

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ КРОСПЛАТФОРМНОГО МОБІЛЬНОГО
ЗАСТОСУНКУ ЧАТУВАННЯ З ОПТИМІЗАЦІЄЮ ОБМІНУ

МЕДІАКОНТЕНТОМ

SOFTWARE IMPLEMENTATION OF A CROSS-PLATFORM MOBILE CHAT
APPLICATION WITH OPTIMIZATION OF MEDIA EXCHANGE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-221

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ДАНІЛІН А.О.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ДАНІЛІНА Андрія Олексійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація кроссплатформного мобільного застосунку чатування з оптимізацією обміну медіаконтентом.

Software Implementation of a Cross-platform Mobile Chat Application with Optimization of Media Exchange

керівник проєкту (роботи) д-р.філос., доцент, ЛЕОЩЕНКО Сергій Дмитрович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЕОЩЕНКО С. Д., доцент		
Нормоконтроль	КАЛІНІНА М. В., асистент		

7. Дата видачі завдання «14» квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Андрій ДАНІЛІН
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій ЛЕОЩЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
88 с., 6 табл., 36 рис., 3 дод., 22 джерел.

КРОСПЛАТФОРМНИЙ МОБІЛЬНИЙ ЗАСТОСУНОК, ЧАТ, ОБМІН
МЕДІАКОНТЕНТОМ, СТИСНЕННЯ ЗОБРАЖЕНЬ, DART, FLUTTER,
FIREBASE.

Об'єкт дослідження – процес обміну медіаконтентом у мобільних чат-
додатках.

Предмет дослідження – методи оптимізації передавання зображень у
мобільних застосунках.

Мета роботи – розробка кросплатформного мобільного застосунку
чатування з можливістю оптимізованого обміну медіаконтентом.

Матеріали, методи та технічні засоби: ноутбук під керуванням Windows,
середовище розробки Visual Studio Code, мова програмування Dart, фреймворк
Flutter, база даних Firebase. Для тестування – мобільні пристрої на Android та
емулятор iOS.

Результати. Розроблено кросплатформний мобільний застосунок для
чатування, що дозволяє користувачам передавати зображення у стиснутому та
оригінальному форматах, що оптимізує використання трафіку. Також
реалізовано функції облікових записів користувачів, додавання та виключення з
чорного списку користувачів, відправка скарги на повідомлення, чорна та біла
тема додатку.

Висновки. Реалізовано ефективний механізм оптимізації обміну
медіаконтентом у чат-додатку. Проведені тести підтвердили, що алгоритми
стиснення дозволяють значно зменшити обсяг передаваних даних без суттєвої
втрати якості зображень.

Галузь використання – мобільні комунікаційні застосунки для обміну повідомленнями та мультимедійним контентом.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor's: 88 pages, 6 tables, 36 figures, 3 appendixes, 22 sources.

CROSS-PLATFORM MOBILE APPLICATION, CHAT, MEDIA CONTENT EXCHANGE, IMAGE COMPRESSION, DART, FLUTTER, FIREBASE.

The object of research is the process of media content exchange in mobile chat applications.

The subject of research is methods for optimizing image transmission in mobile applications.

Purpose - to develop a cross-platform mobile chat application with the ability to optimize the exchange of media content.

Materials, methods and equipment: a laptop running Windows, Visual Studio Code development environment, Dart programming language, Flutter framework, Firebase database. For testing: Android mobile devices and iOS emulator.

Results. A cross-platform mobile chat application has been developed that allows users to transfer images in compressed and original formats, which optimizes traffic usage. The functions of user accounts, adding and removing users from the blacklist, sending a complaint about a message, black and white themes of the application are also implemented.

Conclusions. We have implemented an effective mechanism for optimizing the exchange of media content in a chat application. Tests have confirmed that compression algorithms can significantly reduce the amount of transmitted data without significant loss of image quality.

Application area - mobile communication applications for messaging and multimedia content exchange.

ЗМІСТ

С.

Перелік скорочень та умовних позначок	9
Вступ.....	10
1 Аналіз предметної області.....	12
1.1 Огляд предметної області.....	12
1.2 Огляд сучасного стану предметної області.....	13
1.3 Необхідність програмних рішень	15
1.4 Порівняння існуючих аналогів	16
1.4.1 WhatsApp.....	16
1.4.2 Telegram	17
1.4.3 Signal	19
1.4.4 Порівняння функціональності	20
1.5 Висновки за розділом	21
2 Матеріали і методи.....	22
2.1 Вибір мови програмування	22
2.2 Вибір середовища розробки.....	25
2.3 Вибір платформи для бекенд-розробки	26
2.4 Висновки за розділом	28
3 Основні рішення щодо реалізації компонентів системи.....	30
3.1 Опис обраної архітектури системи	30
3.1.1 Використання патерну MVVM.....	31
3.2 Проєктування клієнтської частини	32
3.2.1 Основні принципи та підходи.....	33
3.2.2 Використані пакети та бібліотеки	34
3.2.3 Реалізація функції стиснення зображень та оптимізація медіаконтенту	34
3.3 Проєктування серверної частини	36
3.3.1 Основні складові серверної частини.....	36

	8
3.3.2 Основні робочі процеси серверної частини	37
3.3.3 Діаграми	37
3.4 Проєктування бази даних	39
3.4.1 Основні складові структури бази даних	39
3.4.2 Особливості проєктування бази даних	40
3.5 Проєктування інтерфейсу користувача	42
3.5.1 Ключові принципи дизайну	42
3.5.2 Основні екрани додатку	43
3.5.3 Навігаційна структура	44
3.6 Висновки за розділом	45
4 Експлуатація, тестування та експериментальне дослідження програми	47
4.1 Сценарії використання та призначення програми	47
4.2 Системні вимоги та умови експлуатації	48
4.2.1 Апаратні та програмні вимоги	48
4.2.2 Умови експлуатації	49
4.3 Демонстрація екранних рішень	49
4.4 Тестування функцій додатку	56
4.5 Оцінка зручності використання	59
4.6 Висновки за розділом	60
Висновки	62
Перелік джерел посилання	64
Додаток А Технічне завдання	66
Додаток Б Текст програми	70
Додаток В Слайди презентації	80

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;
IDE – Integrated Development Environment;
JPEG – Joint Photographic Experts Group;
MVVM – Model-View-ViewModel;
SDK – Software Development Kit;
UI – User Interface;
UX – User Experience;
WebP – WEB Pictures.

ВСТУП

Сучасна комунікація зазнала кардинальних змін завдяки стрімкому розвитку інформаційних технологій, мобільного зв'язку та повсюдній доступності Інтернету. Мобільні чат-застосунки стали одним із найпопулярніших способів взаємодії між людьми, забезпечуючи миттєвий обмін повідомленнями, зображеннями, відео та іншими типами медіаконтенту. Завдяки своїй зручності, безпеці та можливості персоналізації ці застосунки стали невід'ємною частиною повсякденного життя.

Проте із розширенням функціональності сучасних систем зросли вимоги до їх продуктивності та оптимізації, особливо в аспекті передачі медіафайлів. Зображення часто займають багато пам'яті та можуть спричиняти надмірне навантаження на мережу, уповільнюючи роботу застосунку та збільшуючи витрати мобільного трафіку. Саме тому актуальним є питання стиснення зображень перед їх передачею, що дозволяє суттєво зменшити обсяг даних без значної втрати якості.

Метою даної дипломної роботи є розробка кросплатформного мобільного чат-застосунку, який забезпечує ефективну передачу зображень як у стиснутому, так і в оригінальному форматі. Такий підхід дозволяє користувачам самостійно обирати оптимальний варіант залежно від своїх потреб – швидкості доставки, якості зображення чи економії мобільного трафіку.

У процесі реалізації проєкту було впроваджено функціонал авторизації користувачів, можливість додавання або видалення з чорного списку, подання скарг на повідомлення, а також підтримку світлої й темної тем оформлення інтерфейсу. Застосунок розроблено за допомогою сучасного фреймворку Flutter, що дозволяє створювати рішення для Android та iOS, а також із застосуванням хмарного сервісу Firebase для зберігання даних, автентифікації користувачів та виконання інших серверних функцій.

Робота зосереджена на ключових етапах проєктування, розробки та тестування мобільного застосунку з акцентом на оптимізацію обміну

медіаконтентом. Особлива увага приділяється ергономіці інтерфейсу, швидкодії застосунку та практичній доцільності запропонованих рішень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд предметної області

У сучасному цифровому середовищі мобільні застосунки для обміну повідомленнями відіграють ключову роль у забезпеченні швидкої та зручної комунікації між користувачами. Зростання популярності смартфонів та широке розповсюдження інтернету сприяли розвитку різноманітних месенджерів, які надають можливості не лише для текстового спілкування, але й для обміну зображеннями, аудіо та іншими файлами.

Однією з ключових вимог до сучасних мобільних застосунків є кросплатформність – здатність працювати на різних операційних системах, таких як Android та iOS. Це дозволяє розробникам охоплювати ширшу аудиторію та зменшувати витрати на створення окремих версій для кожної платформи. Серед популярних фреймворків для кросплатформної розробки виділяються Flutter та React Native.

Flutter, розроблений компанією Google, використовує мову програмування Dart і забезпечує високу продуктивність та гнучкість у створенні інтерфейсів користувача. Його особливістю є використання власного графічного рушія, що дозволяє досягти стабільної частоти кадрів та однакового вигляду інтерфейсу на різних платформах. Крім того, Flutter надає широкий набір віджетів, які відповідають як Material Design, так і Cupertino (iOS) стилям, що спрощує створення інтерфейсів для обох платформ [1].

React Native, створений Facebook, дозволяє використовувати JavaScript та має широку спільноту розробників. Він використовує нативні компоненти інтерфейсу користувача, що забезпечує вигляд та відчуття, близькі до нативних застосунків. React Native також підтримує "гаряче перезавантаження" (hot reloading), що прискорює процес розробки, дозволяючи миттєво бачити внесені зміни [1].

