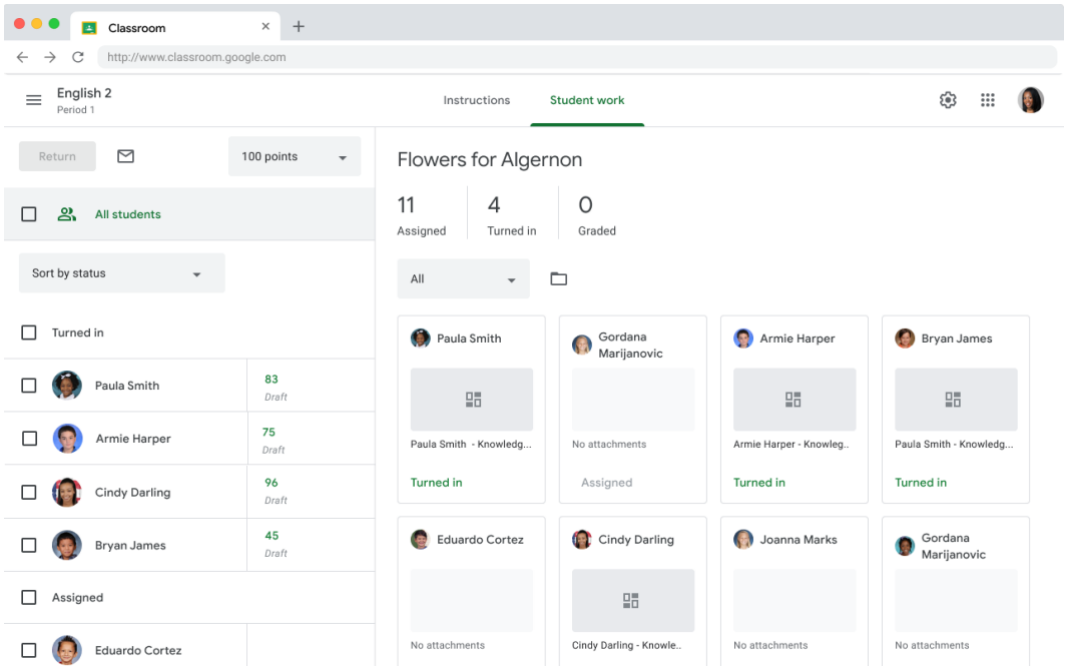

Рисунок 1.2 – Інтерфейс Google Classroom

1.2.2 Microsoft Teams

Microsoft Teams є частиною екосистеми Microsoft 365 і пропонує розширені можливості для комунікації та співпраці (рис. 1.3). Платформа підтримує чати, відеоконференції, спільне редагування документів та інтеграцію з іншими сервісами Microsoft (OneDrive, SharePoint, Outlook). Microsoft Teams дозволяє створювати канали для окремих груп або курсів (рис. 1.3), що сприяє організованій та ефективній комунікації [2]. Недоліки включають складність налаштування для нових користувачів та вимоги до потужних ресурсів для стабільної роботи.

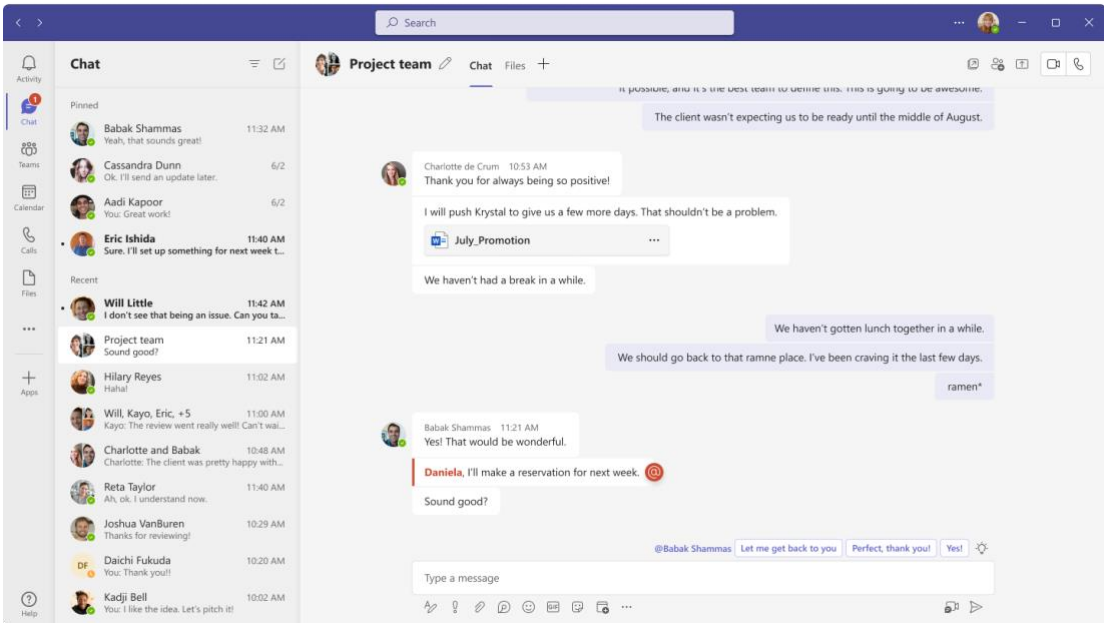

Рисунок 1.3 – Microsoft Teams

1.2.3 Moodle

Moodle є відкритою платформою для управління навчальним процесом, яка широко використовується в багатьох освітніх закладах. Вона підтримує обмін повідомленнями, завантаження та обмін файлами, створення форумів та групових дискусій. Основною перевагою Moodle є її гнучкість та можливість кастомізації під специфічні потреби навчального закладу. Проте, інтерфейс платформи може здаватися застарілим і складним для нових користувачів, а також вимагає значних зусиль для налаштування та підтримки.

1.2.4 Slack

Slack є популярним інструментом для командної співпраці, який також може бути адаптований для використання в освітніх закладах. Платформа підтримує обмін повідомленнями в реальному часі, створення

каналів для окремих груп або курсів, інтеграцію з іншими інструментами (Google Drive, Dropbox, Zoom)[3] та потужні можливості для пошуку повідомлень і файлів (рис. 1.4). Однак, використання Slack може бути обмежене через відсутність специфічних освітніх функцій та високу вартість для великих груп користувачів.

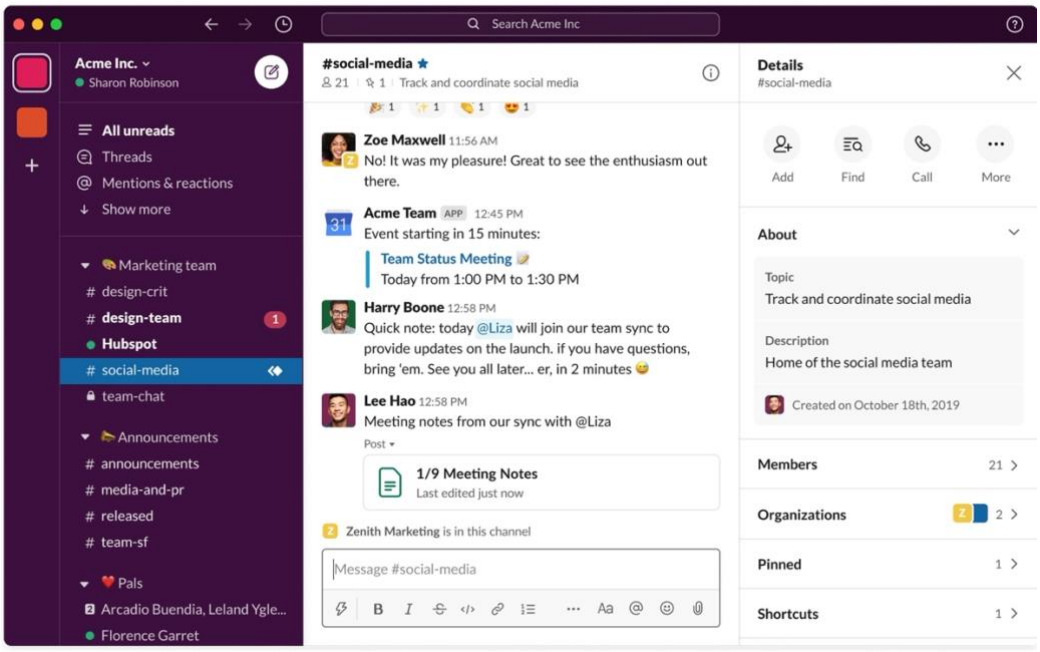

Рисунок 1.4 – Інтерфейс Slack

1.2.5 Discord

Discord, спочатку розроблений для геймерів, набув популярності як інструмент для спільноти та навчання [4]. Він підтримує текстові та голосові чати, обмін файлами, створення каналів та груп (рис. 1.4). Основними перевагами Discord є простота використання, підтримка реального часу комунікації та безкоштовний доступ для базового функціоналу. Проте, як і у випадку зі Slack, Discord може не мати всіх

необхідних освітніх функцій та потребує налаштування для використання в академічному середовищі.

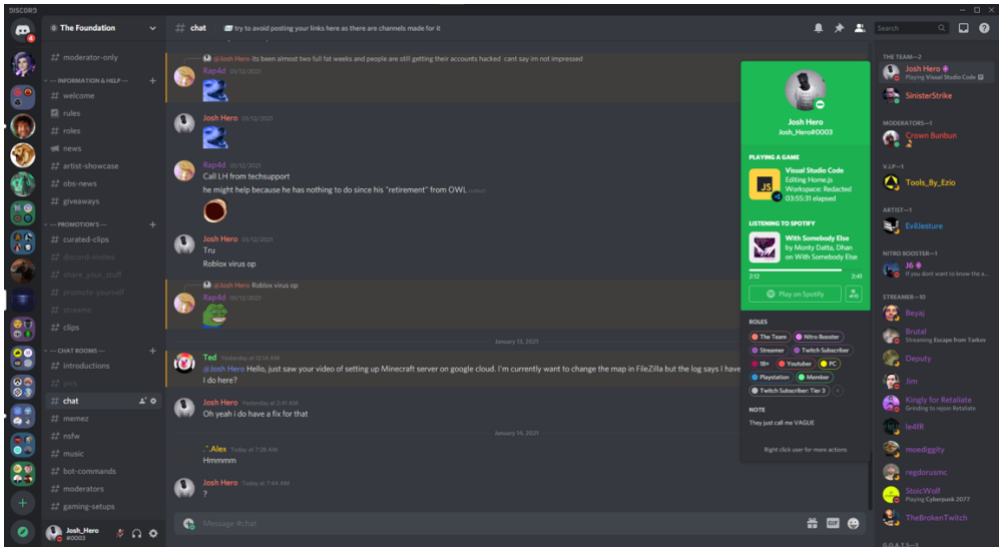

Рисунок 1.4 – Інтерфейс Discord

1.3 Підсумок першого розділу

Порівнюючи ці платформи, можна зробити висновок, що кожна з них має свої унікальні переваги та недоліки. Google Classroom і Microsoft Teams забезпечують інтеграцію з популярними офісними сервісами, тоді як Moodle пропонує високу гнучкість та кастомізацію. Slack і Discord відзначаються простотою використання та підтримкою реального часу комунікації, але можуть не відповідати всім потребам освітніх установ. Підсумок порівняння розглянутих готових рішень наведено в таблиці 1.1.

Таблиця 1.1 - Порівняльна таблиця готових рішень

Платформа	**Основні функції**	**Переваги**	**Недоліки**
Google Classroom	- Управління курсами - Обмін файлами - Оцінювання студентів - Інтеграція з Google Drive	- Інтуїтивно зрозумілий інтерфейс - Інтеграція з іншими продуктами Google - Безкоштовний	- Обмежений функціонал для великих навчальних закладів - Залежність від Google екосистеми
Microsoft Teams	- Чати та відеоконференції - Управління завданнями - Спільне редагування документів	- Інтеграція з Microsoft Office 365 - Підтримка великих команд - Можливість запису зустрічей	- Високі вимоги до ресурсів - Платний для великих організацій
Moodle	- Курси та модулі - Оцінювання та аналітика - Форум та чати - Підтримка SCORM	- Гнучкість налаштувань - Відкрите програмне забезпечення - Велика кількість плагінів	- Вимагає технічної підтримки для налаштування та адміністрування - Складний інтерфейс

Slack	- Чати та канали - Обмін файлами - Інтеграція з іншими сервісами - Пошук повідомлень	- Зручність використання - Велика кількість інтеграцій - Підтримка командної роботи	- Обмеження на кількість повідомлень у безкоштовній версії - Відсутність відеоконференцій
Discord	- Голосові та відеочати - Текстові канали - Обмін файлами - Стрімінг екрана	- Безкоштовний - Інтуїтивно зрозумілий інтерфейс - Підтримка великої кількості учасників	- Спочатку орієнтований на геймерів, тому не всі функції підходять для освітніх цілей - Реклама

Врахування цих факторів при розробці власної системи обміну повідомленнями дозволить створити ефективний інструмент, який відповідатиме специфічним вимогам ВНТУ.

2 ВИБІР ТЕХНОЛОГІЧНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ПРОЄКТУ

2.1 Обґрунтування вибору цільової платформи розробки

Браузер є основною платформою для розробки багатьох сучасних веб-додатків завдяки своїй доступності, кросплатформеності та підтримці сучасних веб-технологій. Для розробки системи обміну повідомленнями для навчального процесу у Вінницькому національному технічному університеті (ВНТУ) браузер є найбільш підходящою платформою з кількох важливих причин. Основна система JetIQ також працює в браузері, що забезпечує зручну інтеграцію та єдине середовище для користувачів.

2.1.1 Аналіз застосування браузерних рішень

1. Кросплатформеність

   Плюси: Веб-додатки можуть працювати на будь-якому пристрої, що має браузер (ПК, ноутбуки, планшети, смартфони). Це забезпечує широкий доступ до додатку незалежно від операційної системи.

   Мінуси: Можливі відмінності у відображенні та функціонуванні додатку на різних браузерах, які потрібно враховувати та тестувати.
2. Доступність та простота використання

   Плюси: Користувачі можуть легко отримати доступ до додатку, просто відкривши веб-сторінку. Не потрібно встановлювати додаткове програмне забезпечення.

   Мінуси: Залежність від інтернет-з'єднання, хоча прогресивні веб-додатки (PWA) можуть частково працювати офлайн [5].
3. Інтеграція з існуючою системою JetIQ

Плюси: Легка інтеграція з JetIQ, оскільки обидві системи працюють у браузері. Це спрощує обмін даними та синхронізацію між системами.

Мінуси: Обмеження функціональності можуть бути зумовлені можливостями API системи JetIQ.

4. Сучасні веб-технології

Плюси: Використання сучасних веб-технологій (HTML5, CSS3, JavaScript, WebRTC) дозволяє створювати багатофункціональні, інтерактивні та ефективні веб-додатки.

Мінуси: Необхідність постійного оновлення знань про нові технології та стандарти.

2.1.2 Аналіз застосування мобільних додатків як цільової платформи

1. Мобільні додатки (iOS та Android)

Плюси:

- Оптимізація під мобільні пристрої.
- Підтримка нативних функцій (сповіщення, доступ до апаратних можливостей).

Мінуси:

- Необхідність розробки та підтримки двох окремих версій (для iOS та Android).
- Високі витрати на розробку та підтримку.
- Обмежена інтеграція з веб-системою JetIQ.

2. Настільні додатки

Плюси:

- Більш гнучке управління ресурсами та продуктивністю.

- Можливість використання нативних функцій операційної системи.

Мінуси:

- Обмежена кросплатформеність (потрібна розробка для різних ОС).
- Необхідність встановлення програмного забезпечення на кожен пристрій.
- Відсутність мобільності у використанні.

2.1.3 Вибрана платформа

Враховуючи попередньо проведений аналіз платформ, складемо порівняльну таблицю для них (таблиця 2.1).

Таблиця 2.1 - Порівняння можливих цільових платформ для розробки додатку

Платформа	**Основні функції**	**Переваги**	**Недоліки**
Браузер	- Доступ з будь-якого пристрою з браузером - Автоматичне оновлення - Легка розробка	- Не вимагає встановлення - Кросплатформенність - Простота підтримки та оновлення - Інтеграція з JetIQ	- Залежність від інтернет-з'єднання - Обмежена продуктивність
Мобільні додатки	- Доступ до апаратних функцій пристрою	- Висока продуктивність - Зручність використання	- Необхідність розробки окремо для різних

	- Пуш-повідомлення - Офлайн-режим	- Підтримка офлайн-режиму	платформ (iOS, Android) - Вимога встановлення
Настільні додатки	- Висока продуктивність - Доступ до ресурсів комп'ютера - Підтримка складних завдань	- Висока продуктивність - Зручність для професійного використання - Можливість роботи офлайн	- Залежність від конкретної операційної системи - Необхідність встановлення та оновлення

Враховуючи кросплатформеність, доступність, простоту інтеграції з системою JetIQ, а також можливості сучасних веб-технологій, браузер є оптимальною платформою для розробки системи обміну повідомленнями для ВНТУ. Інші платформи, такі як мобільні або настільні додатки, мають свої переваги, але вони також несуть значні витрати та складнощі у розробці та підтримці. Вибір браузера дозволить створити ефективну, доступну та зручну для користувачів систему, яка відповідатиме потребам навчального процесу.

2.2 Вибір мови програмування та фреймворку для користувацької частини

2.2.1 JavaScript та його фреймворки

JavaScript є однією з найпопулярніших мов програмування для веб-розробки [6], і для розширення його можливостей було створено кілька

фреймворків. Тут ми розглянемо переваги та недоліки трьох основних JavaScript фреймворків: React.js, Vue.js та Angular.

1. React.js

Переваги:

- Віртуальний DOM: React використовує віртуальний DOM, що покращує продуктивність, мінімізуючи кількість прямих маніпуляцій з DOM. Це призводить до швидшого оновлення та відтворення інтерфейсу користувача. [7]
- Компонентна Архітектура: React сприяє використанню багаторазових компонентів, що покращує підтримуваність та повторне використання коду. Компоненти можуть бути об'єднані для створення складних інтерфейсів користувача.
- Велика Екосистема та Спільнота: React має велику екосистему бібліотек, інструментів та сильну спільноту. Це забезпечує розробникам багато ресурсів, плагінів та підтримки.
- SEO-дружність: React може бути відтворений на сервері за допомогою таких інструментів, як Next.js, що покращує SEO веб-додатків.
- Гнучкість: React забезпечує гнучкість у виборі бібліотек та інструментів для управління станом, маршрутизації та інших функцій. Розробники не прив'язані до певного набору інструментів.

Недоліки:

- Крива Навчання: Екосистема React може бути надто складною для початківців через необхідність вивчення додаткових бібліотек та інструментів (наприклад, Redux для управління станом, React Router для маршрутизації).

- Шаблонний Код: React часто вимагає більше шаблонного коду порівняно з іншими фреймворками, особливо при інтеграції з іншими бібліотеками та управлінні станом.
- Швидкі Зміни: Екосистема React швидко розвивається, що може бути складним для розробників, які намагаються встигнути за новими оновленнями та найкращими практиками.