Окрім текстового спілкування, користувачі очікують можливості обміну медіаконтентом, зокрема зображеннями. Ефективне стиснення та передача

зображень є критично важливими для забезпечення швидкості та економії трафіку. Формат зображень WebP, розроблений Google, пропонує як втратне, так і безвтратне стиснення, дозволяючи зменшити розмір файлів без значної втрати якості. Згідно з дослідженнями, використання WebP може зменшити розмір зображень на 25–34% порівняно з JPEG при збереженні аналогічної якості [2].

Використання WebP дозволяє значно зменшити обсяг переданих даних, що особливо важливо для мобільних мереж з обмеженою пропускнуою здатністю. Крім того, зменшення розміру файлів сприяє швидшому завантаженню зображень та економії часу користувачів.

Таким чином, розробка кросплатформного мобільного застосунку для обміну повідомленнями вимагає врахування сучасних технологій та стандартів, зокрема вибору відповідного фреймворку та ефективних методів стиснення медіаконтенту. Застосування передових рішень у цих сферах дозволить створити продукт, який відповідатиме високим вимогам користувачів щодо функціональності, швидкодії та ефективності використання ресурсів.

1.2 Огляд сучасного стану предметної області

У 2020–2024 роках ринок мобільних застосунків для обміну повідомленнями демонструє стійке зростання. За даними аналітичної платформи Statista, щомісячна кількість активних користувачів месенджерів перевищує 3 мільярди осіб, і ця цифра щороку зростає [3]. Найпопулярнішими серед них залишаються WhatsApp, Facebook Messenger, Telegram, Signal та Viber, які надають широкий набір функцій: особисті й групові чати, обмін медіафайлами, голосові та відеодзвінки, реакції на повідомлення тощо.

Функціональність месенджерів постійно розширюється. В останні роки особливої уваги набули такі аспекти, як інтеграція з іншими сервісами (наприклад, хмарне зберігання, боти, платіжні системи), оптимізація передачі медіаконтенту та підтримка офлайн-режиму. Зокрема, Telegram застосовує власний протокол передачі даних MTProto та зберігає медіа на хмарних серверах

для швидшого доступу. У WhatsApp – зображення автоматично стискаються перед надсиланням, що дозволяє зменшити навантаження на мережу та пристрій.

Технічна реалізація сучасних месенджерів зазвичай базується на хмарній інфраструктурі з використанням служб типу Firebase, AWS або Google Cloud Platform. Firebase, зокрема, забезпечує розробникам готові рішення для реєстрації користувачів, зберігання повідомлень у режимі реального часу (Firebase Realtime Database або Firestore), хмарне сховище (Firebase Storage), а також механізм push-сповіщень [4].

Ще одним важливим напрямом розвитку є оптимізація передавання мультимедійного контенту. У більшості випадків мобільні месенджери стискають зображення перед надсиланням, щоби скоротити обсяг трафіку, зменшити час передачі та оптимізувати досвід користувача. Найефективнішими форматами для цього є WebP та HEIF (для iOS), які підтримуються нативно багатьма сучасними платформами.

У межах UX-дизайну спостерігається тенденція до спрощення інтерфейсів, акценту на мінімалізмі та доступності. Важливою складовою стає адаптивність: інтерфейс має однаково добре відображатися на різних типах пристроїв (від смартфонів до планшетів) та підтримувати темну/світлу тему.

Також варто відзначити поступове зростання популярності кросплатформних рішень, що спрощують процес розробки та дозволяють швидко запускати оновлення. Flutter, наприклад, з 2021 року активно інтегрується у великі проєкти, включаючи месенджери з підтримкою мультимедіа. Це пов'язано як із високою продуктивністю, так і з можливістю створення єдиного UI-коду для обох основних мобільних ОС [5].

Таким чином, сучасний стан предметної області характеризується високою конкуренцією серед мобільних застосунків, гнучкістю у впровадженні нових технологій, а також зростаючим попитом на рішення, здатні забезпечити швидкий, надійний та оптимізований обмін повідомленнями й медіа між користувачами.

1.3 Необхідність програмних рішень

У сучасному цифровому середовищі мобільні застосунки для обміну повідомленнями відіграють ключову роль у забезпеченні швидкої та ефективної комунікації. Зростаюча популярність смартфонів та доступність інтернету сприяють збільшенню кількості користувачів месенджерів, що підвищує вимоги до їх функціональності та продуктивності.

Покращення обміну медіаконтентом між платформами: Впровадження протоколу Rich Communication Services (RCS) в iOS 18 значно поліпшило якість обміну повідомленнями між пристроями на базі iOS та Android. Це оновлення забезпечує передачу високоякісних зображень та відео, що усуває попередні проблеми з низькою роздільною здатністю медіафайлів при міжплатформенній комунікації [6].

Інтеграція різних платформ обміну повідомленнями: Відсутність єдиних стандартів для обміну повідомленнями між різними платформами створює бар'єри для користувачів. Спроби сторонніх розробників, таких як Weeger, забезпечити сумісність між iMessage та Android стикаються з технічними та юридичними перешкодами [7].

Використання штучного інтелекту для персоналізації комунікації: Сучасні тенденції включають інтеграцію AI та ML для персоналізації повідомлень та покращення взаємодії з користувачем. Наприклад, платформа MoEngage використовує AI для прогнозування оптимального часу та змісту повідомлень, що підвищує ефективність комунікації [8].

З огляду на вищезазначене, існує висока потреба в розробці нових програмних рішень, які забезпечать високоякісний обмін медіаконтентом між різними платформами, інтеграцію різних сервісів обміну повідомленнями та використання передових технологій для покращення користувацького досвіду.

1.4 Порівняння існуючих аналогів

У сфері мобільних застосунків для обміну повідомленнями існує кілька популярних платформ, які забезпечують користувачів різноманітними функціями для комунікації. Розглянемо три основні з них: WhatsApp, Telegram та Signal.

1.4.1 WhatsApp

WhatsApp залишається одним із найпопулярніших месенджерів у світі, пропонуючи широкий спектр функцій для користувачів.

Приклад інтерфейсу на рисунку 1.1 [9].

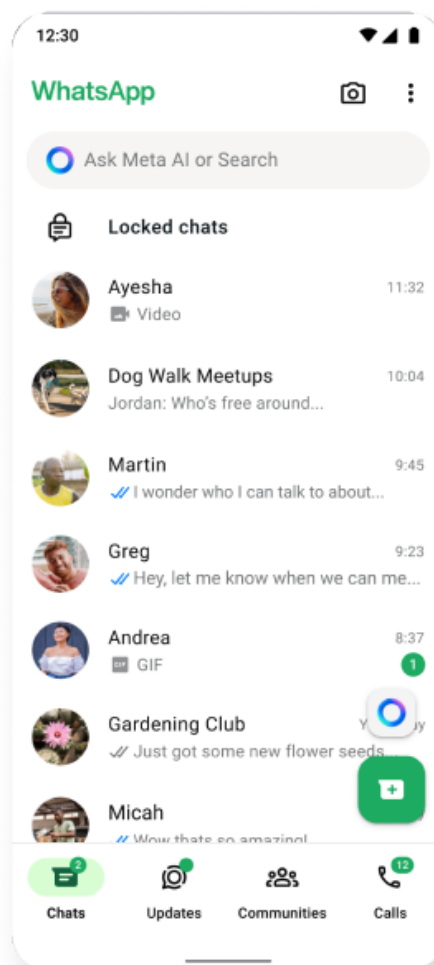


Рисунок 1.1 – Інтерфейс WhatsApp [9]

У 2025 році додано кілька нових можливостей:

- AI Studio для чатів: дозволяє взаємодіяти з AI-персонами, включаючи культурних ікон та відомих персонажів, що робить спілкування більш інтерактивним [10];
- вбудована цифрова клавіатура для користувачів iPhone: надає можливість здійснювати дзвінки без необхідності збереження номерів у контактах [10];
- швидкі реакції емодзі подвійним натисканням: спрощує вираження емоцій у чатах [10];
- користувацькі стікери та легке їх поширення: дозволяє створювати персоналізовані стікери з власних фотографій та ділитися ними через посилання [10];
- фонові ефекти для фотографій: дає змогу додавати фільтри та ефекти до зображень перед їх відправленням [10];
- персоналізовані повідомлення при пересиланні: дозволяє додавати власний текст до пересилаємих повідомлень, медіа чи посилань [10].

1.4.2 Telegram

Telegram відомий своєю гнучкістю та широкими можливостями для організації групових чатів, що робить його популярним вибором для спільнот, бізнесу та освітніх проєктів.

Приклад інтерфейсу на рисунку 1.2 [11].

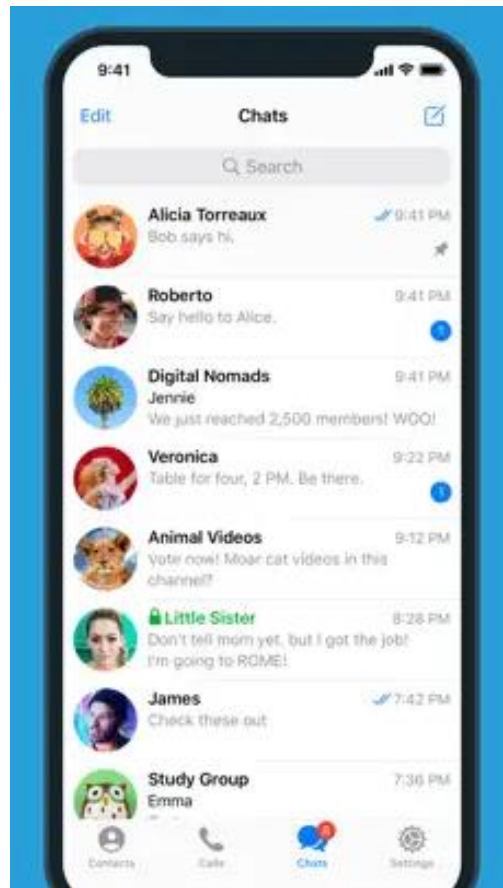


Рисунок 1.2 – Інтерфейс Telegram [11]

Серед основних функцій:

- підтримка групових чатів до 200 000 учасників: забезпечує можливість створення великих спільнот [12];
- канали для трансляції повідомлень необмеженій аудиторії: дозволяє адміністраторам передавати інформацію великій кількості підписників [12];
- редагування та видалення повідомлень після відправлення: корисно для коригування помилок або оновлення інформації [12];
- функція "Таємні чати" з наскрізним шифруванням: забезпечує підвищену конфіденційність для окремих розмов [12];
- можливість відправлення файлів розміром до 2 Гб: спрощує обмін великими медіафайлами та документами [12].