2. Vue.js

Переваги:

- Простота та Легкість у Вивченні: Vue.js має простий та інтуїтивно зрозумілий синтаксис, що робить його легким для вивчення та початку роботи. Документація добре організована та легко зрозуміла.
- Гнучкість: Vue забезпечує гнучкість у структурі додатків. Розробники можуть використовувати його як бібліотеку для покращення конкретних частин додатку або як повноцінний фреймворк.
- Реактивність: Система реактивності Vue проста та зрозуміла, що дозволяє розробникам створювати динамічні та відзивні інтерфейси користувача [8].
- Компонентна Архітектура: Як і React, Vue сприяє використанню багаторазових компонентів, що покращує підтримуваність та повторне використання коду.
- Однофайлові Компоненти: Vue дозволяє розробникам писати HTML, CSS та JavaScript в одному файлі, що покращує організацію та читабельність коду.

Недоліки:

- Менша Екосистема: Хоча екосистема Vue зростає, вона все ще менша порівняно з React та Angular. Це означає менше бібліотек та інструментів.
- Ринок Праці: Існує менше можливостей для роботи розробникам Vue порівняно з React та Angular, оскільки він менш широко використовується в індустрії.
- Спільнота та Підтримка: Vue має меншу спільноту порівняно з React та Angular, що може призвести до меншої підтримки та менше ресурсів для вирішення проблем.

3. Angular

Переваги:

- Комплексний фреймворк: Angular є повноцінним фреймворком, який забезпечує комплексне рішення для створення масштабних додатків. Він включає інструменти для маршрутизації, управління станом, обробки форм тощо.
- Двостороння зв'язка даних: двостороння зв'язка даних Angular забезпечує синхронізацію виду та моделі, що спрощує процес розробки.
- Впровадження Залежностей: Вбудоване впровадження залежностей Angular покращує модульність та тестування коду, полегшуючи управління залежностями.
- TypeScript: Angular побудований на TypeScript, що забезпечує статичну типізацію та покращує якість, читабельність та підтримуваність коду.
- Сильна спільнота та підтримка від корпорацій: Angular має сильну підтримку від Google та велику спільноту, що забезпечує постійні оновлення, поліпшення та велику кількість ресурсів.

Недоліки:

- Складність: Angular має стрімку криву навчання через свою складність та обширний набір функцій. Це може бути складним для початківців.
- Продуктивність: продуктивність Angular може бути повільнішою порівняно з React та Vue, особливо у додатках з великою кількістю динамічних елементів.
- Складний синтаксис: синтаксис Angular може бути громіздким та складним, що призводить до більшої кількості шаблонного коду та потенційно зменшує швидкість розробки.
- Часті оновлення: Angular зазнає частих оновлень, що може бути складним для підтримки та може вимагати значного перероблення коду.

2.2.2 Python та його фреймворки

1. Django

Переваги:

- Повноцінний фреймворк з усім необхідним для розробки веб-додатків.
- Добра підтримка безпеки та аутентифікації.
- Висока швидкість розробки завдяки вбудованим модулям.

Недоліки:

- Менша продуктивність у порівнянні з JavaScript фреймворками.
- Може бути занадто важким для простих проєктів [9].

2. Flask

Переваги:

- Легкий та гнучкий фреймворк.
- Добре підходить для малих та середніх проєктів.
- Простий у вивченні та використанні.

Недоліки:

- Менша кількість вбудованих функцій у порівнянні з Django.
- Потребує більше налаштувань та конфігурацій.

2.2.3 Ruby та його фреймворки

Ruby on Rails

Переваги:

- Висока швидкість розробки завдяки великій кількості генераторів та скриптів.
- Конвенційний підхід, який сприяє дотриманню стандартів [10].
- Велика спільнота та багато доступних бібліотек.

Недоліки:

- Менша продуктивність у порівнянні з JavaScript фреймворками.
- Менша популярність серед сучасних технологій.

2.2.4 Java та його фреймворки

Spring Boot

Переваги:

- Потужний та масштабований фреймворк.
- Добра підтримка корпоративних рішень.
- Підтримка різних видів баз даних та API.

Недоліки:

- Висока складність, потребує багато часу на навчання.

- Великий розмір додатків.

2.2.3 Dart та його фреймворки

Flutter

Переваги:

- Кросплатформеність: Flutter дозволяє створювати додатки, які працюють на різних платформах (веб, мобільні додатки для iOS та Android) з одним кодовою базою. Це значно зменшує час і ресурси, необхідні для розробки та підтримки додатків на різних платформах [11].
- Висока продуктивність: Dart компілюється у машинний код, що забезпечує високу продуктивність додатків. Flutter використовує власний рендеринг, що дозволяє створювати плавні анімації та швидкі інтерфейси.
- Єдиний код для UI та логіки: Flutter дозволяє писати UI безпосередньо у коді за допомогою віджетів, що спрощує процес розробки та усунення помилок. Відсутність потреби в окремих файлах XML для UI, як у деяких інших фреймворках, робить розробку більш ефективною.
- Швидкий цикл розробки: Функція Hot Reload у Flutter дозволяє швидко вносити зміни до коду і миттєво бачити результати, що прискорює процес розробки та тестування.
- Багата бібліотека віджетів: Flutter надає велику кількість вбудованих віджетів, які легко налаштовуються, що дозволяє створювати гарні та функціональні інтерфейси.
- Добра документація та активна спільнота: Flutter має добре структуровану документацію і активну спільноту, що

забезпечує доступ до численних навчальних матеріалів, прикладів та підтримки.

Недоліки:

- Великий розмір додатків: Додатки, розроблені на Flutter, можуть мати більший початковий розмір у порівнянні з додатками, написаними на нативних мовах. Це може бути критично для мобільних додатків, де розмір інсталяційного файлу має значення.
- Обмежена підтримка нативних функцій: Хоча Flutter постійно розширює свою підтримку нативних функцій, деякі специфічні нативні функції можуть бути важко реалізувати або вимагати написання додаткового нативного коду.
- Відносно нова технологія: Flutter є відносно новою технологією, тому існує менше готових бібліотек та плагінів у порівнянні з більш зрілими технологіями. Це може обмежувати можливості розробки в деяких випадках.
- Менше ресурсів для навчання та підтримки: Хоча Flutter має добру документацію, кількість ресурсів для навчання та підтримки може бути меншою у порівнянні з популярними фреймворками, такими як React або Angular.
- Не всі веб-функції підтримуються: На момент написання, не всі функції, які можна реалізувати в нативних веб-додатках, підтримуються у Flutter для веб. Це може створювати певні обмеження в розробці веб-версій додатків.

2.2.4 Вибір мови програмування і фремворку

Dart і Flutter пропонують значні переваги для розробки кросплатформених додатків, особливо у контексті цього проєкту, де

потрібна підтримка веб-інтерфейсу та можливість швидкого розроблення і тестування. Висока продуктивність, багата бібліотека віджетів і швидкий цикл розробки роблять Flutter привабливим вибором.

2.3 Вибір програмної платформи для серверної частини

Для розробки серверної частини проєкту можуть використовуватись різні платформи. Вибір платформи залежить від вимог до продуктивності, масштабованості, безпеки та зручності розробки. Нижче наведено декілька популярних платформ з їхніми перевагами та недоліками.

1. Node.js

Переваги:

- Асинхронна обробка запитів: Node.js використовує неблокуючу модель введення/виведення, що дозволяє ефективно обробляти велику кількість одночасних запитів.
- Велика спільнота та екосистема: Наявність великої кількості бібліотек та модулів через npm, що спрощує розробку.
- JavaScript на сервері: Використання однієї мови (JavaScript) на сервері та клієнті спрощує процес розробки.

Недоліки:

- Обмеження в обчислювальних задачах: Node.js не підходить для обчислювально-інтенсивних задач через однотредову природу.
- Відносно нова технологія: Хоча Node.js швидко розвивається, він все ще вважається відносно новим у порівнянні з більш зрілими платформами.

2. Django (Python)

Переваги:

- Повноцінний фреймворк: Django пропонує багато вбудованих функцій для розробки, таких як аутентифікація, адмін-панель, ORM, тощо.
- Швидкість розробки: Завдяки високому рівню абстракції та готовим модулям, розробка на Django може бути швидкою та ефективною.
- Безпека: Django має вбудовані механізми для захисту від поширених вразливостей.

Недоліки:

- Продуктивність: Django може бути менш продуктивним у порівнянні з деякими іншими платформами через свою високорівневу природу.
- Менша гнучкість: Велика кількість вбудованих функцій може обмежувати гнучкість налаштувань.

3. Spring Boot (Java)

Переваги:

- Масштабованість: Spring Boot підходить для розробки великих масштабованих додатків.
- Модульність: Добре організована архітектура дозволяє легко розширювати та підтримувати додаток.
- Безпека: Spring Boot пропонує потужні інструменти для забезпечення безпеки додатків.

Недоліки:

- Складність: Висока крива навчання через складну конфігурацію та велику кількість налаштувань.
- Великий обсяг пам'яті: Додатки на Spring Boot можуть споживати більше пам'яті у порівнянні з іншими платформами.

4. Express.js (Node.js)

Переваги:

- Легкість та швидкість: Express.js є легким фреймворком, який дозволяє швидко створювати веб-сервери.
- Гнучкість: Express.js надає мінімальну структуру, дозволяючи розробникам вибирати власні інструменти та бібліотеки.
- JavaScript на сервері: Використання JavaScript на сервері дозволяє використовувати одну мову для всього стеку.

Недоліки:

- Відсутність вбудованих функцій: На відміну від Django чи Spring Boot, Express.js не надає багато вбудованих функцій, що може потребувати більше налаштувань та конфігурацій.
- Менша продуктивність: У деяких випадках Express.js може бути менш продуктивним у порівнянні з іншими платформами через відсутність деяких оптимізацій.

5. Flask (Python)

Переваги:

- Легкість та простота: Flask є легким фреймворком з мінімальними налаштуваннями, що робить його простим у використанні.
- Гнучкість: Flask надає гнучкість у виборі компонентів та налаштувань, дозволяючи створювати як прості, так і складні додатки.
- Велика спільнота: Flask має активну спільноту, що забезпечує доступ до багатьох ресурсів та розширень.

Недоліки:

- Відсутність вбудованих функцій: Flask не надає багато вбудованих функцій, що може вимагати більше часу на розробку деяких функціональностей.

- Менша масштабованість: Flask може бути менш підходящим для великих проєктів через відсутність деяких масштабованих рішень.

6. Firebase (Google)

Графічний інтерфейс платформи наведений на рисунку 2.1.

Переваги:

- Безсерверна архітектура: Firebase пропонує безсерверну архітектуру, що усуває необхідність в управлінні серверами [12].
- Інтеграція з іншими сервісами Google: Firebase легко інтегрується з іншими сервісами Google, такими як Cloud Firestore, Authentication, Cloud Functions, Hosting, тощо.
- Реалтайм база даних: Firebase Realtime Database дозволяє легко створювати додатки з підтримкою реального часу.

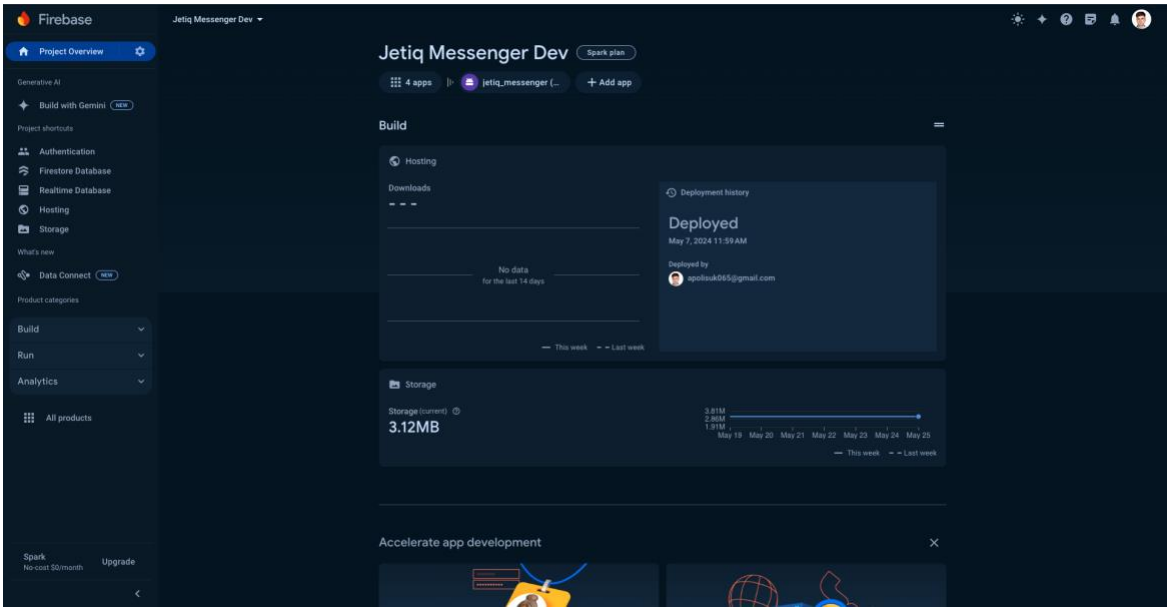

Рисунок 2.1 – Інтерфейс консолі Firebase

Недоліки:

- Вартість: Firebase може бути дорогим для великих проєктів через обмеження безкоштовного тарифу.

- Менша контрольованість: Безсерверна архітектура обмежує контроль над інфраструктурою, що може бути критичним для деяких проєктів.
- Обмеження масштабованості: Хоча Firebase добре підходить для малих та середніх проєктів, масштабованість може бути проблемою для дуже великих проєктів.

Проаналізувавши всі плюси та мінуси розглянутих платформ було вирішено зупинити вибір на Firebase. Головними причинами цього рішення були: безсерверна архітектура та простота інтеграції.

2.4 Вибір середовища розробки

Вибір середовища розробки є важливим аспектом успішної реалізації проєкту, оскільки воно впливає на продуктивність розробників, швидкість розробки та зручність тестування. Для розробки клієнтської частини на Flutter розглянемо два основні середовища: Android Studio та Visual Studio Code (VS Code).

2.4.1 Android Studio

Інтерфейс Android Studio з відкритим проєктом наведений на рисунку 2.2.

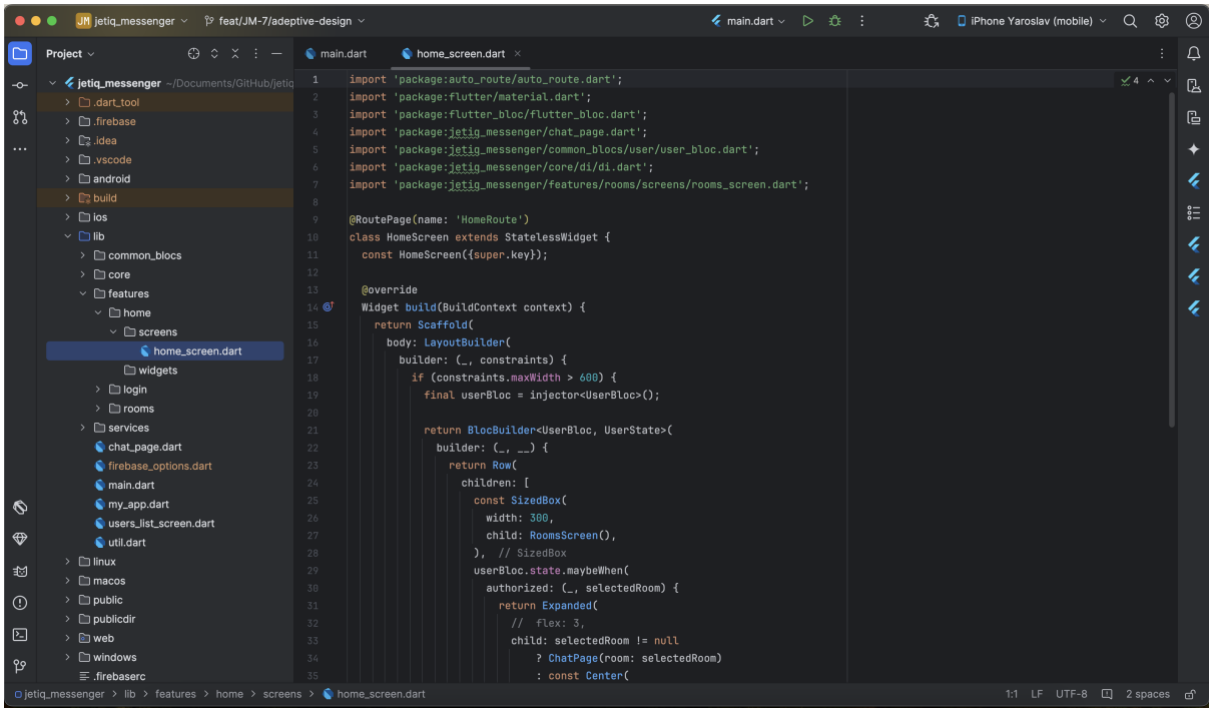

Рисунок 2.2 – Інтерфейс Android Studio

Переваги:

- Інтеграція з Android SDK: Android Studio має вбудовану інтеграцію з Android SDK, що спрощує налаштування середовища для розробки мобільних додатків.
- Потужний редактор коду: Редактор коду в Android Studio надає розширені можливості автозаповнення, аналізу коду та рефакторингу, що допомагає розробникам писати більш чистий та ефективний код.
- Інтегрована система збірки: Android Studio використовує Gradle як систему збірки, що дозволяє легко керувати залежностями та конфігураціями збірки [13].
- Розширені інструменти для тестування: Android Studio надає потужні інструменти для тестування, включаючи емулятори Android, інтеграцію з Firebase Test Lab та можливості для юніт-тестування.

- Підтримка Flutter: Хоча Android Studio спочатку призначений для розробки на Java/Kotlin, він має офіційну підтримку Flutter через плагін, що забезпечує зручну розробку Flutter-додатків.

Недоліки:

- Вимогливість до ресурсів: Android Studio є досить вимогливим до апаратних ресурсів, що може вплинути на продуктивність системи, особливо на старіших або менш потужних комп'ютерах.
- Складність налаштування: Налаштування Android Studio може бути складним для новачків, особливо якщо виникають проблеми з конфігурацією SDK або емуляторів.

2.4.2 Visual Studio Code (VS Code)

Інтерфейс програми Visual Studio Code наведений на рисунку 2.3

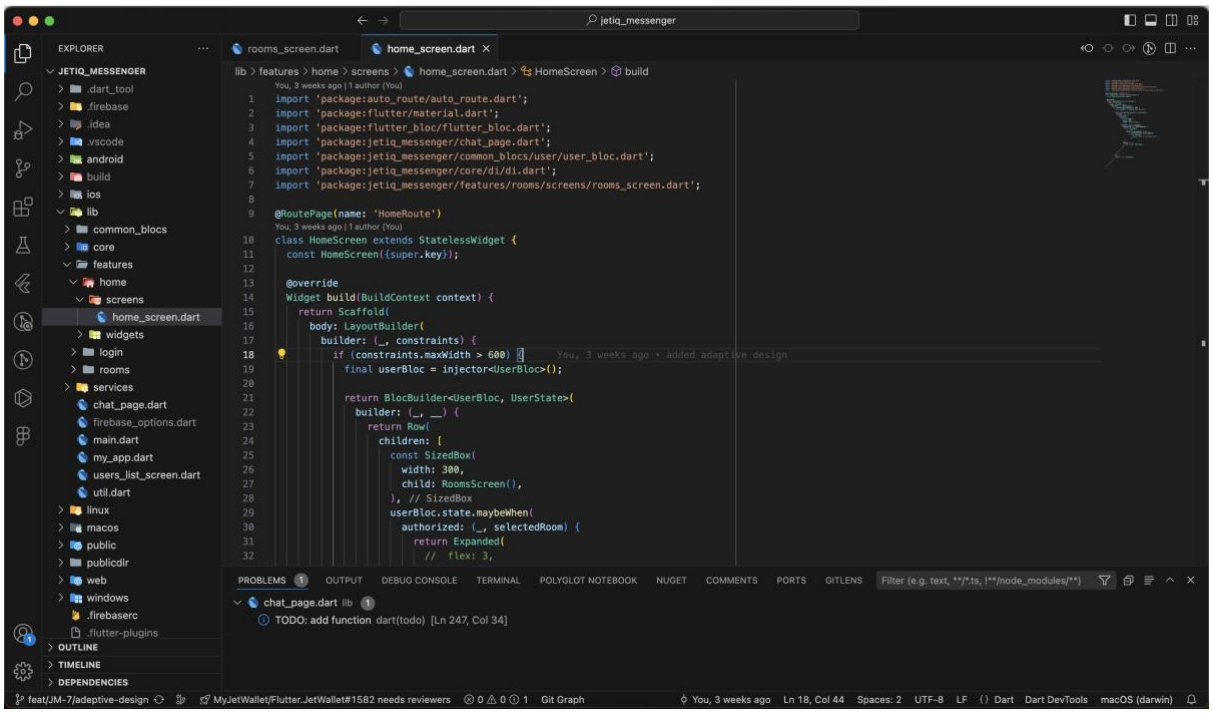

Рисунок 2.3 – Інтерфейс Visual Studio Code

Переваги:

- Легкість та швидкість: VS Code є легким редактором коду, який швидко завантажується і працює, споживаючи мінімальні ресурси системи.
- Розширення та плагіни: VS Code підтримує великий вибір розширень, зокрема для Flutter та Dart, що дозволяє налаштувати середовище під конкретні потреби розробки [14].
- Інтеграція з системами контролю версій: VS Code має потужну інтеграцію з Git та іншими системами контролю версій, що спрощує спільну роботу над проєктом.
- Налаштування та кастомізація: VS Code легко налаштовується та кастомізується через налаштування користувача та розширення, що дозволяє створити оптимальне середовище для розробки.
- Менша вимогливість до ресурсів: У порівнянні з Android Studio, VS Code споживає значно менше ресурсів, що робить його більш продуктивним на менш потужних комп'ютерах.

Недоліки:

- Менш інтегрована система збірки: У порівнянні з Android Studio, VS Code не має такої ж глибокої інтеграції з системою збірки, що може потребувати додаткових налаштувань для роботи з Gradle та іншими інструментами.
- Обмежені інструменти для тестування: VS Code має обмежені можливості для тестування порівняно з Android Studio, що може потребувати використання зовнішніх інструментів або налаштувань для повного циклу тестування.

2.4.4 Вибране середовище розробки

Для розробки клієнтської частини проєкту на Flutter як Android Studio, так і Visual Studio Code мають свої переваги та недоліки. В таблиці 2.2 наведено порівняння деяких характеристик та фонкцій Android Studio, так VS Code.

Таблиця 2.2 – Порівняння виконання для розробки проекту запитів Android Studio та Visual Studio Code

Параметр	**Android Studio**	**Visual Studio Code**
Системні вимоги		
Операційна система	Windows, macOS, Linux	Windows, macOS, Linux
Процесор	Intel Core i5 або вище	Intel Core i5 або вище
Оперативна пам'ять	8 GB (рекомендується 16 GB)	4 GB (рекомендується 8 GB)
Вільне місце на диску	10 GB	200 MB
Роздільна здатність екрана	1280 x 800 мінімум	1024 x 768 мінімум
Підтримка Flutter	Так	Так
Інтеграція з Firebase	Так	Так
Інструменти розробки		
Редактор коду	Вбудований редактор з автозавершенням та підсвічуванням синтаксису	Плагіни для Flutter і Dart
Інтеграція з VCS	Вбудована підтримка Git	Плагіни для Git

Підтримка розширень	Широкий набір вбудованих функцій	Величезний вибір плагінів і розширень
Налаштування середовища	Широкі можливості налаштування	Дуже гнучке налаштування через плагіни
Продуктивність		
Швидкість завантаження IDE	Повільніше (1-2 хвилини)	Швидке (близько 10-15 секунд)
Використання оперативної пам'яті	Високе	Низьке
Використання процесора	Високе	Низьке
Інші параметри		
Гарячий перезавантаження (Hot Reload)	Так	Так
Налагодження (Debugging)	Розширені можливості налагодження	Вбудовані засоби налагодження
Підтримка емуляторів/симуляторів	Вбудовані емулятори Android	Потрібне встановлення емуляторів окремо

Виходячи з даних таблиці Android Studio надає потужні інструменти для розробки, тестування та збірки додатків, але є вимогливим до апаратних ресурсів. VS Code, з іншого боку, є легким і швидким редактором, який легко налаштовується та споживає менше ресурсів, але може вимагати додаткових налаштувань для деяких задач. Вибір середовища розробки залежить від специфічних потреб проєкту та вподобань команди розробників. У нашому випадку, враховуючи необхідність швидкої та

ефективної розробки додатка, використання Visual Studio Code може бути більш оптимальним рішенням через його легкість, гнучкість та підтримку Flutter.

2.5 Вибір допоміжних інструментів

2.5.1 Система контролю версій

Для ефективного управління розробкою програмного забезпечення в сучасних проєктах необхідно використовувати допоміжні інструменти, що забезпечують контроль версій, співпрацю між розробниками та зберігання коду. У нашому проєкті для клієнтської частини такими інструментами будуть Git та GitHub.

Git – це розподілена система контролю версій, яка дозволяє розробникам відслідковувати зміни у коді, керувати різними версіями проєкту та співпрацювати з іншими розробниками. Використання Git у проєкті має кілька важливих переваг:

1. Контроль версій:

- Git дозволяє зберігати історію змін у коді, що дає змогу легко повернутися до попередніх версій у разі виникнення помилок або необхідності аналізу.
- Можливість створення гілок (branches) дозволяє розробникам працювати над новими функціями, виправленнями помилок або експериментальними змінами без впливу на основну кодову базу.

2. Спільна робота:

- Git забезпечує можливість одночасної роботи над проєктом для декількох розробників, дозволяючи об'єднувати зміни з різних

гілок за допомогою механізмів злиття (merge) та ребейзингу (rebase).

- Конфлікти злиття, що виникають при одночасних змінах одного й того ж файлу різними розробниками, можуть бути легко виявлені та вирішені.

3. Безпека та надійність:

- Розподілена природа Git дозволяє кожному розробнику мати повну копію репозиторію, що забезпечує надійність та безпеку даних.
- Можливість роботи в автономному режимі дозволяє розробникам вносити зміни та коміти навіть без доступу до інтернету.

GitHub – це веб-сервіс для хостингу Git-репозиторіїв, який надає додаткові інструменти для співпраці, управління проєктами та автоматизації (рис. 2.4).

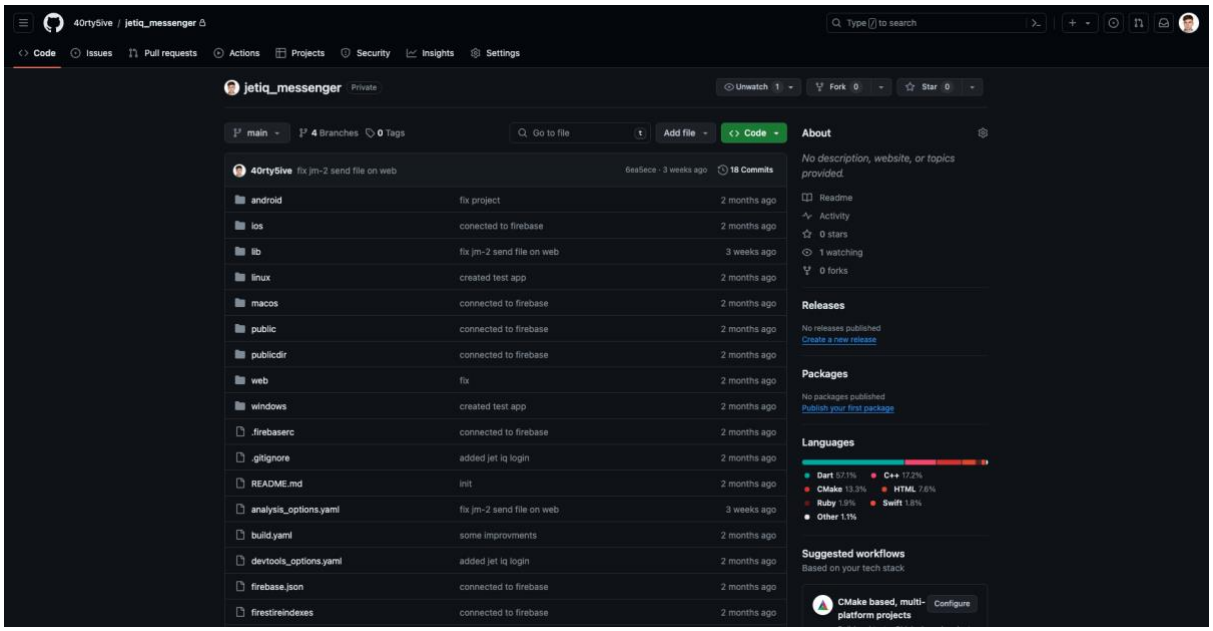

Рисунок 2.4 – Інтерфейс GitHub

Використання GitHub у проєкті додає наступні переваги:

1. Хостинг та резервне копіювання:
   - GitHub забезпечує зручний хостинг Git-репозиторіїв з можливістю приватного або публічного доступу, забезпечуючи надійне зберігання та резервне копіювання коду [15].
2. Інструменти для співпраці:
   - GitHub надає зручний інтерфейс для роботи з pull request'ами, що дозволяє легко переглядати та обговорювати зміни перед їх об'єднанням з основною кодовою базою.
   - Система управління задачами та проєктами (GitHub Issues та Projects) дозволяє відслідковувати завдання, планувати роботу та координувати зусилля команди.
3. Автоматизація та CI/CD:
   - GitHub Actions дозволяє налаштовувати автоматичні процеси, такі як збірка, тестування та розгортання додатків, що забезпечує безперервну інтеграцію та доставку (CI/CD).
   - Інтеграція з іншими сервісами та інструментами, такими як Slack, Trello та інші, дозволяє створювати гнучкі та ефективні робочі процеси.
4. Документація та спільнота:
   - GitHub Pages дозволяє створювати та хостити документацію проєкту, що забезпечує доступність інформації для всіх учасників команди та користувачів.
   - Велика спільнота розробників на GitHub сприяє обміну знаннями та досвідом, а також забезпечує доступ до численних відкритих репозиторіїв та бібліотек.

Використання Git та GitHub у нашому проєкті забезпечує надійний контроль версій, зручні інструменти для співпраці та управління проєктом,

а також можливості для автоматизації процесів розробки та тестування. Це дозволяє ефективно організувати роботу, забезпечуючи високу якість коду та швидкість впровадження нових функцій.

2.5.2 Система управління проєктом

Для організації робочого процесу та ефективного управління проєктом, важливим є використання потужного інструменту для відстеження завдань і координації роботи команди. Одним з найпопулярніших таких інструментів є Jira.

Jira – це система управління проєктами, розроблена компанією Atlassian, яка використовується для відстеження помилок, планування спринтів, управління завданнями та координації роботи команди (рис. 2.5). Вона особливо популярна серед команд, які використовують методології Agile, такі як Scrum і Kanban.

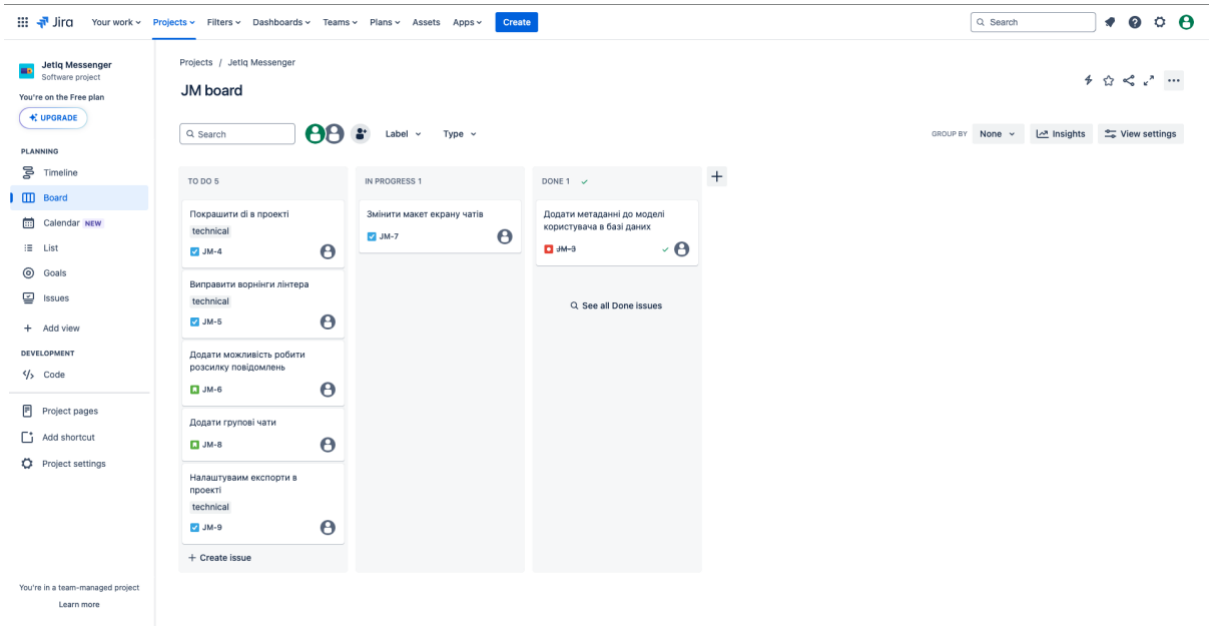

Рисунок 2.5 – Інтерфейс Jira

Переваги використання Jira

1. Гнучкість і налаштування:

- Jira дозволяє створювати проєкти, які можна налаштовувати відповідно до специфічних вимог команди, включаючи налаштування робочих процесів, полів завдань та типів завдань.
- Інструмент підтримує як Scrum, так і Kanban-дошки, що дозволяє адаптувати робочий процес до різних методологій управління проєктами.

2. Відстеження завдань та управління пріоритетами:

- У Jira можна створювати завдання (issues), підзадачі (sub-tasks), баги (bugs), історії (stories) та епіки (epics), що дозволяє детально структурувати роботу над проєктом.
- Завдання можуть мати різні пріоритети, етапи виконання та відповідальних осіб, що допомагає команді ефективно розподіляти ресурси та фокусуватися на найважливіших завданнях.

3. Планування спринтів і управління беклогом:

- Для команд, що працюють за методологією Scrum, Jira надає можливість створювати спринти, планувати їх вміст, а також управляти беклогом проєкту.
- Функціонал для оцінки завдань (story points) та управління навантаженням допомагає оптимально планувати роботу команди.

4. Візуалізація прогресу:

- Jira надає інструменти для візуалізації прогресу роботи, включаючи Kanban-дошки, Scrum-дошки, діаграми згорання (burndown charts) та діаграми прогресу (burnup charts).

- Ці інструменти дозволяють команді і зацікавленим сторонам швидко оцінити стан проєкту, виявити можливі проблеми та вжити необхідних заходів.

5. Інтеграція з іншими інструментами:

- Jira інтегрується з багатьма іншими інструментами, такими як Confluence (для документування), Bitbucket та GitHub (для контролю версій), Slack (для комунікації) та іншими.
- Це дозволяє створити єдину екосистему інструментів, що забезпечує безшовний робочий процес та покращує ефективність команди.

У контексті нашого проєкту "Месенджер для університетської системи" на основі Flutter, Jira буде використовуватись для наступних цілей:

1. Планування та розподіл завдань:

- Створення завдань для різних етапів розробки, таких як дизайн інтерфейсу, розробка функціоналу, тестування та інтеграція.
- Встановлення пріоритетів та дедлайнів для кожного завдання, що допоможе команді ефективно планувати свій час та ресурси.

2. Відстеження прогресу:

- Використання Scrum-дошки для планування спринтів та відстеження прогресу виконання завдань.
- Регулярні зустрічі (stand-ups) для обговорення виконаних завдань, виниклих проблем та планування наступних кроків.

3. Співпраця та комунікація:

- Ведення документації та обговорень у завданнях Jira, що забезпечить прозорість процесу розробки та доступність інформації для всієї команди.

- Використання інтеграцій з іншими інструментами, такими як GitHub, для автоматичного відстеження змін у коді та оновлення статусу завдань.

4. Аналіз та звітність:

- Вивчення діаграм прогресу для оцінки ефективності роботи команди та виявлення можливих проблем на ранніх етапах.
- Підготовка звітів для зацікавлених сторін, що дозволить їм отримувати актуальну інформацію про стан проєкту та приймати обґрунтовані рішення.

Використання Jira в нашому проєкті забезпечить організованість та ефективність робочого процесу та зосередитися на досягненні основних цілей. Це інструмент, який дозволить нам відстежувати прогрес, управляти завданнями та забезпечити успішне завершення проєкту.

3 ПРОЕКТУВАННЯ МОДЕЛІ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Визначення мети додатку

Метою розробки додатку "Месенджер для університетської електронної навчальної системи" є створення ефективного, зручного та функціонального інструменту для комунікації між учасниками навчального процесу у Вінницькому національному технічному університеті. Додаток має забезпечити можливість обміну текстовими повідомленнями та файлами у режимі реального часу, що сприятиме покращенню взаємодії між студентами та викладачами, а також підвищить ефективність організації навчального процесу.

Основними завданнями, які покликаний вирішити цей додаток, є надання користувачам можливості швидко та зручно обмінюватися інформацією без необхідності використовувати сторонні платформи чи засоби зв'язку. Особлива увага приділяється забезпеченню безпеки передачі даних, захисту особистої інформації користувачів та інтеграції з існуючою університетською системою JetIQ, що дозволить синхронізувати дані користувачів та забезпечити автентифікацію.

Додаток розробляється з урахуванням потреб і особливостей навчального процесу, що дозволить створити максимально корисний та зручний інструмент для студентів і викладачів. Важливим аспектом є також забезпечення стабільної роботи додатку при великій кількості користувачів, що досягається за рахунок використання сучасних технологій, таких як Flutter для клієнтської частини та Firebase для серверної частини.