1.4.3 Signal

Signal відзначається високим рівнем безпеки та конфіденційності, що робить його вибором для користувачів, які цінують приватність.

Приклад інтерфейсу на рисунку 1.3 [13].

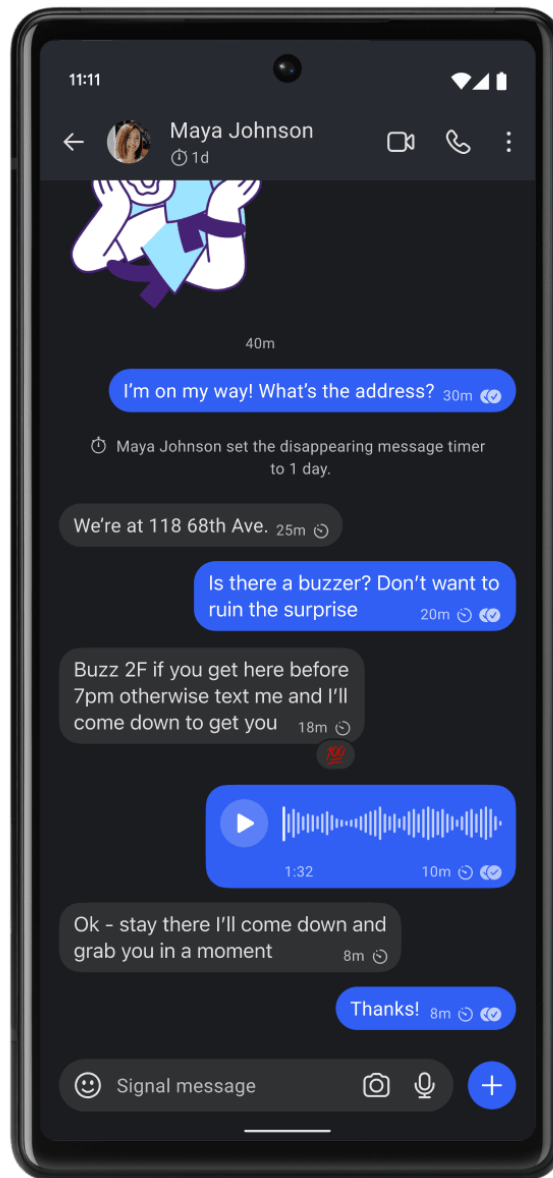


Рисунок 1.3 – Інтерфейс Signal [13]

Основні функції включають:

– автоматичне наскрізне шифрування всіх повідомлень та дзвінків: гарантує, що лише відправник та отримувач можуть переглядати зміст комунікації [14];

- підтримка групових чатів до 1 000 учасників: дозволяє створювати великі захищені спільноти [14];
- функція зникнення повідомлень: дозволяє встановлювати таймер для автоматичного видалення повідомлень після певного часу [14];
- відсутність реклами та збору даних користувачів: Signal фінансується за рахунок донатів, що дозволяє уникнути комерційного використання даних [14];
- відкритий вихідний код: забезпечує прозорість та можливість незалежного аудиту безпеки [14].

1.4.4 Порівняння функціональності

Нижче наведено порівняльну таблицю 1.1 основних функцій цих трьох месенджерів:

Таблиця 1.1 – Порівняння додатків аналогів

Функція	WhatsApp	Telegram	Signal
Наскрізне шифрування	Так	Лише в "Таємних чатах"	Так
Максимальна кількість учасників у групі	До 1 024 учасників	До 200 000 учасників	До 1 000 учасників
Підтримка каналів	Ні	Так	Ні
Редагування повідомлень	Ні	Так	Ні
Відправлення великих файлів	До 2 Гб	До 2 Гб	Обмежено
Функція зникнення повідомлень	Так	Так	Так

Продовження таблиці 1.1

Функція	WhatsApp	Telegram	Signal
Персоналізація стікерів	Так	Так	Ні

Порівнюючи функціональні можливості WhatsApp, Telegram та Signal, можна відзначити, що всі вони успішно задовольняють базові вимоги сучасних месенджерів – від забезпечення обміну повідомленнями і файлів до реалізації функцій зникнення повідомлень. При цьому WhatsApp і Signal відрізняються високим рівнем безпеки завдяки наскрізному шифруванню, тоді як Telegram вирізняється розширеними можливостями для роботи з груповими чатами, каналами та редагуванням повідомлень.

1.5 Висновки за розділом

У результаті аналізу було визначено основні тенденції розвитку мобільних застосунків для обміну повідомленнями. Серед ключових вимог – кросплатформність, підтримка медіаобміну та швидкодія. Такі фреймворки, як Flutter та React Native, дозволяють ефективно створювати застосунки для Android та iOS з єдиною кодовою базою.

Розгляд аналогів (WhatsApp, Telegram, Signal) показав, що на ринку вже існують потужні рішення, однак кожне з них має свої сильні сторони та обмеження. Це відкриває простір для розробки нових застосунків з акцентом на оптимізацію роботи з медіафайлами та зручність інтерфейсу.

Таким чином, актуальним залишається створення кросплатформного застосунку чатування, що поєднує базову функціональність месенджера з оптимізованою передачею зображень.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

У процесі розробки кросплатформного мобільного застосунку для обміну повідомленнями та медіаконтентом критично важливим є вибір відповідної мови програмування. Основними кандидатами для розгляду є Dart, JavaScript та Kotlin. Для об'єктивного вибору проведемо порівняння цих мов за наступними критеріями:

- продуктивність: оцінка швидкості виконання коду та ефективності використання ресурсів;
- читабельність та синтаксис: наскільки зрозумілий та зручний для написання та підтримки код;
- легкість або швидкість навчання: час та зусилля, необхідні для опанування мови;
- підтримка спільноти та екосистема: наявність активної спільноти розробників та доступність бібліотек і фреймворків;
- сумісність та інтеграція: можливість взаємодії з іншими мовами та платформами;
- кросплатформеність: підтримка розробки для різних платформ з використанням єдиного коду;
- наявність інструментів та середовищ розробки: розвиненість та зручність використання інструментів для розробки та налагодження.

Порівняння наведено в таблиці 2.1 [15]-[16].

Таблиця 2.1 – Порівняння мов програмування [15]-[16]

Критерій	Dart	JavaScript	Kotlin
Продуктивність	Висока: компіляція у машинний код забезпечує швидке виконання	Середня: інтерпретована мова, залежить від середовища виконання	Висока: компіляція у байт-код JVM забезпечує швидке виконання
Читабельність та синтаксис	Чистий та зрозумілий синтаксис, схожий на C- подібні мови	Гнучкий, але може бути неуніфікованим через динамічну природу	Консистентний та лаконічний синтаксис, схожий на Java
Легкість або швидкість навчання	Легкий для вивчення, особливо для тих, хто знайомий з JavaScript	Легкий для початку, але складний для глибокого опанування через особливості	Помірна складність, особливо для тих, хто знайомий з Java
Підтримка спільноти та екосистема	Зростаюча спільнота, активний розвиток Flutter	Велика спільнота, багата екосистема бібліотек та фреймворків	Широка спільнота, особливо в Android- розробці, підтримка JetBrains
Сумісність та інтеграція	Добра інтеграція з Flutter, але обмежена з іншими платформами	Висока сумісність з вебтехнологіями та серверними платформами	Відмінна інтеграція з Java та Android, підтримка мультиплатформенн ості

Продовження таблиці 2.1

Критерій	Dart	JavaScript	Kotlin
Крос-платформеність	Підтримує мобільні, веб та десктопні платформи через Flutter	Основна сфера – веб, обмежена підтримка мобільних платформ через фреймворки	Підтримує Android та iOS через Kotlin Multiplatform
Наявність інструментів та середовищ розробки	Добра підтримка в Visual Studio Code та Android Studio	Широкий вибір інструментів, включаючи Visual Studio Code, WebStorm	Відмінна інтеграція з Android Studio та IntelliJ IDEA

Після детального аналізу, для розробки кросплатформного мобільного застосунку було обрано мову програмування Dart у поєднанні з фреймворком Flutter. Це рішення обумовлене можливістю використовувати єдину кодову базу для Android та iOS, що значно зменшує час та витрати на розробку. Крім того, Flutter забезпечує високу продуктивність додатків завдяки компіляції в нативний код та наявності власного графічного движка [17].

У порівнянні з іншими мовами, такими як JavaScript та Kotlin, Dart пропонує більш інтегроване та оптимізоване середовище для кросплатформної розробки. JavaScript, хоча й широко використовується у веброботі, може мати обмеження щодо продуктивності та інтеграції з мобільними платформами. Kotlin є відмінним вибором для нативної розробки на Android, але для досягнення кросплатформності за допомогою Kotlin Multiplatform може знадобитися більше часу та ресурсів.

Зважаючи на ці фактори, обрання Dart з Flutter є найбільш оптимальним рішенням, забезпечуючи ефективну та економічну розробку кросплатформного мобільного застосунку.