Таким чином, основна мета додатку – це створення інтегрованого, надійного та зручного у використанні інструменту для комунікації в межах

ВНТУ, який дозволить покращити взаємодію між учасниками навчального процесу та зробити обмін інформацією більш ефективним і безпечним.

3.2 Створення моделі даних

Моделі даних відіграють ключову роль у структуризації та організації інформації в програмному забезпеченні. Існує кілька типів моделей даних, кожна з яких має свої переваги і недоліки в залежності від конкретного застосування. Основними моделями даних є:

1. Реляційна модель:

Опис: Реляційна модель базується на використанні таблиць для представлення даних [18]. Кожна таблиця складається з рядків і стовпців, де рядки представляють записи, а стовпці – атрибути.

Переваги: Висока нормалізація даних, зменшення дублювання, потужні засоби для запитів (SQL).

Недоліки: Не завжди ефективна для зберігання складних або ієрархічних даних.

2. Ієрархічна модель:

Опис: Ієрархічна модель організовує дані у вигляді дерева, де кожен вузол має одного батька і може мати кілька нащадків.

Переваги: Природна репрезентація ієрархічних даних, ефективність у виконанні певних типів запитів.

Недоліки: Складність у реалізації багатьох-з-багатьма зв'язків, обмежена гнучкість.

3. Мережева модель:

Опис: Мережева модель є розвитком ієрархічної моделі і дозволяє кожному вузлу мати кількох батьків.

Переваги: Гнучкість у репрезентації складних взаємозв'язків.

Недоліки: Складність у реалізації та управлінні, важко масштабувати.

4. Об'єктно-орієнтована модель:

Опис: Об'єктно-орієнтована модель представляє дані у вигляді об'єктів, як у програмуванні. Кожен об'єкт має атрибути і методи.

Переваги: Природна відповідність об'єктно-орієнтованому програмуванню, легкість у представленні складних даних.

Недоліки: Може бути складною для реалізації в реляційних СУБД, потребує додаткових шарів абстракції (ORM).

5. Документна модель:

Опис: Документна модель організовує дані у вигляді документів (наприклад, JSON або XML) [23], де кожен документ може мати складну ієрархічну структуру.

Переваги: Гнучкість у зберіганні неоднорідних даних, природна підтримка складних ієрархій.

Недоліки: Відсутність жорстких схем, що може ускладнити підтримку даних, обмежена потужність запитів у порівнянні з SQL.

6. Графова модель:

Опис: Графова модель представляє дані у вигляді вузлів і ребер, що з'єднують ці вузли.

Переваги: Природна відповідність для представлення взаємозв'язків, висока ефективність для певних типів запитів (наприклад, соціальні мережі).

Недоліки: Складність у реалізації, обмежена підтримка у порівнянні з реляційними СУБД.

Для цього проєкту було обрано документну модель даних. Цей вибір був зроблений на основі аналізу вимог до зберігання даних та функціональності додатку. Основними причинами вибору документної моделі є:

- Гнучкість у зберіганні даних: додаток повинен зберігати різні типи даних, включаючи текстові повідомлення, файли,

інформацію про користувачів та їх аватарки. Документна модель дозволяє легко зберігати ці неоднорідні дані у вигляді документів.

- Підтримка складних структур: повідомлення можуть містити вкладені структури, такі як списки вкладень, що природно підтримується документною моделлю.
- Інтеграція з Firebase: firebase Firestore, як платформа для зберігання даних, використовує документну модель [19]. Це забезпечує легкість інтеграції та використання вбудованих можливостей Firebase для зберігання, синхронізації та доступу до даних у реальному часі.
- Динамічне оновлення: документна модель, підтримувана Firebase, дозволяє легко реалізувати динамічне оновлення даних, що є критичним для додатка з обміном повідомленнями у реальному часі.

Отже, документна модель даних виявилася найкращим вибором для реалізації потреб проєкту, забезпечуючи гнучкість, зручність зберігання різнорідних даних та підтримку динамічного оновлення, що відповідає вимогам сучасного месенджера для університетської системи.

На рисунку 3.1 показано структуру колекції "users" у базі даних Firebase Firestore, яка містить інформацію про користувачів додатку. Колекція складається з документів, де кожен документ представляє окремого користувача. Документи організовані за унікальними ідентифікаторами (ID).

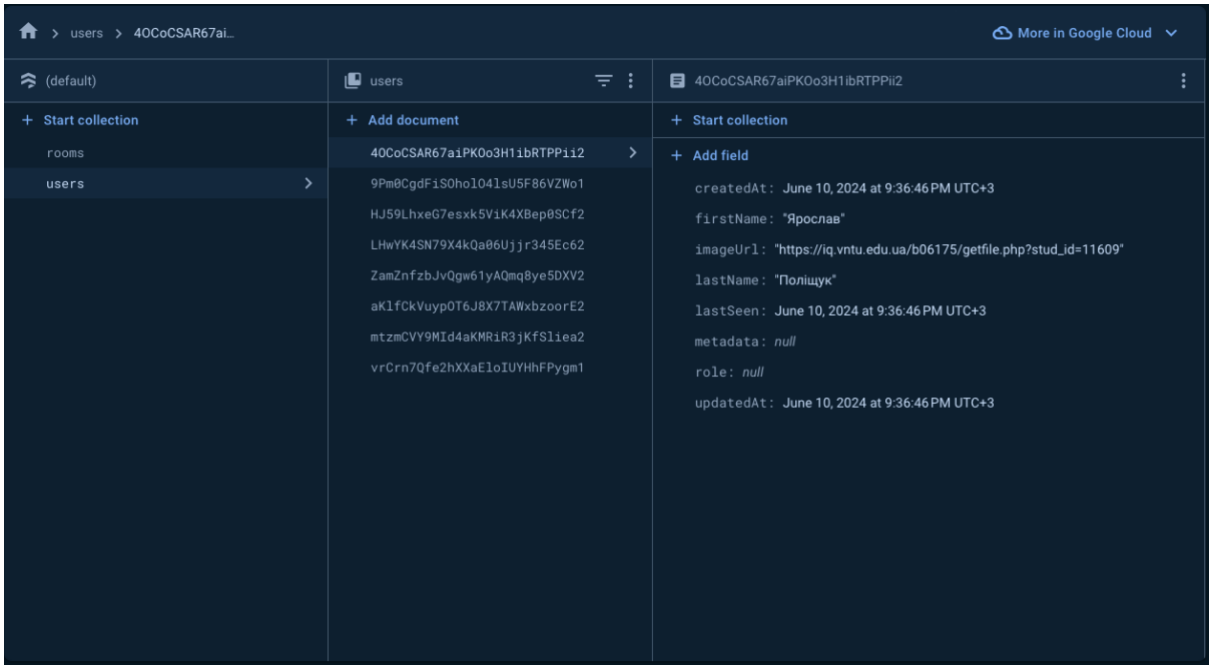

Рисунок 3.1 – Структура колекції users

Ось детальний опис структури одного з документів у колекції "users":

1. createdAt:
   - Опис: Дата і час створення документа.
   - Тип даних: Timestamp.
   - Приклад значення: June 10, 2024 at 9:36:46 PM UTC+3.
2. firstName:
   - Опис: Ім'я користувача.
   - Тип даних: String.
   - Приклад значення: Ярослав.
3. imageUrl:
   - Опис: URL зображення профілю користувача.
   - Тип даних: String.
   - Приклад значення: https://iq.vntu.edu.ua/b06175/getfile.php?stud_id=11609.
4. lastName:

- Опис: Прізвище користувача.
- Тип даних: String.
- Приклад значення: Поліщук.

5. lastSeen:
   - Опис: Дата і час останнього перебування користувача в мережі.
   - Тип даних: Timestamp.
   - Приклад значення: June 10, 2024 at 9:36:46 PM UTC+3.
6. metadata:
   - Опис: Додаткові метадані про користувача.
   - Тип даних: Map.
   - Приклад значення: null.
7. role:
   - Опис: Роль користувача у системі (наприклад, студент, викладач).
   - Тип даних: String.
   - Приклад значення: user.
8. updatedAt:
   - Опис: Дата і час останнього оновлення документа.
   - Тип даних: Timestamp.
   - Приклад значення: June 10, 2024 at 9:36:46 PM UTC+3.

Колекція "users" містить декілька документів, кожен з яких представляє окремого користувача з унікальним ідентифікатором. Ця структура дозволяє легко додавати, оновлювати та видаляти інформацію про користувачів. Кожен документ містить стандартний набір полів, що забезпечує збереження важливої інформації про користувача, такої як ім'я, прізвище, URL зображення профілю, дата створення та останнього оновлення, а також дату останнього перебування в мережі.

Ця структура забезпечує гнучкість і масштабованість, що є критичним для системи з великою кількістю користувачів. Додаткові поля,

такі як metadata і role, дозволяють розширювати інформацію про користувачів у майбутньому без необхідності змінювати основну структуру бази даних.

На рисунку 3.2 показано структуру колекції "rooms" у базі даних Firebase Firestore, яка містить інформацію про чати (кімнати) у додатку. Колекція складається з документів, де кожен документ представляє окрему кімнату. Документи організовані за унікальними ідентифікаторами (ID).

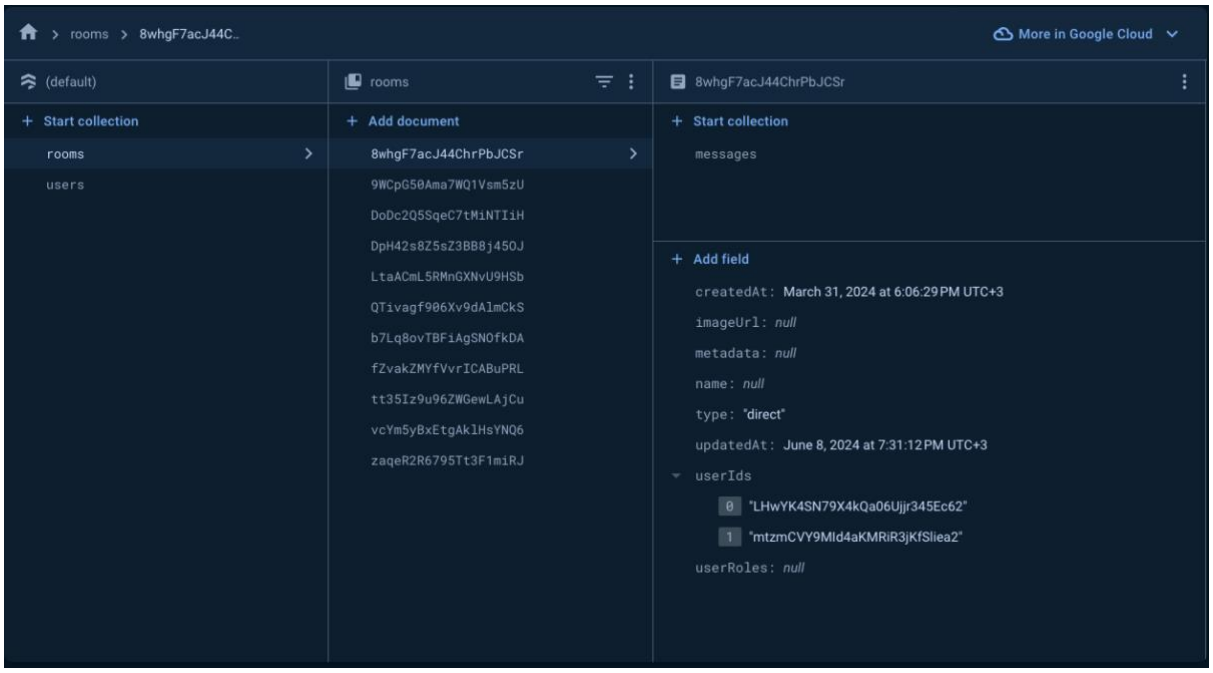

Рисунок 3.2 – Структура колекції rooms

Ось детальний опис структури одного з документів у колекції "rooms":

1. createdAt:
   - Опис: Дата і час створення документа.
   - Тип даних: Timestamp.
   - Приклад значення: March 31, 2024 at 6:06:29 PM UTC+3.
2. imageUrl:
   - Опис: URL зображення кімнати.

- Тип даних: Nullable String.
- Приклад значення: null.

3. metadata:
   - Опис: Додаткові метадані про кімнату.
   - Тип даних: Nullable.
   - Приклад значення: null.
4. name:
   - Опис: Назва кімнати.
   - Тип даних: Nullable String.
   - Приклад значення: null.
5. type:
   - Опис: Тип кімнати (наприклад, "direct" для прямих повідомлень).
   - Тип даних: String.
   - Приклад значення: direct.
6. updatedAt:
   - Опис: Дата і час останнього оновлення документа.
   - Тип даних: Timestamp.
   - Приклад значення: June 8, 2024 at 7:31:12 PM UTC+3.
7. userIds:
   - Опис: Список ID користувачів, які є учасниками кімнати.
   - Тип даних: Array of Strings.
   - Приклад значень: [LHwYK4SN79X4kQa06Ujjr345Ec62, mtzmCVY9MId4aKMRI3jKfSliea2]
8. userRoles:
   - Опис: Ролі користувачів у кімнаті.
   - Тип даних: Nullable.
   - Приклад значення: null.

Колекція "rooms" містить декілька документів, кожен з яких представляє окрему кімнату з унікальним ідентифікатором. Ця структура дозволяє легко додавати, оновлювати та видаляти інформацію про чати. Кожен документ містить стандартний набір полів, що забезпечує збереження важливої інформації про кімнату, такої як тип кімнати, дата створення та останнього оновлення, а також список учасників.

Документи в колекції "rooms" можуть містити вкладені колекції. У прикладі рисунку 3.4 видно вкладену колекцію "messages", що дозволяє зберігати повідомлення, пов'язані з конкретною кімнатою. Це забезпечує зручну та організовану структуру для зберігання та обробки даних, що є критично важливим для додатків з функціональністю чату.

На рисунку 3.3 показано структуру колекції "messages" у базі даних Firebase Firestore, яка міститься всередині документа колекції "rooms". Кожен документ у колекції "messages" представляє окреме повідомлення в чаті (кімнаті). Колекція складається з документів, кожен з яких має унікальний ідентифікатор (ID).

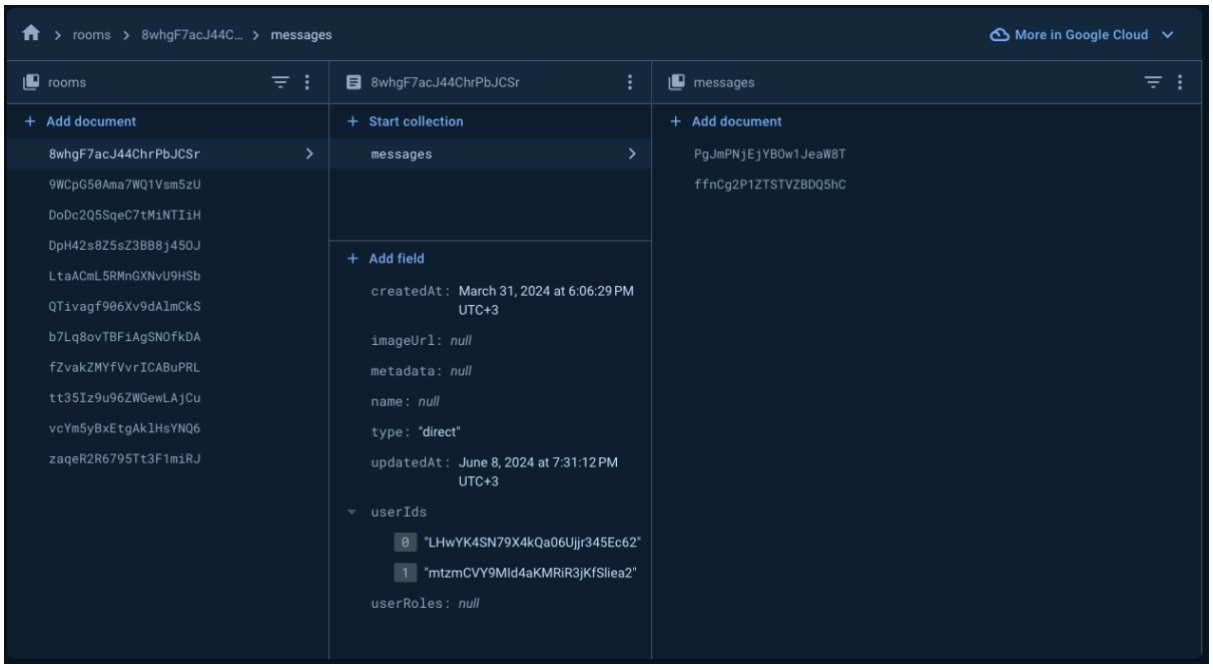

Рисунок 3.3 – Структура колекції messages

Ось детальний опис структури одного з документів у колекції "messages":

1. authorId:
   - Опис: Ідентифікатор користувача, який створив повідомлення.
   - Тип даних: String.
   - Приклад значення: LHwYK4SN79X4kQa06Ujjr345Ec62.
2. createdAt:
   - Опис: Дата і час створення повідомлення.
   - Тип даних: Timestamp.
   - Приклад значення: June 8, 2024 at 7:31:12 PM UTC+3.
3. text:
   - Опис: Текст повідомлення.
   - Тип даних: String.
   - Приклад значення: test.
4. type:
   - Опис: Тип повідомлення (наприклад, "text" для текстових повідомлень).
   - Тип даних: String.
   - Приклад значення: text.
5. updatedAt:
   - Опис: Дата і час останнього оновлення повідомлення.
   - Тип даних: Timestamp.
   - Приклад значення: June 8, 2024 at 7:31:12 PM UTC+3.

Колекція "messages" може містить декілька документів, кожен з яких представляє окреме повідомлення з унікальним ідентифікатором. Ця структура дозволяє легко додавати, оновлювати та видаляти повідомлення в чаті. Кожен документ містить стандартний набір полів, що забезпечує

збереження важливої інформації про повідомлення, такої як автор повідомлення, текст, дата створення та останнього оновлення.

Колекція "messages" розміщена всередині документа колекції "rooms", що дозволяє чітко зв'язувати повідомлення з конкретною кімнатою. Це забезпечує зручну та організовану структуру для зберігання та обробки повідомлень, що є критично важливим для додатків з функціональністю чату.

На основі необхідних даних побудуємо результуючу ER-діаграму типів предметної галузі (рис. 3.4).