2.2 Вибір середовища розробки

При виборі середовища розробки (IDE) для створення кросплатформного мобільного застосунку було проведено порівняння трьох популярних середовищ: Visual Studio Code (VS Code), Android Studio та IntelliJ IDEA. Порівняння виконувалося за наступними критеріями:

- ресурсоемність: вимоги до системних ресурсів (оперативної пам'яті, процесора);
- підтримка Flutter: наявність вбудованих інструментів або можливість додавання через розширення для роботи з фреймворком Flutter;
- продуктивність: швидкодія середовища при роботі з великими проєктами;
- налагодження коду: наявність та зручність інструментів для налагодження програм;
- вартість: ціна ліцензії або доступність безкоштовної версії;
- підтримка плагінів: можливість розширення функціональності через додаткові плагіни.

Ці критерії були обрані тому, що вони безпосередньо впливають на ефективність та зручність процесу розробки, а також на кінцеву якість та продуктивність мобільного застосунку.

Порівняння наведено в таблиці 2.2.

Таблиця 2.2 – Порівняння середовищ розробки

Критерій	Visual Studio Code	Android Studio	IntelliJ IDEA
Ресурсоемність	Низька	Висока	Висока
Підтримка Flutter	Через розширення	Є	Є
Продуктивність	Висока	Середня	Середня
Налагодження	Вбудоване	Розширене	Розширене

Продовження таблиці 2.2

Критерій	Visual Studio Code	Android Studio	IntelliJ IDEA
Вартість	Безкоштовне	Безкоштовне	Платна
Плагіни	Широка підтримка	Обмежена	Розширена

Після детального аналізу та порівняння, Visual Studio Code було обрано як оптимальне середовище розробки для проєкту. Його низька ресурсоемність, висока продуктивність, можливість розширення через плагіни та підтримка Flutter роблять його найбільш підходящим для ефективної та зручної розробки кросплатформного мобільного застосунку.

2.3 Вибір платформи для бекенд-розробки

Для реалізації бекенд-частини кросплатформного мобільного застосунку було обрано платформу Firebase. Firebase – це Backend-as-a-Service (BaaS) від компанії Google, що надає широкий спектр інструментів для розробки, зокрема: базу даних реального часу, автентифікацію, хмарні функції та аналітику [18].

Вибір платформи для бекенд-розробки базувався на наступних критеріях:

- функціональність: наявність необхідних сервісів для розробки та підтримки застосунку;
- сумісність із Flutter/Dart: легкість інтеграції та наявність офіційних або спільнотних пакетів для взаємодії з Flutter та Dart;
- простота інтеграції: легкість впровадження та сумісність з обраною технологією фронтенду;
- вартість: фінансова доступність та гнучкість тарифних планів;
- спільнота та підтримка: наявність активної спільноти розробників та офіційної підтримки.

Ці критерії були обрані з метою забезпечення ефективної розробки та стабільної роботи застосунку.

Порівняння наведено в таблиці 2.3 [19]-[20].

Таблиця 2.3 – Порівняння платформ для бекенд-розробки [19]-[20]

Критерій	Firebase	AWS Amplify	Supabase
Функціональність	Широкий спектр BaaS-сервісів, включаючи базу даних реального часу, автентифікацію та хмарні функції.	Надає інструменти для розробки та розгортання бекенду з інтеграцією AWS-сервісів.	Open-source альтернатива Firebase з підтримкою PostgreSQL та реального часу.
Сумісність із Flutter/Dart	Офіційні FlutterFire плагіни забезпечують просту та надійну інтеграцію з Flutter.	Підтримка Flutter через Amplify Flutter; потребує додаткового налаштування для повної сумісності.	Плагін supabase_flutter надає засоби для інтеграції з Flutter.
Простота інтеграції	Легка інтеграція з мобільними та вебзастосунками.	Потребує налаштування AWS-сервісів, що може ускладнити інтеграцію.	Простий RESTful API для інтеграції.
Вартість	Безкоштовний тариф з обмеженнями; платні плани при зростанні використання.	Платформа з оплатою за використання AWS-ресурсів.	Відкрите ПЗ; витрати на хостинг та обслуговування бази даних.

Продовження таблиці 2.3

Критерій	Firebase	AWS Amplify	Supabase
Спільнота та підтримка	Велика спільнота та офіційна підтримка від Google.	Активна спільнота AWS та офіційна підтримка.	Зростаюча спільнота, підтримка через open-source канали.

Після детального аналізу було обрано платформу Firebase для бекенд-розробки. Її переваги включають широкий набір інструментів, простоту інтеграції з мобільними застосунками та автоматичне масштабування. Крім того, оскільки Firebase, Dart та Flutter є продуктами компанії Google, їхня взаємна інтеграція є більш оптимізованою та стабільною, що сприяє зменшенню потенційних проблем сумісності та спрощує процес розробки. Хоча існують альтернативи, такі як AWS Amplify та Supabase, Firebase забезпечує оптимальний баланс між функціональністю, вартістю та підтримкою, що робить її найбільш підходящою для даного проєкту.

2.4 Висновки за розділом

У цьому розділі було детально обґрунтовано вибір основних технологій та інструментів для розробки кросплатформного мобільного застосунку. Зокрема, було обрано мову програмування Dart у поєднанні з фреймворком Flutter, що забезпечує ефективну розробку та підтримку єдиної кодової бази для платформ Android та iOS.

Для середовища розробки було обрано Visual Studio Code за його легкість, розширюваність та продуктивність. Серед альтернатив розглядалися Android Studio та IntelliJ IDEA, але Visual Studio Code виявився оптимальним вибором з огляду на потреби проєкту.

Щодо бекенд-частини, після порівняння платформ Firebase, AWS Amplify та Supabase, було обрано Firebase. Цей вибір обумовлений його широким спектром інструментів, простотою інтеграції з Flutter та стабільною підтримкою від компанії Google. Крім того, спільне походження Firebase, Dart та Flutter від Google сприяє їхній оптимальній взаємодії та стабільності роботи.

Таким чином, вибрані технології та інструменти забезпечують ефективну, економічно вигідну та зручну розробку кросплатформного мобільного застосунку, відповідаючи всім вимогам та цілям проєкту.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Опис обраної архітектури системи

Система розроблена за принципами кросплатформеної клієнт-серверної архітектури із використанням хмарних сервісів (Firebase) для обробки аутентифікації, зберігання даних та медіа-файлів. Загальна структура системи поділена на кілька логічних шарів.

Клієнтський шар. Цей шар відповідає за графічний інтерфейс користувача, який реалізовано з використанням Flutter. Він містить набір віджетів, що відповідають за основні функції взаємодії:

- відображення UI: реалізовано через екрани для аутентифікації, чатів, перегляду контактів та налаштувань;

- управління станом: реактивне оновлення даних забезпечується за допомогою провайдера стану (Provider), що дозволяє автоматично синхронізувати вміст між різними екранами.

Сервісний шар (Business Logic). У цьому шарі об'єднано бізнес-логіку додатку, таку як аутентифікація користувачів, обмін повідомленнями, обробка медіаконтенту, а також функції модерації (блокування, звітність). Завдяки ізоляції бізнес-логіки від представлення ця частина системи стає більш чистою, масштабованою та легко тестованою.

Бекенд (Firebase). Використовується Cloud Firestore для зберігання даних у реальному часі, Firebase Authentication – для безпечного входу користувачів, а також Firebase Storage – для зберігання медіа-файлів. Цей підхід дозволяє делегувати більшість серверних операцій хмарним сервісам, що зменшує витрати на розробку власної серверної частини і сприяє масштабованості системи.

На рисунку 3.1 наведено компонентну діаграму системи.

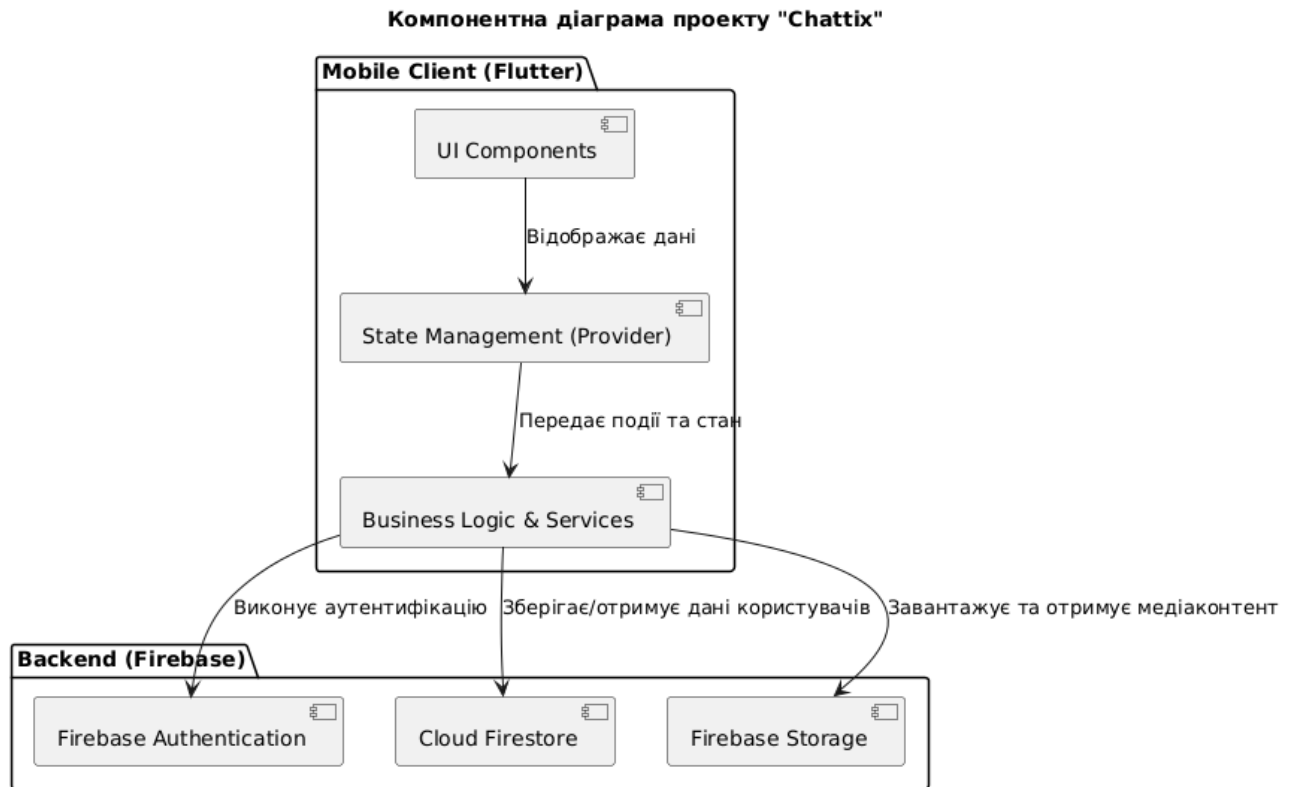


Рисунок 3.1 – Компонентна діаграма системи

3.1.1 Використання патерну MVVM

Обрана архітектура також спирається на принципи патерну Model-View-ViewModel (MVVM), що дозволяє чітко розділити відповідальність між різними компонентами додатку.