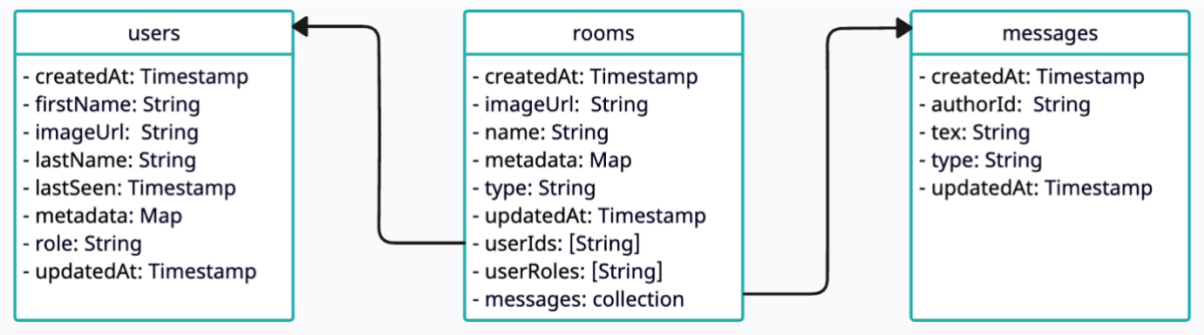

Рисунок 3.4 – Результуюча ER-діаграма

Використовуйчи класи Dart для моделі даних створили класи Dart, що відповідають кожній сутності в додатку. Додамо необхідні поля до цих класів на основі атрибутів, які визначили раніше. Приклад моделі даних для сутності "User" у додатку (рис. 3.5)

```dart
/// A class that represents user.
@JsonSerializable()
@immutable
abstract class User extends Equatable {
  /// Creates a user.
  const User._({
    this.createdAt,
    this.firstName,
    required this.id,
    this.imageUrl,
    this.lastName,
    this.lastSeen,
    this.metadata,
    this.role,
    this.updatedAt,
  });

  const factory User({
    int? createdAt,
    String? firstName,
    required String id,
    String? imageUrl,
    String? lastName,
    int? lastSeen,
    Map<String, dynamic>? metadata,
    Role? role,
    int? updatedAt,
  }) = _User;
```

Рисунок 3.5 – Модель даних для сутності «User»

3.3 Архітектура додатка

BLoC (Business Logic Component) — це архітектурний патерн, який використовується для розділення бізнес-логіки від презентаційної логіки в додатках на основі Flutter [20]. Основна мета BLoC — забезпечити чітке розділення відповідальностей між різними частинами додатка, що спрощує тестування, повторне використання коду та обслуговування.

Основні принципи BLoC:

1. Реактивне програмування: BLoC активно використовує реактивне програмування для обробки подій та станів. Потоки (Streams) використовуються для передачі подій від інтерфейсу користувача до BLoC і для відправки нових станів назад до інтерфейсу.

2. Чисті функції: BLoC містить чисті функції, які отримують події та поточний стан як вхідні дані і повертають новий стан. Це забезпечує передбачуваність та контрольованість змін стану.
3. Відокремлення бізнес-логіки: Вся бізнес-логіка розміщується в окремих компонентах (BLoC), що дозволяє легко змінювати логіку, не впливаючи на інтерфейс користувача.

Компоненти BLoC

1. Events (Події): Події є вхідними даними для BLoC, які описують, що має відбутися. Події можуть бути викликані діями користувача (натискання кнопок, введення тексту тощо) або системними подіями (отримання даних з сервера).
2. States (Стани): Стани є вихідними даними BLoC, які описують поточний стан інтерфейсу користувача. Кожен стан відповідає конкретному вигляду або набору даних.
3. Bloc: Клас BLoC містить логіку для обробки подій та генерації нових станів. Він підписується на потоки подій і випускає нові стани у відповідь на ці події.

У нашому проєкті архітектура BLoC була обрана для управління станом та обробки бізнес-логіки додатка з кількох причин:

1. Чітке розділення логіки: Відокремлення бізнес-логіки від інтерфейсу користувача дозволяє легко розробляти та тестувати окремі компоненти. Це особливо важливо в додатках зі складною логікою, таких як наш месенджер.

2. Зручне тестування: Завдяки BLoC, бізнес-логіка інкапсулюється в окремих класах, що дозволяє легко писати модульні тести для кожного BLoC. Це забезпечує високу надійність та якість коду.
3. Реактивне програмування: Використання потоків для передачі даних між компонентами спрощує роботу з асинхронними операціями,

такими як запити до сервера, обробка подій користувача та оновлення інтерфейсу.

4. Повторне використання коду: Чітке розділення відповідальностей дозволяє використовувати один і той же BLoC для різних частин додатка або навіть для інших проектів.

3.4 Структура додатка

Проєкт має типову структуру для додатку, розробленого за допомогою Flutter з використанням Firebase [16] (рисунок 3.6). Нижче наведено опис основних папок і файлів у структурі проєкту:

1. .dart_tool: Містить службові файли та каталоги, необхідні для роботи Dart.
2. .firebase: Вміщує налаштування для інтеграції з Firebase.
3. .idea: Налаштування проєкту для середовища розробки IntelliJ IDEA.
4. .vscode: Налаштування проєкту для середовища розробки Visual Studio Code.
5. android: Містить файли та налаштування для побудови Android-версії додатку.
6. assets: Каталог для зберігання статичних ресурсів, таких як зображення, шрифти і т.д.
7. build: Містить файли, згенеровані під час процесу побудови проєкту.
8. ios: Містить файли та налаштування для побудови iOS-версії додатку.
9. lib: Основний каталог, де розташований код програми, написаний на мові Dart. Зазвичай тут знаходяться файли з бізнес-логікою додатку.
10. linux: Містить файли та налаштування для побудови Linux-версії додатку.
11. macos: Містить файли та налаштування для побудови macOS-версії додатку.

12. web: Містить файли та налаштування для побудови веб-версії додатку.
13. windows: Містить файли та налаштування для побудови Windows-версії додатку.

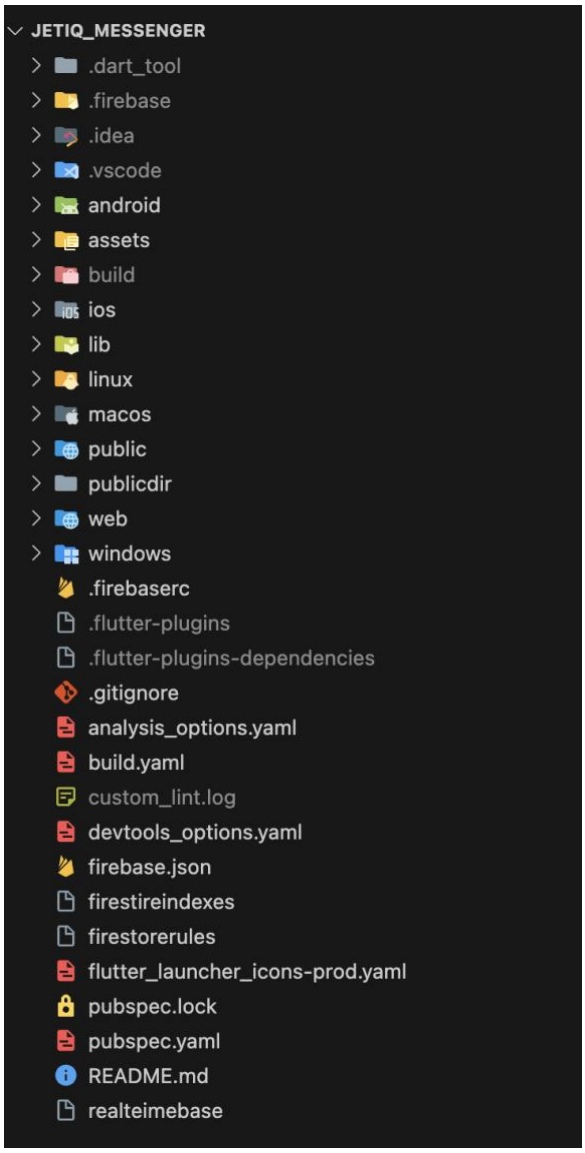

Рисунок 3.6 – Структура проєкту

Папки android, ios, macos, linux windows присутні в проєкті на випадок якщо потрібно буду портувати проєкт на ці платформи.

Також у кореневому каталозі проєкту знаходяться наступні файли:

1. .firebaserc: Файл налаштувань Firebase для різних середовищ (наприклад, production, development).
2. .flutter-plugins та .flutter-plugins-dependencies: Містять інформацію про використані плагіни Flutter.
3. .gitignore: Файл для визначення файлів та папок, які слід ігнорувати системі контролю версій Git.
4. analysis_options.yaml: Файл конфігурації аналізатора коду Dart.
5. build.yaml: Файл конфігурації побудови проєкту.
6. custom_lint.log: Лог-файл налаштувань лінтера.
7. devtools_options.yaml: Налаштування для DevTools.
8. firebase.json: Файл конфігурації Firebase.
9. firestoreindexes та firestorerules: Конфігураційні файли для налаштування індексів та правил безпеки Firebase Firestore.
10. flutter_launcher_icons-prod.yaml: Налаштування для генерації іконок додатку.
11. pubspec.lock та pubspec.yaml: Файли для управління залежностями Dart і Flutter.
12. README.md: Файл з інформацією про проєкт.
13. realtimebase: Можливо, конфігураційний файл для використання Firebase Realtime Database (назва файлу з помилкою, має бути "realtimebase.rules" або подібне).

Ця структура забезпечує можливість розробки, тестування та деплою додатку на різні платформи з використанням інструментів Flutter та Firebase.

Структура папки lib в цьому проєкті (рис. 3.7) добре організована та відповідає принципам модульності й розподілу обов'язків.

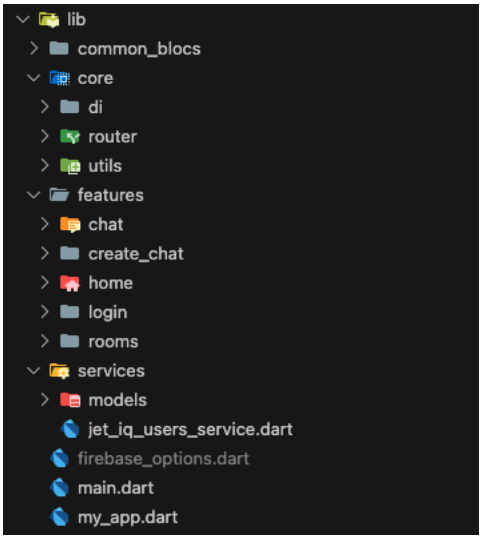

Рисунок 3.7 – Структура папки lib

Ось детальний опис:

1. common_blocs/user:
    - Містить загальну логіку BLoC (Business Logic Component)[17] для управління станом користувачів. BLoC є шаблоном проектування, який відокремлює бізнес-логіку від UI.
2. core:
    - di (Dependency Injection): Каталог для налаштувань залежностей. Це може включати ініціалізацію сервісів, репозиторіїв тощо, для інжектування в різні частини додатку.
    - router: Містить файли, що відповідають за маршрутизацію в додатку. Це може бути налаштування маршрутів, навігаційних гвардій і т.д.
    - utils: Каталог з утилітами та допоміжними класами, які використовуються в різних частинах додатку.
3. features:

Каталог для окремих функціональних модулів додатку. Кожен підкаталог відповідає за певну функціональність.

- chat: Модуль для функціоналу чату.

- create_chat: Модуль для створення нового чату.
- home: Модуль для головного екрану додатку.
- login: Модуль для логінізації користувачів.
- rooms: Модуль для управління кімнатами чату.

4. services:
    - Містить сервіси, що забезпечують функціональність додатку, зокрема, роботу з даними.
    - models: Каталог для моделей даних, які використовуються в сервісах.
    - jet_iq_users_service.dart: Сервіс для управління користувачами Jet IQ.
    - firebase_options.dart: Налаштування для інтеграції з Firebase.
5. main.dart:
    - Головний файл програми, який є точкою входу в додаток.
6. my_app.dart:
    - Файл, що містить головний клас додатку та його налаштування.

Ця структура дозволяє легко розширювати додаток, підтримувати його та тестувати окремі модулі. Кожна функціональна частина додатку ізольована у власному каталозі, що спрощує навігацію та управління кодом.

На рисунку 3.8 зображено структуру папки login. Вона містить в собі папки: bloc (де мітиться бізнес-логіка цього розділу додатку), screen (де містяться файли з екранами), widgets (де знаходяться ui-компоненти, які використовуються на екранах з сусідньої папки).

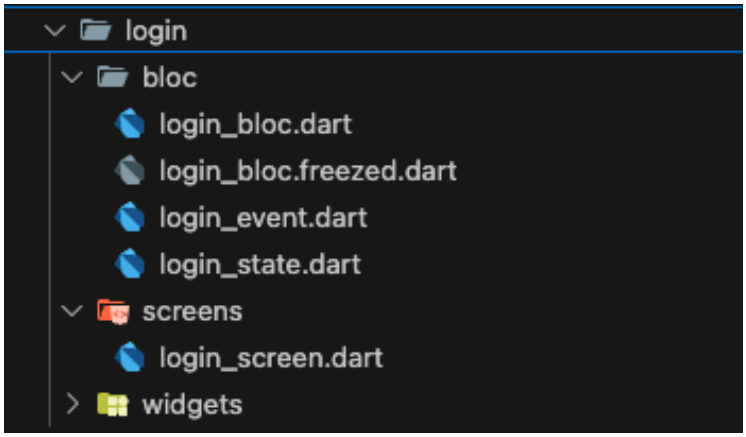

Рисунок 3.8 – Структура папки login

Папка login знаходиться в папці features, разом з іншими папками, які відповідають за конкретні модулі додатку і мають аналогічну структуру.

3.5 Реалізація додатку

Для розробки була використана MVVM архітектура. Суть цієї архітектури полягає в тому що код розділяється на шари: Model, ViewModel та View. Перейдемо до створення цих шарів

Клас UserBloc (рис. 3.7), який є частиною шару ViewModel в архітектурі MVVM, слугує для управління станом користувача в нашому додатку. UserBloc відповідає за обробку подій, пов'язаних із входом у систему, виходом з неї, перевіркою стану користувача при ініціалізації додатку, вибором кімнати чату та закриттям поточної кімнати чату. Нижче надано детальний опис кожної частини коду.

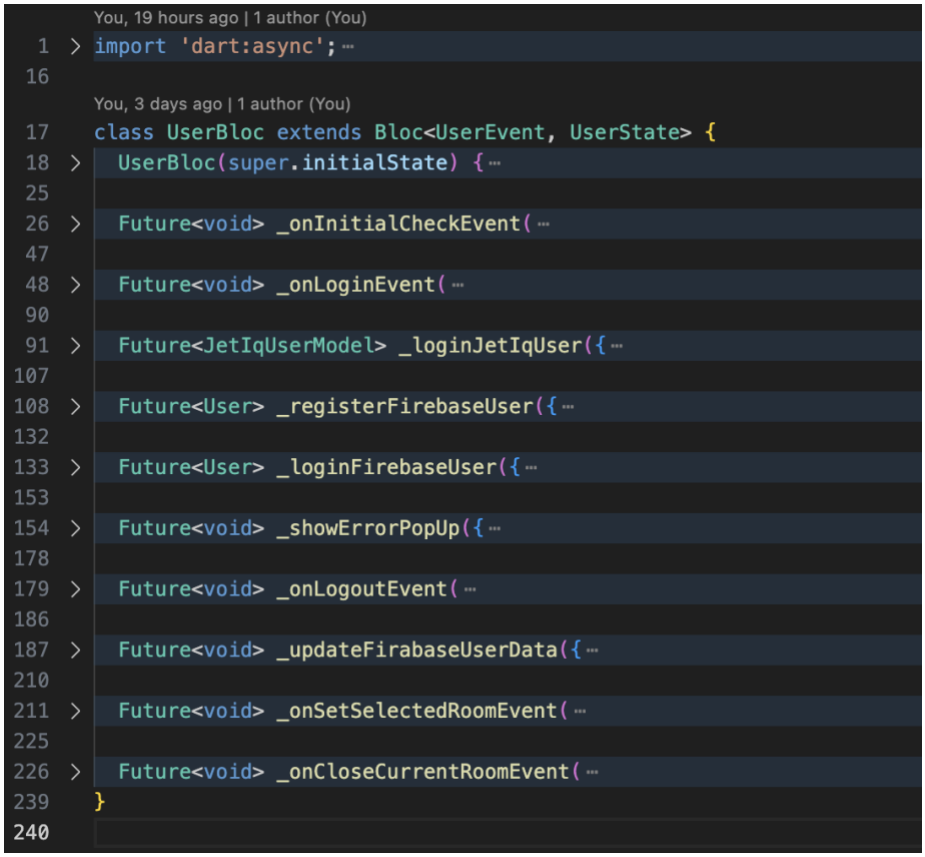

Рисунок 3.7 – Структура класу UserBloc

Клас UserBloc розширює базовий клас Bloc [22] та визначає початковий стан (рис. 3.8). Він також реєструє обробники подій, такі як on<_LoginEvent>, on<_LogoutEvent>, on<_InitCheckEvent>, on<_SetSelectedRoomEvent> та on<_CloseCurrentRoomEvent>, які відповідальні за виклик відповідних методів для обробки цих подій.

```
class UserBloc extends Bloc<UserEvent, UserState> {
  UserBloc(super.initialState) {
    on<_LoginEvent>(_onLoginEvent);
    on<_LogoutEvent>(_onLogoutEvent);
    on<_InitCheckEvent>(_onInitialCheckEvent);
    on<_SetSelectedRoomEvent>(_onSetSelectedRoomEvent);
    on<_CloseCurrentRoomEvent>(_onCloseCurrentRoomEvent);
  }
```

Рисунок 3.8 – Конструктор класу UserBloc

Метод _onInitialCheckEvent (рис. 3.9) обробляє подію ініціалізації. Він перевіряє, чи є користувач вже аутентифікованим у Firebase. Якщо користувач аутентифікований, він випромінює стан authorized, інакше — unauthorized. У разі виникнення помилки, випромінюється стан error.

```
Future<void> _onInitialCheckEvent(
  _InitCheckEvent event,
  Emitter<UserState> emit,
) async {
  try {
    final firebaseAuth = FirebaseAuth.instance;
    final user = firebaseAuth.currentUser;

    if (user != null) {
      emit(
        UserState.authorized(userInfo: user),
      );
    } else {
      emit(const UserState.unauthorized());
    }
  } catch (e) {
    emit(
      UserState.error(error: e.toString()),
    );
  }
}
```

Рисунок 3.8 – Метод onInitialCheckEvent

Метод _onLoginEvent (рис 3.9) обробляє подію входу в систему. Він намагається аутентифікувати користувача через JetIqUsersService і зареєструвати його у Firebase. Якщо користувач вже існує у Firebase, він намагається аутентифікувати його без реєстрації. У разі помилки відображається попередження через _showErrorPopUp.

```dart
Future<void> _onLoginEvent(
  _LoginEvent event,
  Emitter<UserState> emit,
) async {
  JetIqUserModel? jetIqUser;
  try {
    jetIqUser = await _loginJetIqUser(
      login: event.login,
      password: event.password,
    );

    final firebaseUser = await _registerFirebaseUser(jetIqUser: jetIqUser);

    emit(
      UserState.authorized(userInfo: firebaseUser),
    );
  } on FirebaseAuthException catch (error) {
    if (error.code == 'email-already-in-use') {
      final firebaseUser = await _loginFirebaseUser(
        login: event.login,
      );

      emit(
        UserState.authorized(userInfo: firebaseUser),
      );
    } else {
      await _showErrorPopUp(
        errorCase: error.toString(),
      );
    }
  } catch (error) {
    await _showErrorPopUp(
      errorCase: error.toString(),
    );
  } finally {
    if (jetIqUser != null && FirebaseAuth.instance.currentUser != null) {
      await _updateFirabaseUserData(
        jetIqUser: jetIqUser,
      );
    }
  }
}
```

Рисунок 3.9 – Метод _onLoginEvent

Метод _onLogoutEvent (рис. 3.10) обробляє подію виходу з системи, виконує вихід із Firebase і випромінює стан unauthorized.

```dart
Future<void> _onLogoutEvent(
  _LogoutEvent event,
  Emitter<UserState> emit,
) async {
  await FirebaseAuth.instance.signOut();
  emit(const UserState.unauthorized());
}
```

Рисунок 3.10 – Метод _onLogoutEvent

Метод _onSetSelectedRoomEvent (рис. 3.11) обробляє подію вибору кімнати чату. Якщо поточний стан authorized, він оновлює стан з обраною кімнатою.

```
Future<void> _onSetSelectedRoomEvent(
  _SetSelectedRoomEvent event,
  Emitter<UserState> emit,
) async {
  if (state is _UserStateAuthorized) {
    final userInfo = (state as _UserStateAuthorized).userInfo;
    emit(
      UserState.authorized(
        userInfo: userInfo,
        selectedRoom: event.room,
      ),
    );
  }
}
```

Рисунок 3.11 – Метод _onSetSelectedRoomEvent

Метод _onCloseCurrentRoomEvent обробляє подію закриття поточної кімнати чату. Якщо поточний стан authorized, він оновлює стан без вибраної кімнати.

```
Future<void> _onCloseCurrentRoomEvent(
  _CloseCurrentRoomEvent event,
  Emitter<UserState> emit,
) async {
  if (state is _UserStateAuthorized) {
    final userInfo = (state as _UserStateAuthorized).userInfo;
    emit(
      UserState.authorized(
        userInfo: userInfo,
      ),
    );
  }
}
```

Рисунок 3.11 – Метод _onCloseCurrentRoomEvent

Клас UserBloc дозволяє централізовано управляти станом користувача, обробляючи різні події та оновлюючи відповідно стан додатку, що забезпечує чітку організацію бізнес-логіки та зручність у тестуванні й масштабуванні.

Код (рис. 3.12) реалізує HomeScreen, головний екран додатку для відображення списку кімнат і чату, залежно від стану користувача та розміру екрану. Використовується архітектура BLoC для управління станом додатку. Нижче наведено детальний опис коду.

HomeScreen — це StatelessWidget, який відображає головний екран додатку. Він помічений анотацією @RoutePage, що дозволяє використовувати його з бібліотекою auto_route для навігації.

У методі build створюється екземпляр UserBloc за допомогою injector, який відповідає за управління станом користувача.

Scaffold є основним контейнером для екрану, а BlocBuilder від flutter_bloc використовується для побудови інтерфейсу залежно від стану UserBloc.

LayoutBuilder дозволяє визначати розмітку залежно від розміру екрану. Якщо ширина екрану більше 600 пікселів, інтерфейс буде розділено на два стовпці: лівий для списку кімнат (RoomsScreen), а правий для вмісту чату.

```dart
@RoutePage(name: 'HomeRoute')
You, 3 days ago | 1 author (You)
class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    final userBloc = injector<UserBloc>();

    return Scaffold(
      body: BlocBuilder<UserBloc, UserState>(
        builder: (_, __) {
          return LayoutBuilder(
            builder: (_, constraints) {
              if (constraints.maxWidth > 600) {
                return Row(
                  children: [
                    SizedBox(
                      width: 300,
                      child: RoomsScreen(),
                    ), // SizedBox
                    userBloc.state.maybeWhen(
                      authorized: (_, selectedRoom) {
                        return Expanded(
                          //  flex: 3,
                          child: selectedRoom != null
                              ? ChatPage(room: selectedRoom)
                              : const Center(
                                  child: Text('No selected room'),
                                ), // Center
                        ); // Expanded
                      },
                      orElse: () {
                        return const Offstage();
                      },
                    ),
                  ],
                ); // Row
              } else {
                return userBloc.state.maybeWhen<Widget>(
                  authorized: (_, selectedRoom) {
                    return selectedRoom != null
                        ? ChatPage(room: selectedRoom)
                        : RoomsScreen();
                  },
                  orElse: () {
                    return const Offstage();
                  },
                );
              }
            },
          ); // LayoutBuilder
        },
      ), // BlocBuilder
    ); // Scaffold
  }
}
```

Рисунок 3.12 –Клас HomeScreen

При успішній компіляції проєкту в залежності від стану UserBloc, відображаються різні компоненти. Якщо користувач авторизований (authorized), відображається або вибрана кімната (ChatPage), або повідомлення "No selected room" у випадку десктопного інтерфейсу (рис.

3.13). Якщо ширина екрану менше 600 пікселів, відображається або вибрана кімната, або список кімнат (рис. 3.14).

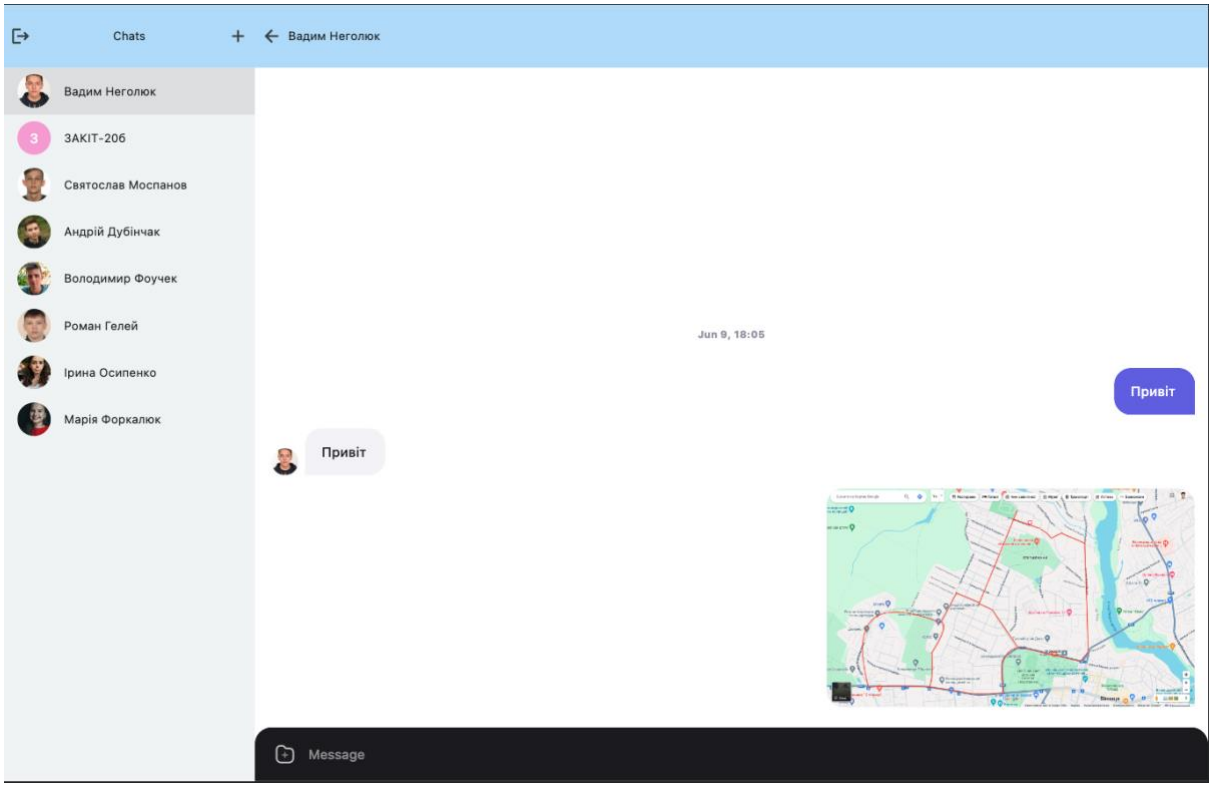

Рисунок 3.13 –Інтерфейс додатку в десктопному режимі

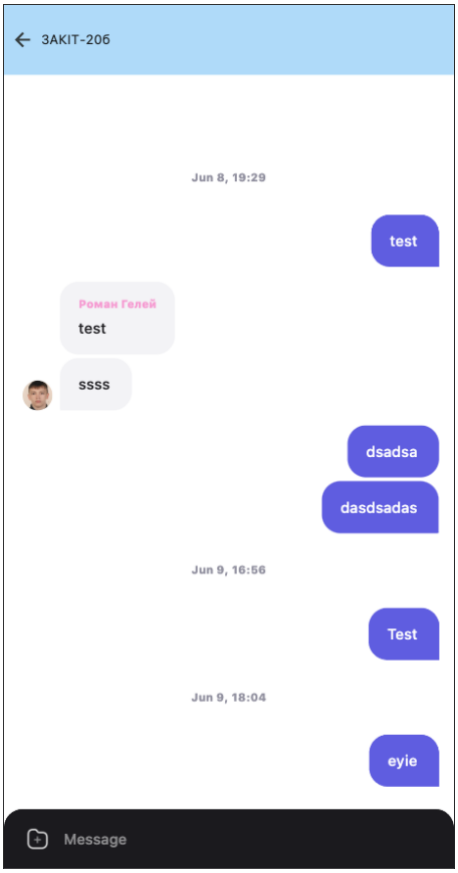

Рисунок 3.14 –Інтерфейс додатку в мобільному режимі

3.6 Висновки до розділу

В процесі розробки додатку для обміну повідомленнями на основі платформи Firebase було застосовано сучасні підходи та технології, зокрема архітектура BLoC для управління станом додатку. Це дозволило досягти високого рівня структурованості та зручності у підтримці та розвитку проєкту.

Використання Flutter забезпечило кросплатформеність додатку, що дозволяє запускати його на різних пристроях без додаткових витрат на розробку окремих версій. Інтеграція з Firebase спростила роботу з автентифікацією, зберіганням даних та управлінням користувачами. Це значно знизило навантаження на сервер та забезпечило високу масштабованість додатку.

Завдяки використанню пакету flutter_bloc, вдалося ефективно розділити бізнес-логіку від презентаційного шару додатку, що зробило код більш читабельним та підтримуваним. Це також спростило процес тестування та налагодження додатку. Впровадження авто маршрутизації з використанням auto_route покращило навігацію в додатку та зробило її більш інтуїтивно зрозумілою для користувачів.

Реалізація адаптивного інтерфейсу забезпечила зручне користування додатком як на великих екранах (настільні комп'ютери та планшети), так і на мобільних пристроях. Це дозволило оптимізувати користувацький досвід, незалежно від типу пристрою.

Загалом, підхід до розробки, який включав використання передових технологій та архітектурних патернів, дозволив створити надійний та функціональний додаток, що відповідає сучасним вимогам до якості та зручності програмного забезпечення.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО МОДУЛЯ

4.1 Вибір способу тестування

При виборі способу тестування розробленого програмного модуля було вирішено застосувати кілька методів для забезпечення комплексного підходу до перевірки якості [21]. Зокрема, було розглянуто такі методи тестування:

1. Юніт-тестування: для перевірки окремих функцій і компонентів додатку. Цей підхід дозволяє ізолювати та протестувати кожну одиницю коду на предмет коректності роботи.
2. Інтеграційне тестування: для перевірки взаємодії між різними компонентами додатку. Це допомагає виявити проблеми, що можуть виникати при інтеграції різних модулів [24].
3. Енд-то-енд тестування: для перевірки всієї системи в цілому. Даний метод тестування дозволяє відстежувати, як різні частини додатку працюють разом і забезпечити цілісність користувацького досвіду.
4. Тестування користувацького інтерфейсу (UI тестування): для перевірки коректності роботи інтерфейсу, його відповідності дизайну та зручності використання.
5. Користувацьке тестування: У цьому випадку реальні користувачі випробовують додаток на своїх пристроях. Вони надають зворотний зв'язок стосовно зручності використання, ергономіки і загального враження від додатку.
6. Тестування автоматизації: Це використання спеціальних інструментів та скриптів для автоматизації процесу тестування. Це може включати автоматизоване тестування функцій, навантажувальне тестування, тестування сумісності та інші.

4.2 Тестування додатку

Тестування користувацького інтерфейсу (UI тестування) є важливим етапом для забезпечення зручності та функціональності додатку з точки зору кінцевого користувача. У випадку нашого проєкту, UI тестування проводилося з метою перевірки відповідності інтерфейсу вимогам дизайну, його коректної роботи та зручності використання на різних пристроях і розмірах екранів.

Перш за все перевіримо авторизацію користувача. Очікуваним результатом є виведення помилки про неправильні дані у разі введення незареєстрованого чи помилкового логіну, помилкового паролю. У випадку правильних даних очікується успішний перехід на головну сторінку. На рисунку 4.1 представлено вигляд сторінки авторизації із введеними помилковими даними.

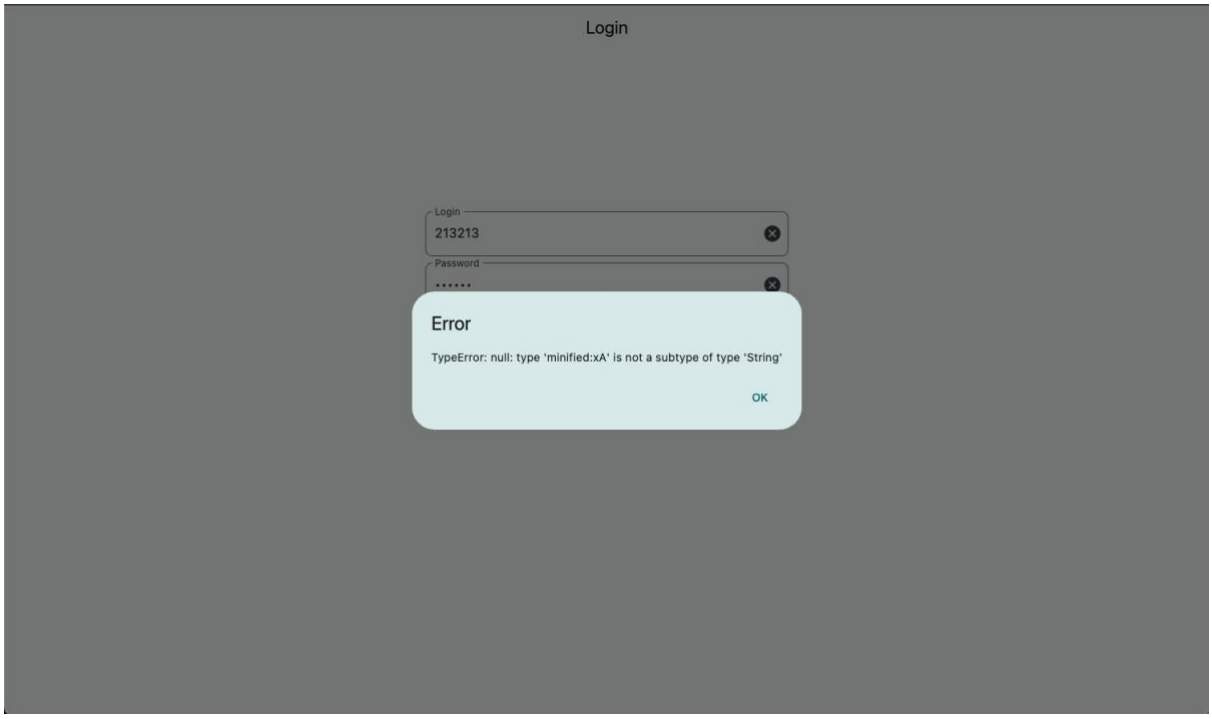

Рисунок 4.1 – Авторизація з помилковими даними

Введення валідних даних авторизації забезпечило перехід на головну сторінку користувача (рис. 4.2).

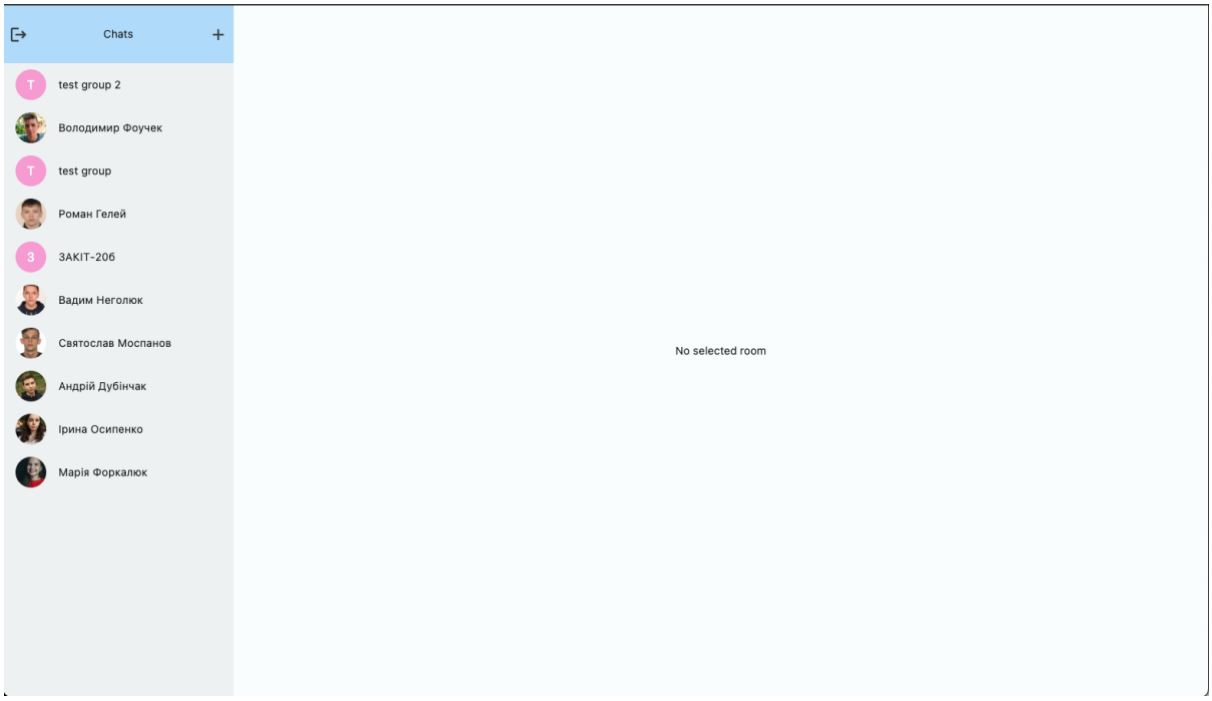

Рисунок 4.2 – Головна сторінка користувача

Дизайн головної сторінки адаптований до розширення екрану і на мобільних пристроях має інший вигляд (рис. 4.3).