MVVM – шаблон проєктування архітектури додатку. В його основі відділення коду призначеного для користувача інтерфейсу (UI) від решти, і створення слабкого зв'язку для синхронізації [21].

Прошарок ViewModel забезпечує взаємодію між базою даних програми та інтерфейсом, але механізм прив'язки реалізований так, щоб окремі блоки в різних частинах коду, можна було міняти не впливаючи на інші. Без жорстких залежностей розробка, масштабування і підтримка додатків проходять набагато простіше [21].

Основні компоненти MVVM:

- Model: база даних і основна логіка додатку для роботи з нею, обробки запитів, проведення обчислень і т.д. [21];

- View: зображення – призначений для користувача інтерфейс, всі візуальні елементи з якими буде взаємодіяти користувач [21];

- ViewModel: посередництво між Model та View. Завдяки механізму реактивного оновлення ViewModel забезпечує передачу даних із Model до View, а також обробляє взаємодію користувача, не дозволяючи безпосередньо впливати на дані. Це дозволяє ізолювати бізнес-логіку від змін UI і полегшує тестування окремих модулів додатку.

На рисунку 3.2 можна побачити діаграму архітектури MVVM.

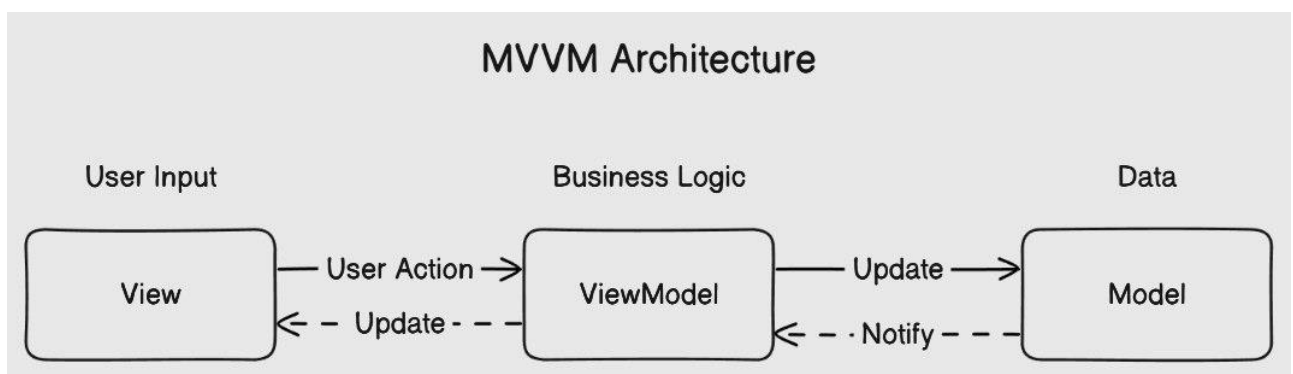


Рисунок 3.2 – Діаграма архітектури MVVM

Ця діаграма демонструє, як ViewModel слугує посередником між View та Model, забезпечуючи двосторонній зв'язок та реактивне оновлення інтерфейсу користувача.

3.2 Проєктування клієнтської частини

Клієнтська частина застосунку розроблена за допомогою Flutter, що дозволяє створити єдиний код для різних платформ – Android, iOS. Основною метою проєктування є забезпечення модульності, зручності використання та високої реактивності інтерфейсу.

3.2.1 Основні принципи та підходи

Для забезпечення ефективної роботи клієнтської частини застосунку було визначено ключові принципи розробки, які лягли в основу всієї системи. Їх реалізація дозволила створити зручне, стабільне та масштабоване рішення, здатне адаптуватися до зростаючих потреб користувачів.

- Модульність і ізоляція компонентів: інтерфейс застосунку розбитий на окремі блоки (екрани аутентифікації, головний екран, чат, налаштування). Кожен екран реалізовано як незалежний набір віджетів, що дозволяє легко змінювати окремі модулі без впливу на загальну архітектуру. Такий підхід сприяє перевикористанню компонентів та спрощує тестування;

- реактивне управління станом: використання бібліотеки Provider (разом із механізмом ChangeNotifier) дозволяє організувати динамічне оновлення інтерфейсу. Зміни даних миттєво відображаються на екрані. Завдяки цьому клієнтська частина забезпечує плавний та приємний досвід для користувача;

- відокремлення логіки від представлення: логіка взаємодії з серверною частиною та обробки даних ізольована від UI-компонентів. Розділення задач дозволяє зосередитися на чистоті оформлення інтерфейсу, залишаючи всю бізнес-логіку сервісному шару. Це спрощує підтримку коду і сприяє його масштабованості;

- адаптивність інтерфейсу: проєктування враховує різні розміри та орієнтації екранів. Використання адаптивних віджетів гарантує, що дизайн пристосовується до будь-якого пристрою. Це є критично важливим для кросплатформених застосунків, оскільки користувачі можуть використовувати застосунок на різних типах пристроїв.

3.2.2 Використані пакети та бібліотеки

У процесі розробки клієнтської частини застосовуються наступні ключові пакети, що дозволяють підвищити продуктивність та ефективність роботи застосунку:

- Flutter SDK: основний інструментарій для розробки користувацького інтерфейсу;
- Provider: бібліотека для управління станом, що дозволяє зручно організувати передачу даних між компонентами;
- Cached Network Image та Photo View: використовуються для завантаження та відображення зображень з мережі, що є важливим у частині обміну медіаконтентом.

3.2.3 Реалізація функції стиснення зображень та оптимізація медіаконтенту

У клієнтській частині застосунку я зробив можливість стиснення зображень, щоб оптимізувати їх передачу і зробити завантаження швидшим. Ідея полягає в тому, щоб перетворити зображення у формат JPEG з меншою якістю, що дозволяє значно зменшити розмір файлу і, отже, скоротити витрати мобільних даних. Це було здійснено за кількома простими етапами:

- вибір зображення: за допомогою пакету `image_picker` користувач може обрати зображення з галереї пристрою;
- декодування і стиснення: обране зображення завантажується у вигляді файлу, декодується за допомогою пакету `image`, після чого здійснюється стиснення за допомогою функції `img.encodeJpg()` з параметром якості, який встановлено на 30. Отримане стиснене зображення використовується для попереднього перегляду в чаті;
- збереження та передача: функціонал повертає об'єкт, що містить два варіанти: стиснуту версію зображення та оригінальний файл, що може бути використаний за потреби.

Хоча формат WebP має кращі показники стиснення та дозволяє отримати менший розмір файлу при збереженні високої якості, бібліотека `image`, яка використовується для обробки зображень, підтримує WebP лише для читання (read-only). Це означає, що неможливо записати стиснуте зображення у форматі WebP за допомогою даної бібліотеки. Тому для реалізації функції стиснення було прийняте рішення обрати формат JPEG. JPEG є загальноприйнятим стандартом і підтримується практично всіма пристроями та платформами, що гарантує сумісність і надійність роботи застосунку. Крім того, можливість налаштування рівня якості дозволяє знайти оптимальний баланс між якістю зображення та його розміром.

Нижче наведено блок-схему, що ілюструє послідовність дій при виборі, обробці та передачі зображення.

На рисунку 3.3 зображено блок-схему послідовності дій при відправці медіа повідомлення.