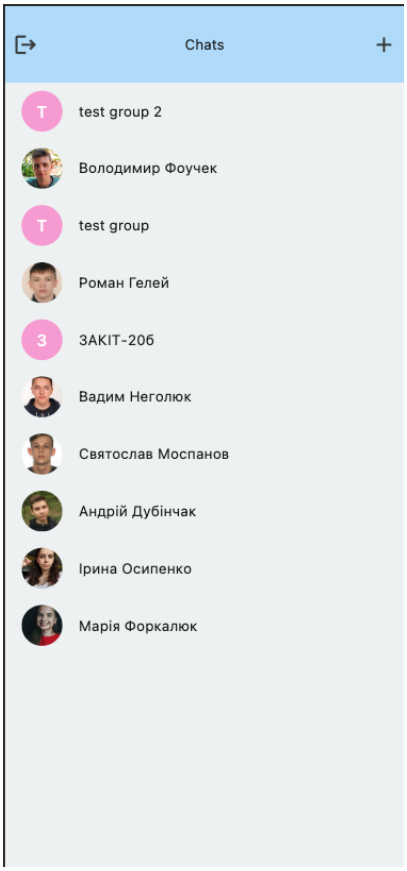

Рисунок 4.3 – Головна сторінка користувача на мобільному пристрої

Після вибору чату зі списку, деталі цього чату в випадку десктопного режиму з’являються на екрані справа (рис. 4.4), або відкриваються на повний екран в випадку мобільного додатку (рис. 4.5).

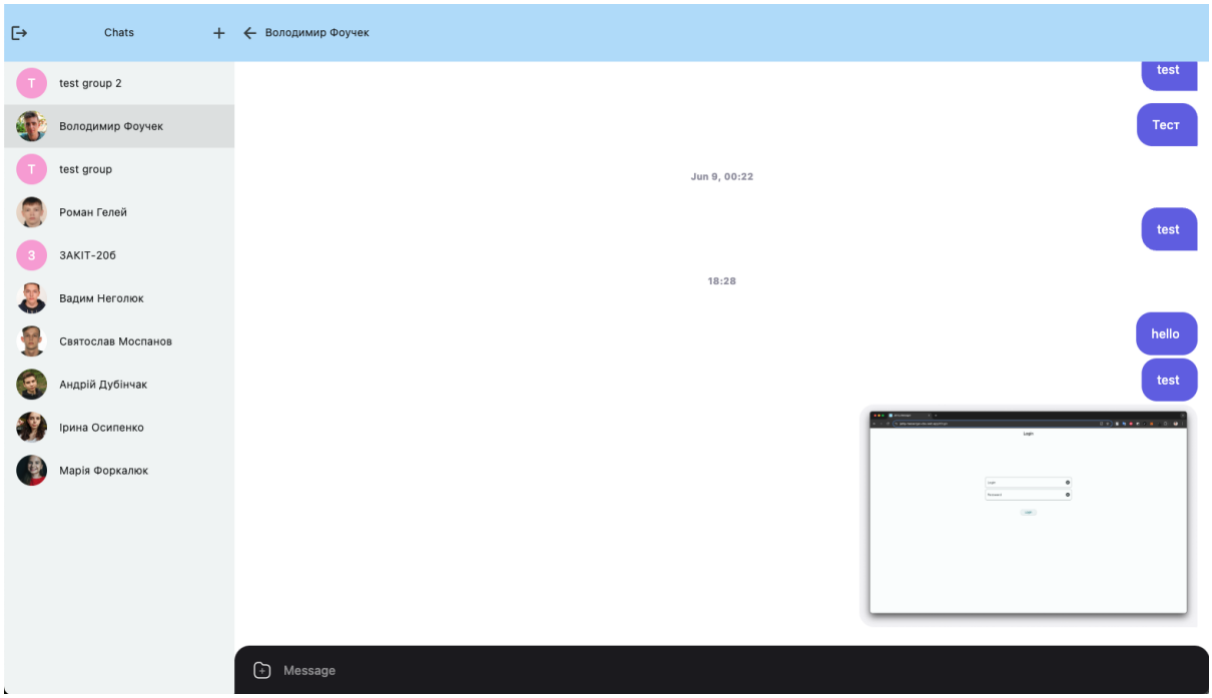

Рисунок 4.4 – Головна сторінка користувача з активним чатом

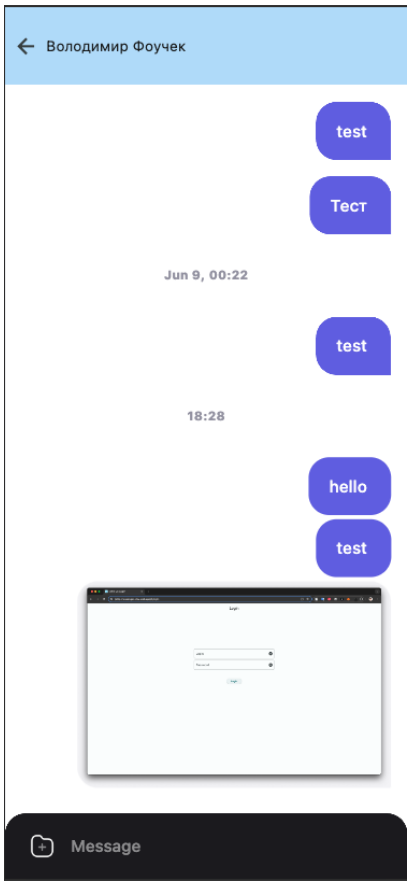

Рисунок 4.4 – Головна сторінка користувача з активним чатом на на мобільному пристрої

Також користувач має можливість, натиснувши на іконку «+» (рис. 4.5), розпочати новий чат з іншим користувачем (рис. 4.6), або створити груповий чат (рис. 4.7).

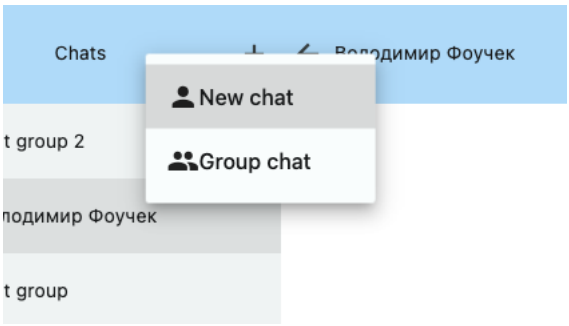

Рисунок 4.5 – Вікно, яке з’являється після натискання іконки «+»

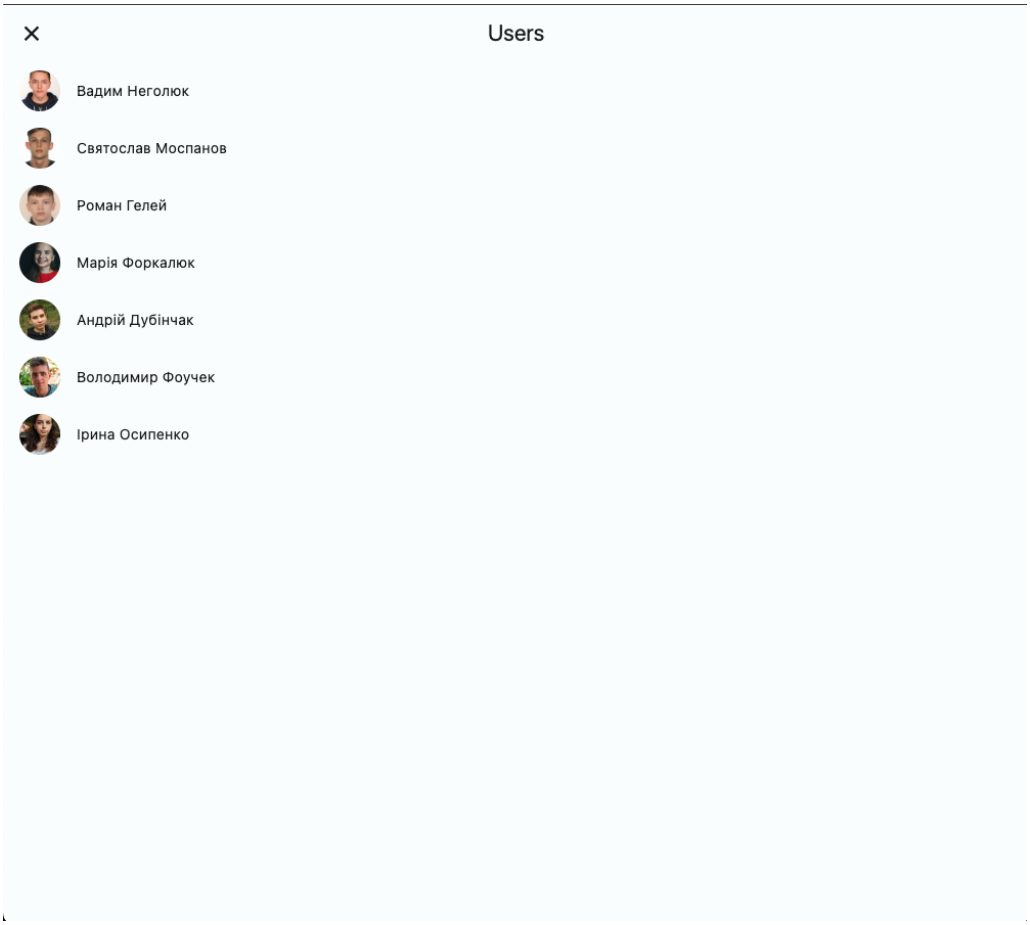

Рисунок 4.6 - Екран створення нового персонального чату

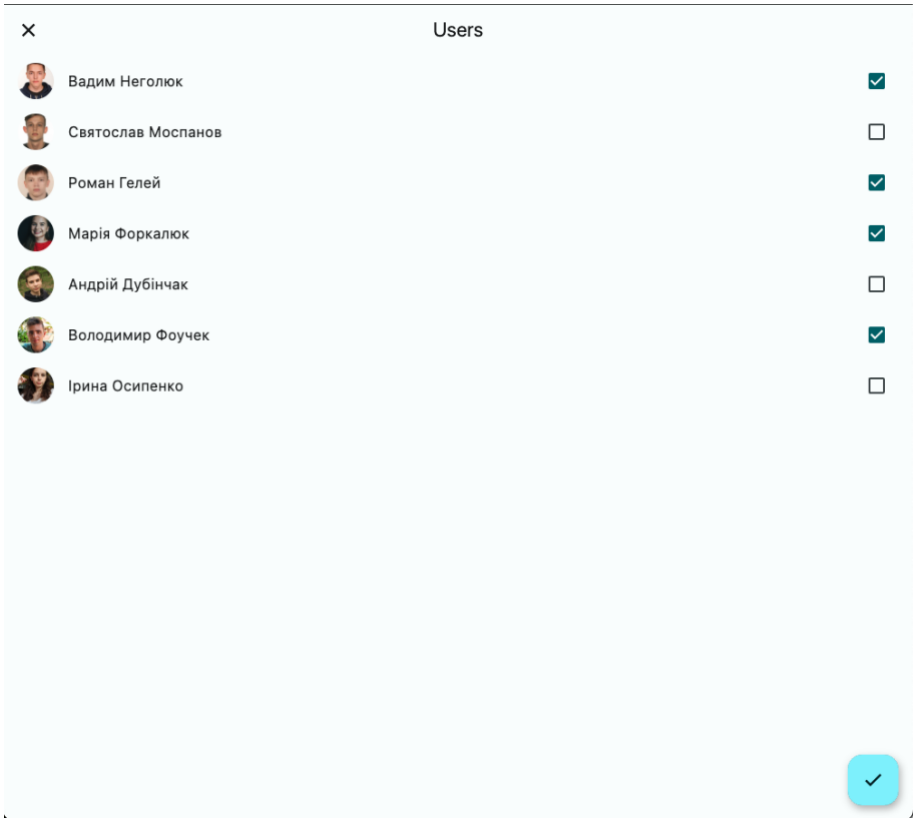

Рисунок 4.7 - Екран створення нового групового чату

В результаті проведеного тестування користувацького інтерфейсу вдалося виявити та усунути ряд проблем, таких як некоректне відображення елементів на деяких пристроях, помилки навігації та непередбачувана поведінка інтерфейсу при виконанні деяких дій. Після виправлення цих проблем було досягнуто високого рівня зручності та функціональності додатку, що забезпечує позитивний користувацький досвід.

Користувацьке тестування (User Testing) є важливим етапом перевірки додатку, оскільки дозволяє отримати зворотний зв'язок від реальних користувачів щодо його зручності та функціональності. У випадку нашого проєкту користувацьке тестування проводилося з метою виявлення проблем та отримання пропозицій щодо покращення додатку.

Етапи користувацького тестування:

1. Залучення тестових користувачів:

- Було залучено групу тестових користувачів, які представляли цільову аудиторію додатку, включаючи студентів, викладачів та адміністративний персонал університету.
- Користувачам було надано доступ до бета-версії додатку та інструкції щодо виконання основних дій та сценаріїв використання.

2. Збір зворотного зв'язку:

- Після використання додатку тестовими користувачами було зібрано зворотний зв'язок за допомогою анкет, інтерв'ю та обговорень.
- Користувачі ділилися своїми враженнями щодо зручності використання, функціональності, дизайну та загальної якості додатку.

3. Аналіз та виправлення проблем:

- На основі зібраного зворотного зв'язку було виявлено ряд проблем, таких як складність навігації, недоліки в дизайні деяких елементів, відсутність деяких функцій, які виявилися важливими для користувачів.
- Було проведено аналіз цих проблем та внесено відповідні виправлення до додатку.

Користувацьке тестування дозволило отримати цінну інформацію щодо реального використання додатку та його відповідності потребам користувачів. Після внесення змін, запропонованих користувачами, додаток став більш зручним та функціональним, що сприяє підвищенню задоволеності користувачів та ефективності його використання в університетській системі.

4.3 Висновки до розділу

Тестування розробленого програмного модуля, включаючи тестування користувацького інтерфейсу та користувацьке тестування,

дозволило забезпечити високий рівень якості та функціональності додатку. Завдяки комплексному підходу до тестування вдалося виявити та усунути всі виявлені проблеми, забезпечивши надійну та стабільну роботу додатку. Отриманий зворотний зв'язок від реальних користувачів дозволив покращити додаток та забезпечити його відповідність потребам цільової аудиторії.

ВИСНОВКИ

У даній бакалаврській роботі було розроблено автоматизований модуль для обміну повідомленнями та файлами в електронній освітній системі на базі платформи Firebase. Основною метою проекту було створення месенджера для комунікації між учасниками освітнього процесу у Вінницькому національному технічному університеті, з інтеграцією в систему JetIQ.

На початкових етапах роботи було здійснено аналіз існуючих рішень та обрано технологічний стек для реалізації проекту. Було вирішено використовувати Flutter для розробки інтерфейсу додатку та Firebase для забезпечення серверної частини і зберігання даних. Це дозволило забезпечити високу швидкість розробки, кросплатформеність та масштабованість додатку.

Архітектура додатку була побудована на основі шаблону BLoC (Business Logic Component), що дозволило відокремити бізнес-логіку від інтерфейсу користувача та забезпечити високу гнучкість і підтримуваність коду. Було реалізовано механізми реєстрації та аутентифікації користувачів, вибору чат-кімнат, обміну повідомленнями та файлами.

Особлива увага приділялась питанням безпеки та захисту даних користувачів. Використання Firebase Authentication забезпечило надійний механізм аутентифікації, а зберігання даних у Firestore дозволило забезпечити їх захист та доступність.

У розділі тестування було проведено перевірку функціональності додатку, включаючи тестування користувацького інтерфейсу та користувацьке тестування. Було виявлено та виправлено ряд помилок, що дозволило значно підвищити якість і стабільність роботи додатку.

Таким чином, розроблений додаток успішно вирішує поставлені завдання та відповідає вимогам, визначеним на початкових етапах проекту.

Месенджер забезпечує зручну та ефективну комунікацію між учасниками освітнього процесу у ВНТУ, а його інтеграція в систему JetIQ дозволяє використовувати додаток як невід'ємну частину загальної освітньої екосистеми університету.

Результати даної роботи можуть бути використані для подальшого розвитку проекту, додавання нових функціональних можливостей та оптимізації існуючих процесів. Додаток має потенціал для масштабування та впровадження в інших навчальних закладах, що сприятиме покращенню комунікації та організації навчального процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wikipedia: Google Classroom. URL: https://uk.wikipedia.org/wiki/Google_Classroom (дата звернення: 20.05.2024).
2. Wikipedia: Microsoft Teams. URL: https://uk.wikipedia.org/wiki/Microsoft_Teams (дата звернення: 20.05.2024).
3. What is Slack and How Can My Team Use It? URL: https://zight.com/blog/what-is-slack/ (дата звернення: 21.05.2024).
4. What is Discord? URL: https://discord.com/safety/360044149331-what-is-discord (дата звернення: 21.05.2024).
5. Що таке прогресивні веб-додатки: детальна відповідь URL: https://wezom.com.ua/ua/blog/pwa-prilozheniya-web-progressive-app (дата звернення: 21.05.2024).
6. TIOBE Index for May 2024 URL: https://www.tiobe.com/tiobe-index/ (дата звернення: 21.05.2024).
7. ReactDOM URL: https://uk.legacy.reactjs.org/docs/react-dom.html (дата звернення: 21.05.2024).
8. Vue.js як JavaScript-фреймворк URL: https://foxminded.ua/vue-js/ (дата звернення: 22.05.2024).
9. What is Django? URL: https://aws.amazon.com/what-is/django/ (дата звернення: 23.05.2024).
10. Pros & Cons of Ruby on Rails You Should Know Before Choosing the Technology for Your StartupURL URL: https://sloboda-studio.com/blog/pros-and-cons-of-ruby-on-rails/ (дата звернення: 23.05.2024).

11. Is Flutter Good for App Development? URL: https://www.purrweb.com/blog/is-flutter-good-pros-and-cons/ (дата звернення: 24.05.2024).
12. What is Google Firebase? URL: https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase (дата звернення: 25.05.2024).
13. Meet Android Studio URL: https://developer.android.com/studio/intro (дата звернення: 25.05.2024).
14. What is Visual Studio Code? Microsoft’s extensible code editor URL: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html (дата звернення: 25.05.2024).
15. What Is GitHub? A Beginner’s Introduction to GitHub URL: https://kinsta.com/knowledgebase/what-is-github/ (дата звернення: 25.05.2024).
16. Understanding the Flutter Project Structure URL: https://medium.com/@logeshgcp/understanding-the-flutter-project-structure-84de4ec3ce5f/ (дата звернення: 27.05.2024).
17. Bloc Concepts URL https://bloclibrary.dev/ru/bloc-concepts/ (дата звернення: 28.05.2024).
18. Data modeling overview URL https://www.sap.com/products/technology-platform/datasphere/what-is-data-modeling.html (дата звернення: 28.05.2024).
19. Choose a data structure URL https://firebase.google.com/docs/firestore/manage-data/structure-data (дата звернення: 28.05.2024).
20. How to Use Flutter BLoC Architecture To Build High-Performance App? URL https://ripenapps.com/blog/how-to-use-flutter-bloc-architecture-to-build-high-performance-apps/ (дата звернення: 28.05.2024).