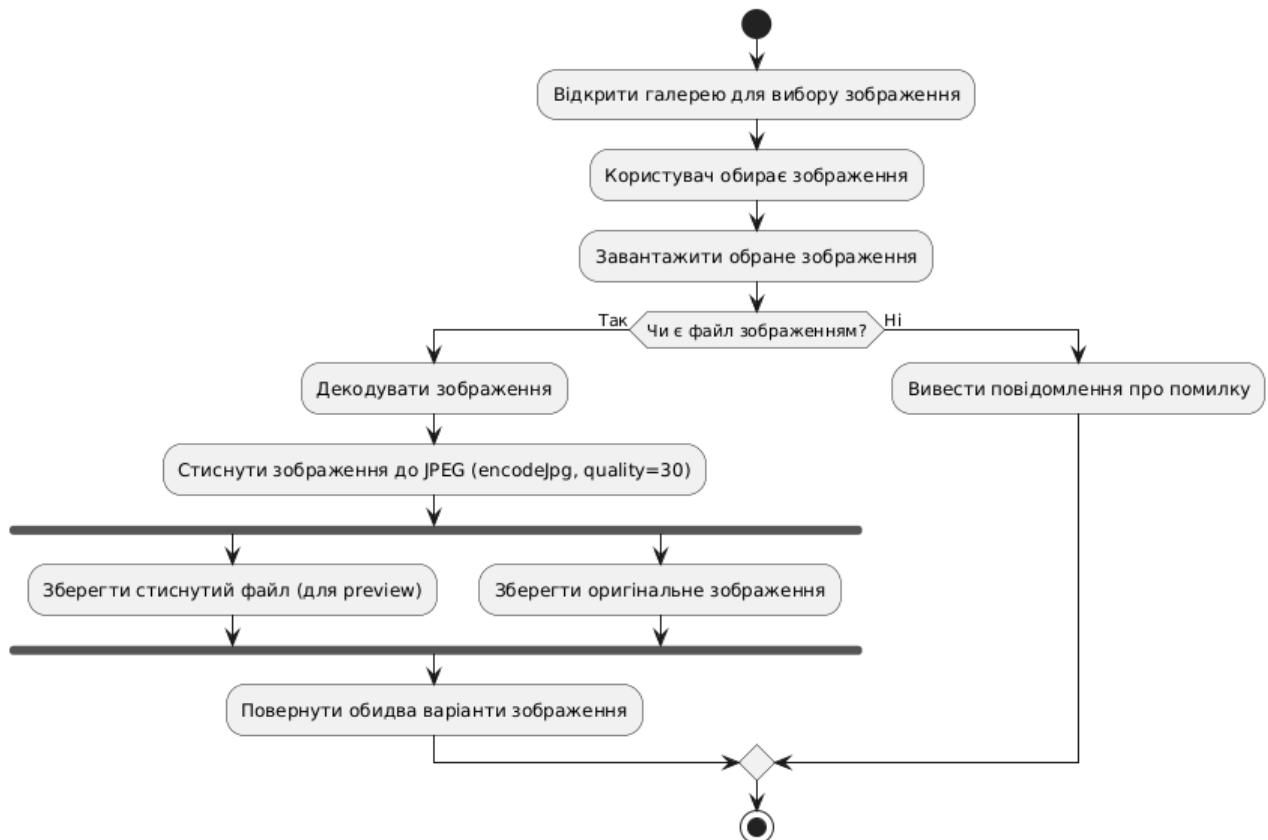


Рисунок 3.3 – Блок-схема послідовності дій при віправці медіа повідомлення

3.3 Проєктування серверної частини

У даному проєкті серверна частина реалізована за принципом «Backend-as-a-Service» із застосуванням платформи Firebase. Цей підхід дозволяє керувати важливими серверними функціями хмарним сервісам, що забезпечує масштабованість, високий рівень безпеки та зручність інтеграції даних у режимі реального часу. Основні сервіси, що використовуються для реалізації серверної логіки, – це Firebase Authentication, Cloud Firestore та Firebase Storage.

3.3.1 Основні складові серверної частини

Серверна частина програми побудована на основі ключових компонентів, які забезпечують її стабільну, безпечну та ефективну роботу. Ці елементи відповідають за автентифікацію користувачів, збереження та обробку даних у реальному часі, а також керування медіафайлами, що надсилаються в чатах.

Злагоджена взаємодія між цими сервісами дозволяє системі працювати швидко й надійно, гарантуючи плавний і зручний досвід для користувача, незалежно від платформи чи пристрою.

- Firebase Authentication: забезпечує безпечну аутентифікацію користувачів через методи входу та реєстрації. Після успішного проходження автентифікації дані користувача записуються в Cloud Firestore для подальшого використання в системі;

- Cloud Firestore: використовується як основне сховище для даних у реальному часі. У Firestore організовано колекції, такі як профілі користувачів, чат-кімнати, повідомлення, звіти про порушення та списки заблокованих користувачів. Дані оновлюються за допомогою потоків даних, що дозволяє миттєво реагувати на зміни;

- Firebase Storage: служить для зберігання медіаконтенту, наприклад, зображень, що надсилаються у чатах. Перед завантаженням файлів відбувається

їх стиснення, що зменшує обсяг передаваних даних та забезпечує швидке завантаження.

3.3.2 Основні робочі процеси серверної частини

Взаємодія між клієнтською та серверною частинами організована через асинхронні виклики та реактивні потоки. Наприклад:

- аутентифікація користувача: коли користувач вводить свої дані, клієнт надсилає запит до Firebase Authentication для перевірки – після успішного входу його дані фіксуються в Cloud Firestore;
- обробка чат-повідомлень: при надсиланні повідомлення, серверна логіка формує об'єкт повідомлення, який записується в відповідну чат-кімнату Firestore та негайно передається підписаним клієнтам через потоки даних;
- модерація та зберігання медіаконтенту: завантажені зображення обробляються та зберігаються у Firebase Storage, а посилання на них інтегруються в об'єкти повідомлень для подальшого відображення.

3.3.3 Діаграми

У цьому розділі представлено серію діаграм, що візуалізують та деталізують ключові процеси та компоненти серверної архітектури системи. Ці схеми є важливим інструментом для розуміння взаємодії елементів бекенду, демонструючи логіку потоків даних та функціональність кожного модуля.

Перша діаграма, зображена на рисунку 3.4, зосереджується на початковому етапі взаємодії користувача із системою. Вона детально ілюструє послідовність дій від моменту введення даних користувачем до їх обробки на серверній стороні. Це включає процес автентифікації користувача через Firebase Authentication та подальший запис даних у базі даних Firestore.

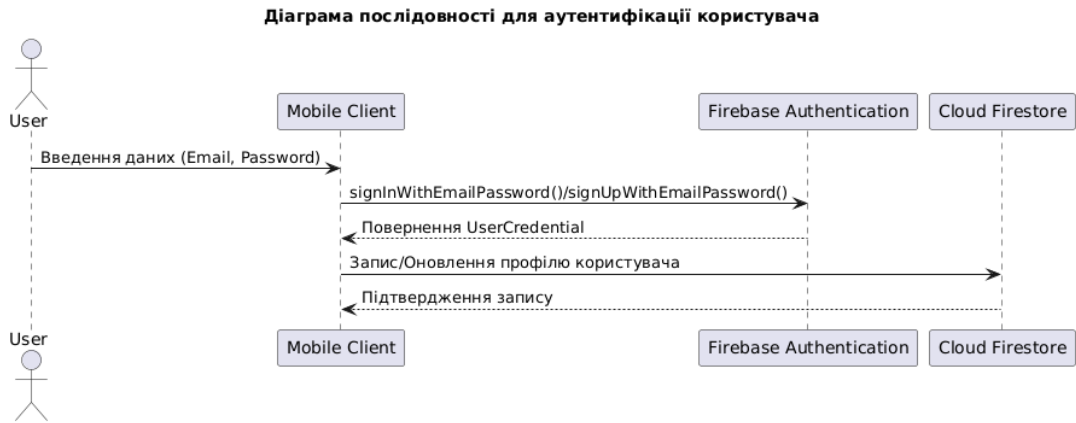


Рисунок 3.4 – Діаграма послідовності для аутентифікації користувача

Наступна діаграма ілюструє процес надсилання повідомлення: від створення повідомлення в клієнті, через сервісну логіку до збереження відповідного запису в Firestore, а також отримання оновлень у реальному часі. Діаграма наведена на рисунку 3.5.

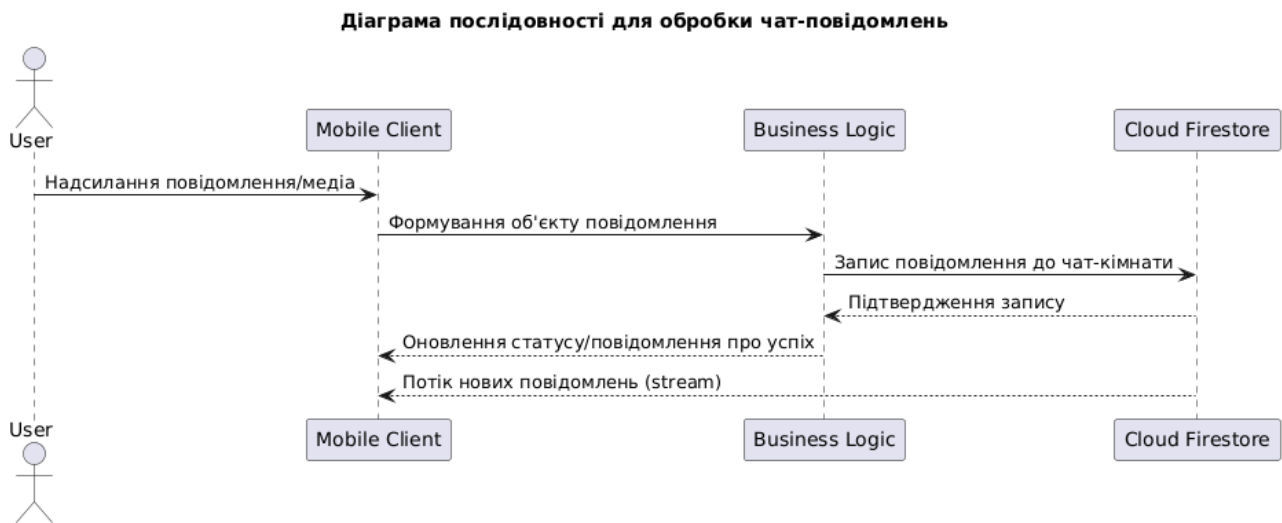


Рисунок 3.5 – Діаграма послідовності для обробки чат-повідомлень

Остання діаграма ілюструє взаємодію основних серверних компонентів Firebase із сервісною логікою додатку, що забезпечує аутентифікацію, обробку даних і завантаження медіаконтенту. Цю діаграму можна побачити на рисунку 3.6.

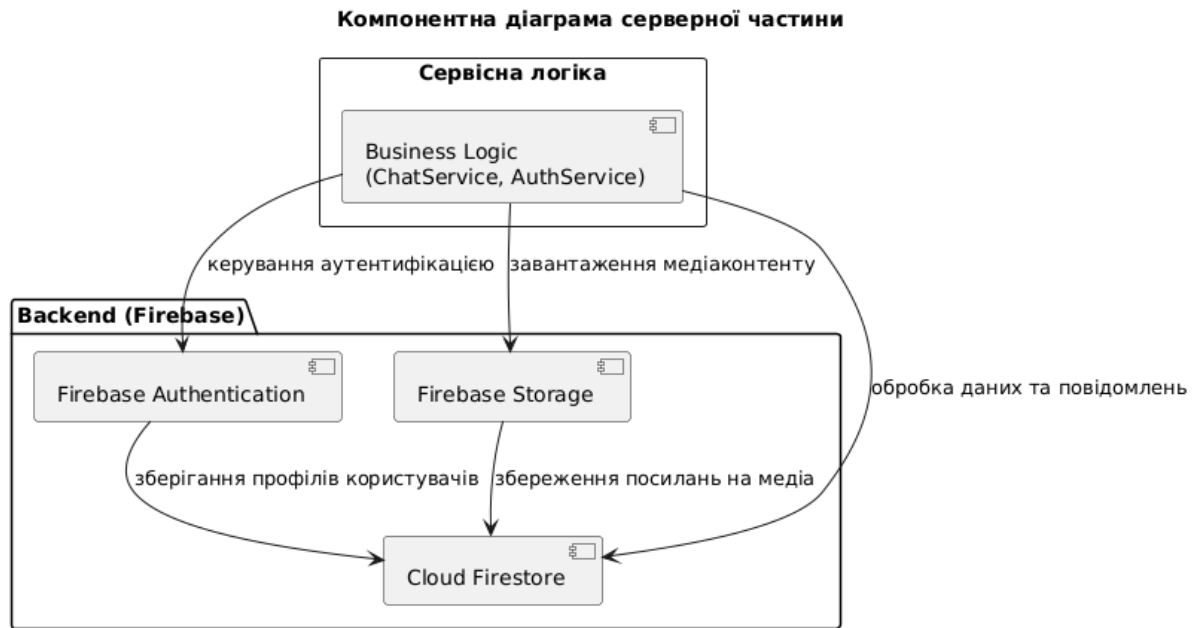


Рисунок 3.6 – Компонентна діаграма серверної частини

3.4 Проєктування бази даних

Для реалізації сховища даних у системі використовується хмарне NoSQL рішення – Cloud Firestore від платформи Firebase. Цей підхід дозволяє організувати зберігання даних за принципом колекцій і документів, що добре підходить для проєктування чат-додатків із високими вимогами до масштабованості та підтримки роботи у реальному часі.

3.4.1 Основні складові структури бази даних

База даних побудована так, щоб зручно та надійно зберігати всю необхідну інформацію для роботи застосунку. Вона містить кілька основних колекцій, які відповідають за користувачів, повідомлення в чатах і модерацію контенту. Кожна з них має свою структуру та призначення, що дозволяє швидко працювати з даними та підтримувати стабільну роботу системи.

Колекція "Users":

- документи користувачів: кожен користувач представлений окремим документом, в якому зберігаються основні атрибути;

- підколекція "BlockedUsers": для кожного користувача існує підколекція, де зберігається перелік ID користувачів, яких він заблокував. Це дозволяє зручно фільтрувати недоступних для спілкування учасників при відображенні списку контактів.

Колекція "chat_rooms":

- унікальні чат-кімнати: для кожної пари користувачів створюється документ, ідентифікований унікальним ID, що формується шляхом сортування їхніх UID та об'єднання їх у єдиний рядок (наприклад, user1_user2). Такий підхід гарантує консистентність ідентифікаторів;

- підколекція "messages": усередині кожного документу чат-кімнати розміщується підколекція, де зберігаються повідомлення. Кожне повідомлення зберігається як окремий документ із полями, а також додаткові дані для медіа-повідомлень.

Колекція "reports":

- документи зі звітами: у цій колекції зберігаються повідомлення про порушення. Це дозволяє організувати функціонал модерації і ефективно реагувати на інциденти.

3.4.2 Особливості проєктування бази даних

Під час проєктування бази даних були враховані основні принципи, які допомагають забезпечити швидку, зручну та надійну роботу системи.

Ключові підходи, на яких базується реалізація бази даних:

- денормалізація даних: структура бази даних оптимізована для швидкого зчитування інформації. Незважаючи на деяку надлишковість даних, денормалізація дозволяє мінімізувати кількість запитів до бази під час отримання інформації у режимі реального часу, що є критичним для чат-додатків;

– реальний час: використання Firestore із потоковими запитами забезпечує миттєве оновлення інтерфейсу при зміні даних. Це підвищує інтерактивність системи та покращує користувацький досвід;

– масштабованість: завдяки гнучкій архітектурі Firestore, яку легко масштабувати, система може ефективно обслуговувати зростаючу кількість користувачів без значних змін у структурі бази даних;

На рисунку 3.7 можна побачити схему бази даних.

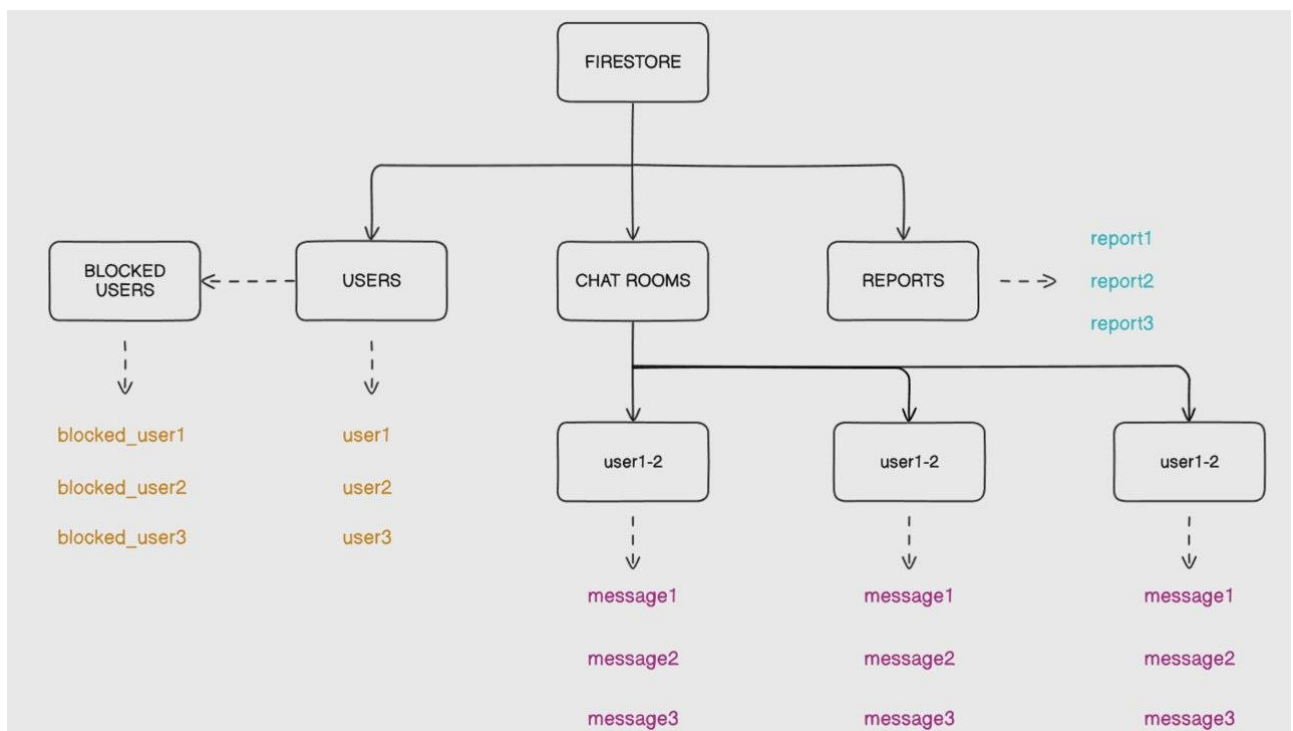


Рисунок 3.7 – Схема бази даних

Проектування бази даних здійснено із застосуванням переваг NoSQL підходу, що дозволяє зберігати дані у вигляді колекцій і документів, забезпечує високий рівень продуктивності та підтримує роботу в реальному часі. Структурування даних у відповідності до наданої схеми гарантує гнучкість, масштабованість і простоту управління інформацією, що є критичним для чат-додатку з оптимізацією обміну медіаконтентом.

3.5 Проєктування інтерфейсу користувача

У процесі розробки інтерфейсу користувача для чат-додатку було прийнято рішення дотримуватися принципу «Less is more» («Менше – це більше»).

Мінімалістичний дизайн, заснований на цьому принципі, передбачає усунення зайвих елементів, що можуть відволікати користувача, зосереджуючи увагу на основному функціоналі та контенті. Такий підхід забезпечує чистий, інтуїтивно зрозумілий інтерфейс, який сприяє покращенню користувацького досвіду [22].

Вибір цього принципу був обумовлений прагненням створити інтерфейс, який би не перевантажував користувача інформацією, а навпаки – полегшував взаємодію з додатком. Зосередженість на простоті та функціональності дозволяє користувачам швидко орієнтуватися в додатку, що особливо важливо для мобільних платформ, де екранний простір обмежений [22].

3.5.1 Ключові принципи дизайну

Дизайн застосунку розроблений із дотриманням ключових принципів, які спрямовані на забезпечення зручності, привабливості та ефективності роботи з інтерфейсом. Основні аспекти дизайну:

- простота та інтуїтивність: інтерфейс забезпечується завдяки чітко визначеним екранам для аутентифікації, головного меню, чатів та налаштувань. Принцип «less is more» дозволяє користувачу швидко орієнтуватися в додатку без перевантаження екранів зайвою інформацією;

- адаптивність: реалізація інтерфейсу базується на використанні віджетів Flutter, таких як Scaffold, AppBar, ListView та інших елементів, що відповідають принципам Material Design. Це дозволяє адаптувати інтерфейс до різних розмірів екранів та орієнтацій пристроїв;

- консистентність стилю: використання єдиного набору кольорових схем, шрифтів і відступів забезпечує однорідність візуального сприйняття на всіх екранах. Також передбачається підтримка як світлого, так і темного режимів, що дозволяє користувачеві обирати відповідний варіант відображення;

- плавність переходів: анімації та переходи між різними екранами реалізовано для забезпечення плавного користувацького досвіду. Це важливо для чат-додатку, де швидкість взаємодії та мінімізація затримок є критичними.