21. Software Testing Methodologies URL https://smartbear.com/learn/automated-testing/software-testing-methodologies/ (дата звернення: 30.05.2024).
22. Jurnal Mantik, Performance Analysis of BLoCand Provider State Management Library on Flutter, 2019. 4 с.
23. Guido Albertengo, Fikru G. Debele, Waqar Hassan, Dario Stramandino, Digital Communications and Networks, 2020. 35 с.
24. Satoshi Masuda, Complex Software Testing Analysis using International Standards, 2013. 50 с.

ДОДАТКИ

Додаток А
(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА
РОЗРОБКА АВТОМАТИЗОВАНОГО МОДУЛЯ ОБМІНУ ПОВІДОМЛЕННЯМИ І ФАЙЛАМИ ЕЛЕКТРОННОЇ ОСВІТНЬОЇ СИСТЕМИ НА ОСНОВІ ПЛАТФОРМИ FIREBASE

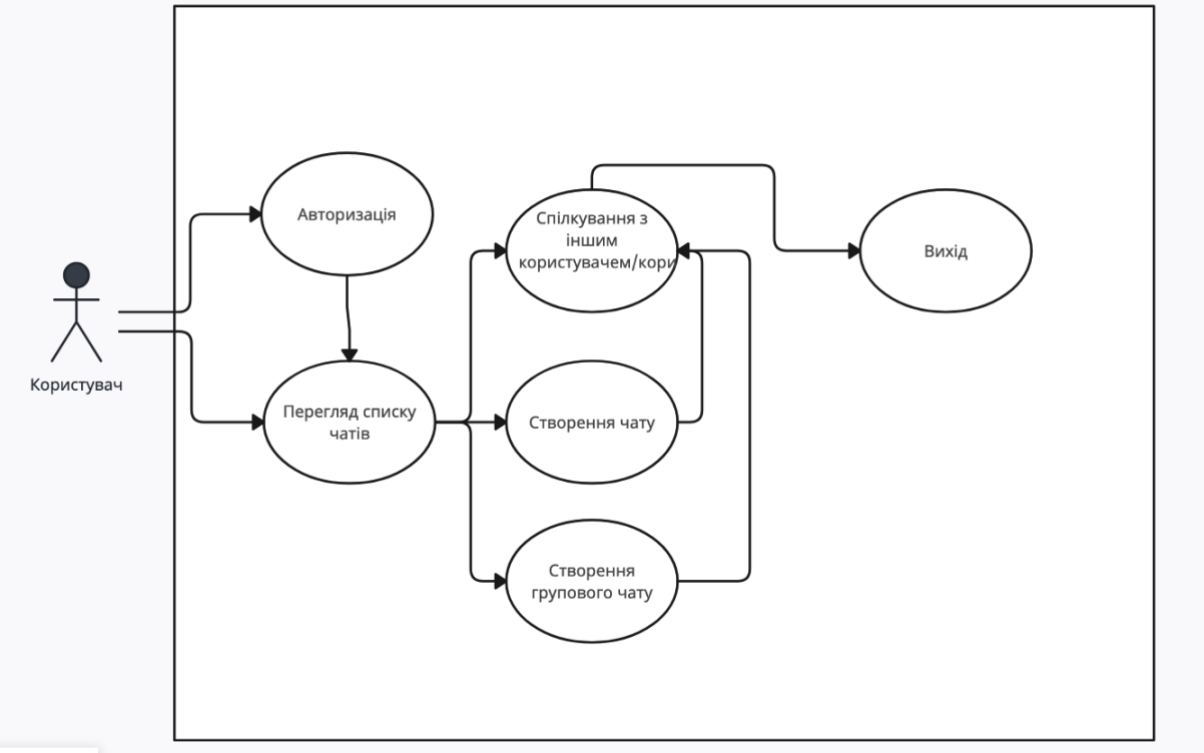

Рисунок А.1 – Use-case діаграма

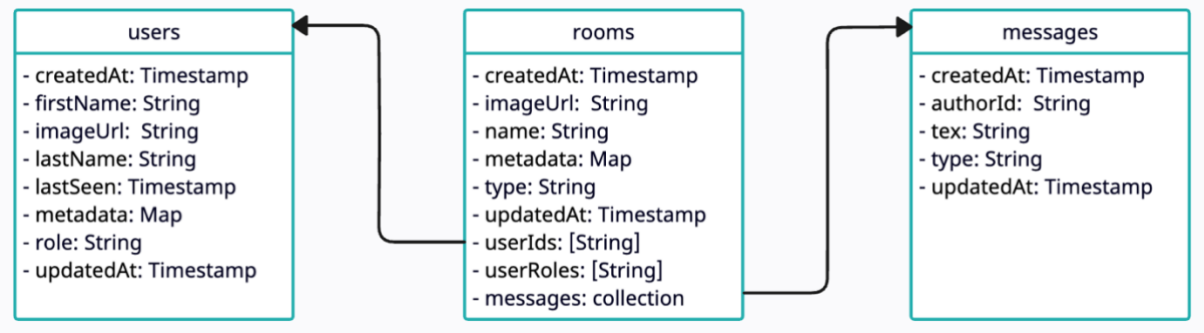

Рисунок А.2 – Результуюча ER-діаграма

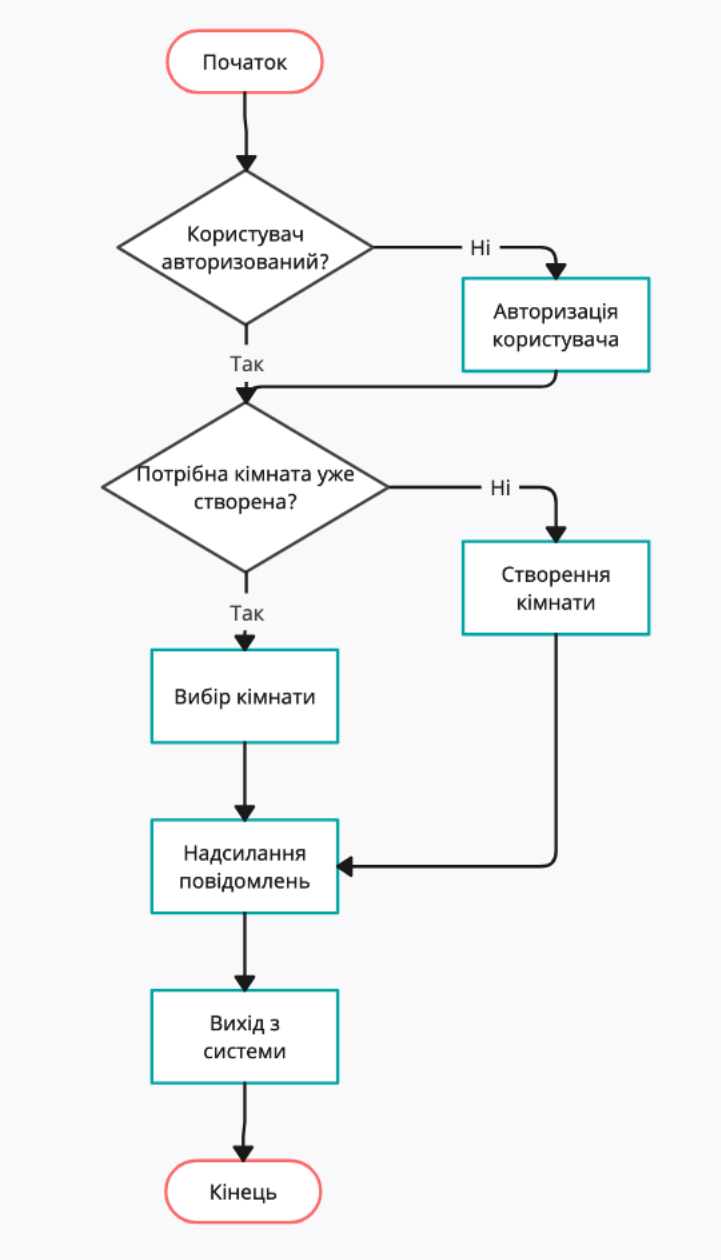

Рисунок А.3 – Схема алгоритму програми

Додаток Б
(обов’язковий)
Лістинг програми

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await initServiceLocator();

  await Firebase.initializeApp(
    name: '[DEFAULT]',
    options: DefaultFirebaseOptions.currentPlatform,
  );

  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (_) => injector<UserBloc>()
        ..add(
          const UserEvent.initCheck(),
        ),
      child: BlocListener<UserBloc, UserState>(
        listener: (_, state) {
          state.maybeWhen(
            authorized: (_, __) {
              appRouter.replaceAll(
                [
                  const HomeRoute(),
                ],
              );
            },
            unauthorized: () {
              appRouter.replaceAll(
                [
```

```
              const LoginRoute(),
            ],
          );
        },
        error: (_) {
          appRouter.replaceAll(
            [
              const LoginRoute(),
            ],
          );
        },
        orElse: () {
          return;
        },
      );
    },
    child: MaterialApp.router(
      debugShowCheckedModeBanner: false,
      title: 'Jet Iq Messager',
      routeInformationParser: appRouter.defaultRouteParser(),
      routerDelegate: AutoRouterDelegate(
        appRouter,
      ),
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(
          seedColor: const Color.fromARGB(255, 0, 162, 174),
        ),
        useMaterial3: true,
      ),
    ),
  ),
 );
 }
}


@RoutePage(name: 'HomeRoute')
class HomeScreen extends StatelessWidget {
 const HomeScreen({super.key});
```

```
@override
Widget build(BuildContext context) {
  final userBloc = injector<UserBloc>();

  return Scaffold(
    body: BlocBuilder<UserBloc, UserState>(
      builder: (_, __) {
        return LayoutBuilder(
          builder: (_, constraints) {
            if (constraints.maxWidth > 600) {
              return Row(
                children: [
                  SizedBox(
                    width: 300,
                    child: RoomsScreen(),
                  ),
                  userBloc.state.maybeWhen(
                    authorized: (_, selectedRoom) {
                      return Expanded(
                        //  flex: 3,
                        child: selectedRoom != null
                            ? ChatPage(room: selectedRoom)
                            : const Center(
                                child: Text('No selected room'),
                              ),
                      );
                    },
                    orElse: () {
                      return const Offstage();
                    },
                  ),
                ],
              );
            } else {
              return userBloc.state.maybeWhen<Widget>(
                authorized: (_, selectedRoom) {
                  return selectedRoom != null
                      ? ChatPage(room: selectedRoom)
                      : RoomsScreen();
                },
```

```
                orElse: () {
                  return const Offstage();
                },
              );
            }
          },
        );
      },
    ),
  );
 }
}


@RoutePage(name: 'LoginRoute')
class LoginScreen extends StatefulWidget {
 const LoginScreen({super.key});

 @override
 State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
 final _focusNode = FocusNode();

 final _passwordController = TextEditingController(text: '');
 var _usernameController = TextEditingController(text: '');

 @override
 Widget build(BuildContext context) {
  return BlocProvider(
    create: (_) => injector<LoginBloc>(),
    child: Scaffold(
      appBar: AppBar(
        systemOverlayStyle: SystemUiOverlayStyle.light,
        title: const Text('Login'),
      ),
      body: BlocBuilder<LoginBloc, LoginState>(
        builder: (contextWithBloc, state) {
          final bloc = BlocProvider.of<LoginBloc>(contextWithBloc);
```

```
return Center(
  child: Column(
    children: [
      LayoutBuilder(
        builder: (_, constraints) {
          final isBigScreen = constraints.maxWidth > 600;

          return SingleChildScrollView(
            child: Container(
              constraints: BoxConstraints(
                maxWidth: isBigScreen ? 500 : double.infinity,
              ),
              padding: const EdgeInsets.only(
                top: 200,
                left: 24,
                right: 24,
              ),
              child: Column(
                children: [
                  TypeAheadField<JetIqSearchUserModel>(
                    controller: _usernameController,
                    suggestionsCallback: (search) {
                      bloc.add(LoginEvent.searchUsers(search));
                      final searchedUsers = state.maybeWhen(
                        enteringLogin: (searchedUsers) =>
                            searchedUsers,
                        orElse: () => <JetIqSearchUserModel>[],
                      );

                      return searchedUsers;
                    },
                    builder: (_, __, focusNode) {
                      return TextField(
                        autocorrect: false,
                        autofocus: true,
                        focusNode: focusNode,
                        controller: _usernameController,
                        decoration: InputDecoration(
                          border: const OutlineInputBorder(
```

```
          borderRadius: BorderRadius.all(
            Radius.circular(8),
          ),
        ),
        labelText: 'Login',
        suffixIcon: IconButton(
          icon: const Icon(Icons.cancel),
          onPressed: _usernameController.clear,
        ),
      ),
      onEditingComplete: _focusNode.requestFocus,
      textInputAction: TextInputAction.next,
      onChanged: (value) {
        bloc.add(
          LoginEvent.searchUsers(value),
        );
      },
      onSubmitted: (value) {
        bloc.add(
          LoginEvent.selectUser(
            JetIqSearchUserModel(
              name: value,
              id: '',
            ),
          ),
        );
      },
    );
  },
  itemBuilder: (_, user) {
    return ListTile(
      title: Text(user.name),
    );
  },
  onSelected: (user) {
    _usernameController =
        TextEditingController(text: user.name);
    bloc.add(LoginEvent.selectUser(user));
  },
  emptyBuilder: (_) => const Offstage(),
```

```
),
Container(
  margin: const EdgeInsets.symmetric(vertical: 8),
  child: TextField(
    autocorrect: false,
    autofillHints: const [AutofillHints.password],
    controller: _passwordController,
    decoration: InputDecoration(
      border: const OutlineInputBorder(
        borderRadius: BorderRadius.all(
          Radius.circular(8),
        ),
      ),
      labelText: 'Password',
      suffixIcon: IconButton(
        icon: const Icon(Icons.cancel),
        onPressed: _passwordController.clear,
      ),
    ),
    focusNode: _focusNode,
    keyboardType: TextInputType.emailAddress,
    obscureText: true,
    onEditingComplete: () {
      injector<UserBloc>().add(
        UserEvent.login(
          login: _usernameController.text,
          password: _passwordController.text,
        ),
      );
    },
    textInputAction: TextInputAction.done,
  ),
),
const SizedBox(height: 40),
ElevatedButton(
  onPressed: () {
    injector<UserBloc>().add(
      UserEvent.login(
        login: _usernameController.text,
        password: _passwordController.text,
```

```
                            ),
                          );
                        },
                        child: const Text('Login'),
                      ),
                    ],
                  ),
                ),
              );
            },
          ),
        ],
      ),
    );
  }
}
```

Додаток В
(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка автоматизованого модуля обміну повідомленнями і файлами електронної освітньої системи на основі платформи Firebase»

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: кафедра АІІТ, ФІІТА, ЗАКІТ-20б
(кафедра, факультет, навчальна група)

Науковий керівник: Паламарчук Є.А., професор каф. АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності Unicheck

Оригінальність 98.99 % Схожість 1.01 %

Аналіз звіту подібності (відмітити потрібне)

■ 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 розгляд експертної комісії кафедри.

☐ 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 _____ Овчинников К. В.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи

Автор _____ Поліщук Я. В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Паламарчук Є.А.
(підпис) (прізвище, ініціали)

Додаток Г

Акт впровадження

ЗАТВЕРДЖУЮ
Директор центру диджиталізації ВНТУ
Тужанський С.Є.
_____
« » _____ 2024 року

АКТ ВПРОВАДЖЕННЯ № 1/ 12.06.2024
результатів розробки підсистеми обміну повідомленнями

Замовник Центр диджиьалізації Вінницького національного технічного університету -
(найменування організації)

Цим актом підтверджується, що результати роботи – «Розробка автоматизованого модуля обміну повідомленнями і файлами електронної освітньої системи на основі платформи Firebase»
(найменування теми)

що виконав студент гр. ЗАКІТ-20б, Поліщук Я. В. на громадських засадах
(виконавець)

відповідно до планів робіт Центру диджиталізації ВНТУ на 2020-2021 р., 2021-2022 р., а також на 2022-2023 р. та впроваджено у Вінницькому національному технічному університеті
(строки виконання) (найменування організації, де здійснювалося впровадження)

1. Вид впроваджених результатів: система "JetIQ Messenger"
(експлуатація виробу, роботи, технології)

2. Характеристика масштабу впровадження: одиничне
(унікальне, одиничне, партія, масове, серійне)

3. Форма впровадження: експериментальний веб-сервіс «JetIQ Messenger» який інтегрований з системою JetIQ VNTU

4. Новизна результатів науково-дослідної роботи: застосування хмарних сервісів для підсистеми спілкування учасників навчального процесу
(піонерські, принципово нові, якісно нові, модифікації, модернізація старих розробок)

5. Впроваджені: в освітньому середовищі системи JetIQ VNTU

6. Соціальний та науково-технічний ефект: покращення та пришвидшення спілкування в навчальному процесі .

(охорона навколишнього середовища, поліпшення й оздоровлення умов праці, удосконалення структури керування, науково-технічних напрямків, спеціальне призначення)

7. Ліцензійні застереження: Всі результати бакалаврської кваліфікаційної роботи можуть бути впроваджені у навчальний процес, наукові дослідження, науково-методичне забезпечення та ІТ-інфраструктуру Вінницького національного технічного університету на безоплатній основі без обмежень прав на використання, копіювання, редагування, доповнення, публікацію, поширення, субліцензування та/або продаж копій.

Програмний код розробки надається університету на безоплатній основі без обмежень прав на використання, копіювання, редагування, доповнення, публікацію, поширення, субліцензування та/або продаж копій за умовами ліцензії MIT:

"ЦЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НАДАЄТЬСЯ «ЯК Є», БЕЗ ГАРАНТІЙ БУДЬ-ЯКОГО ВИДУ, ПРЯМИХ АБО НЕПРЯМИХ, ВКЛЮЧАЮЧИ, АЛЕ НЕ ОБМЕЖУЮЧИСЬ, ГАРАНТІЯМИ КОМЕРЦІЙНОЇ ВИГОДИ, ВІДПОВІДНОСТІ ЙОГО КОНКРЕТНОМУ ПРИЗНАЧЕННЮ Й ВІДСУТНОСТІ ПОРУШЕННЯ ПРАВ. У ЖОДНОМУ РАЗІ АВТОРИ АБО ВЛАСНИКИ АВТОРСЬКИХ ПРАВ НЕ ВІДПОВІДАЮТЬ ЗА БУДЬ-ЯКИМИ СУДОВИМИ ПОЗОВАМИ, ЩОДО ЗБИТКІВ АБО ІНШИХ ПРЕТЕНЗІЙ, ЧИ ДІЙ ДОГОВОРУ, ЦИВІЛЬНОГО ПРАВОПОРУШЕННЯ АБО ІНШИХ, ЩО ВИНИКАЮТЬ ПОЗА, АБО У ЗВ'ЯЗКУ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ АБО ВИКОРИСТАННЯМ ЧИ ІНШИМИ ДІЯМИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ"

Повний текст ліцензії знаходиться за адресою: https://opensource.org/license/mit/

Від виконавця:
студент групи ЗАКІТ-20б

_____ Поліщук Я. В.

Від відділу дистанційної
освіти ВНТУ, керівник,
к.т.н., професор каф. АІІТ:

_____ Паламарчук Є.А.