3.5.2 Основні екрани додатку

У додатку реалізовано кілька основних екранів, які відповідають за ключові функції та забезпечують зручну навігацію. Кожен екран виконує свою роль у системі, що дозволяє створити інтуїтивно зрозумілий і логічний досвід для користувача. Екрани додатку:

- екран запуску (Launch Screen): початковий екран, який містить логотип додатку;

- екран аутентифікації: екрани логіну/реєстрації для введення даних користувача;

- головний екран: сюди входить список чатів користувача з можливістю початку розмови. Також доступ до налаштувань розташовано через відповідні кнопки;

- екран чату: конкретний інтерфейс для спілкування, де відображаються історія повідомлень, можливості надсилання тексту та медіа;

- екран налаштувань: дозволяє користувачу керувати вибором теми та переходом до списку заблокованих користувачів;

- екран заблокованих користувачів: дозволяє переглянути всіх заблокованих користувачів та розблокувати їх.

3.5.3 Навігаційна структура

Щоб користувач завжди розумів, де він знаходиться в додатку, реалізовано чітку схему навігації. Наприклад, після проходження аутентифікації користувач потрапляє на головний екран, де може обрати конкретний чат або перейти до налаштувань. Для цього використовуються стандартні підходи Flutter – маршрути (routes) та панелі навігації (наприклад, Navigation Drawer чи Bottom Navigation Bar).

Нижче на рисунку 3.8 наведено приклад діаграми навігації, яка концептуально зображує потік між ключовими екранами додатку.

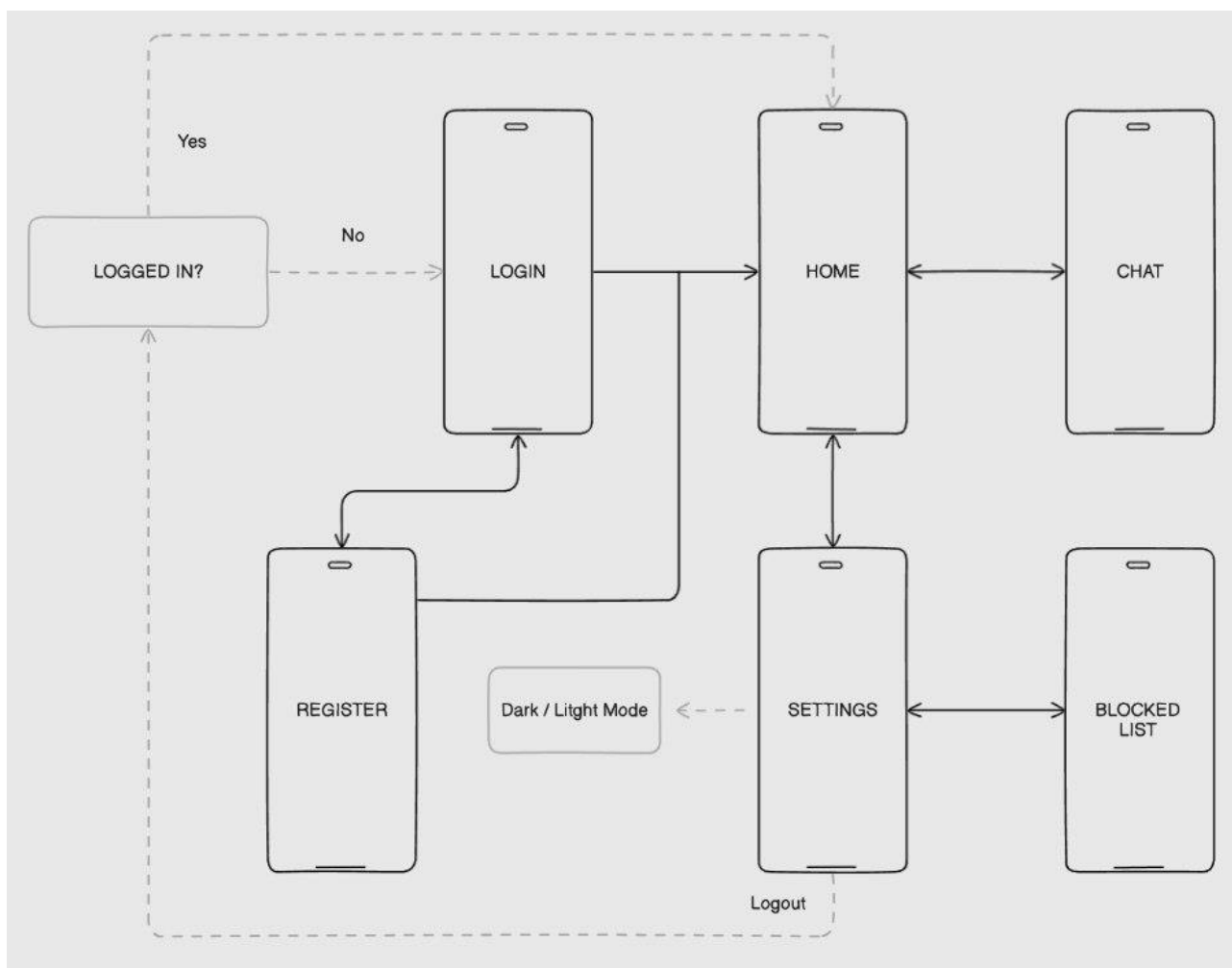


Рисунок 3.8 – Діаграма навігації додатку

Ця концептуальна діаграма допомагає зрозуміти загальну логіку переміщення між екранами, що є критичним для забезпечення інтуїтивного і оперативного користувацького досвіду.

Таким чином, проектування інтерфейсу користувача у додатку базується на чітко структурованій, адаптивній і консистентній моделі, де кожен екран відповідає за конкретний функціонал. Такий підхід забезпечує зручність взаємодії, швидкість відгуку та інтуїтивність роботи для кінцевого користувача.

3.6 Висновки за розділом

У цьому розділі було детально розглянуто всі ключові аспекти проектування системи. Вибрана архітектура забезпечує масштабованість, гнучкість та високу доступність, що дозволяє легко адаптувати додаток у відповідності до зростаючих вимог та при необхідності інтегрувати новий функціонал.

Клієнтська частина проєкту реалізована за допомогою Flutter, що дозволило створити єдиний код для різних платформ і розробити інтуїтивно зрозумілий, адаптивний інтерфейс. Чітко спланована навігація та ефективне управління станом додатка сприяють зручності та швидкості взаємодії користувачів із системою.

Серверна логіка базується на можливостях хмарних сервісів Firebase, зокрема використанні Authentication, Cloud Firestore і Storage. Такий підхід дозволяє забезпечити безпечну автентифікацію, обробку даних у режимі реального часу та надійне зберігання медіаконтенту, що є особливо важливим для чат-додатків.

Проектування бази даних реалізоване через організацію інформації у вигляді колекцій і документів, де денормалізація даних дозволяє зменшити кількість запитів до бази. Хоча цей підхід веде до певної надлишковості, він гарантує максимально швидке зчитування необхідної інформації, що критично для забезпечення оперативної роботи додатку.

Інтерфейс користувача спроектований на основі принципу «less is more», що дозволяє мінімізувати зайві елементи і зосередитись на основних функціях. Такий мінімалістичний підхід сприяє створенню чистого, сучасного та зручного для користувача дизайну, завдяки якому додаток стає інтуїтивно зрозумілим і комфортним у використанні.

Таким чином, інтеграція всіх описаних компонентів – від архітектури і ефективної реалізації клієнтської та серверної частин до продуманого проєктування бази даних і мінімалістичного інтерфейсу – дозволяє створити сучасний, масштабований і продуктивний чат-додаток.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Сценарії використання та призначення програми

Програма розроблена як чат-додаток для швидкого та зручного обміну повідомленнями в режимі реального часу. Її основна мета – забезпечення приватного спілкування між двома користувачами, що дозволяє оперативно обмінюватися текстовими повідомленнями та медіаконтентом. Dodatok орієнтований на тих, хто цінує простоту, інтуїтивність та ефективність комунікації без зайвих функцій.

Типовий сценарій використання виглядає наступним чином: користувач запускає додаток, проходить автентифікацію через екран входу або реєстрації, після чого потрапляє на головний екран, де відображається список доступних контактів або попередніх розмов. Обравши потрібного співрозмовника, користувач переходить до приватного чату, де може надсилати текстові повідомлення або медіа. Зображення, що надсилаються, спочатку відображаються як стиснутий передпоказ, а при натисканні – відкриваються у повному розмірі. Також користувач має можливість змінити тему інтерфейсу, обираючи між світлим та темним режимами, що дозволяє налаштовувати комфортне середовище для спілкування.

Основні функції додатку:

- автентифікація: безпечний вхід та реєстрація користувачів;
- приватний чат: обмін повідомленнями між двома користувачами в режимі реального часу;
- обмін медіаконтентом: надсилення зображень із стиснутим припоказом та можливістю перегляду оригіналу;
- зміна теми інтерфейсу: перемикання між світлим та темним режимами для зручності користувача;

– управління контактами: включає блокування небажаних користувачів, розблокування раніше заблокованих контактів та систему скарг на повідомлення для фільтрації порушень.

Таким чином, призначення програми полягає у створенні простого, але ефективного засобу для приватного спілкування, де кожна функція спрямована на покращення користувацького досвіду та оперативності обміну інформацією.

4.2 Системні вимоги та умови експлуатації

Для стабільної роботи та оптимальної взаємодії з чат-додатком необхідно дотримуватися наступних системних вимог і умов експлуатації. Оскільки додаток розроблено за допомогою Flutter з використанням хмарних сервісів Firebase, на апаратному рівні він не вимагає надзвичайно потужних ресурсів, але правильна конфігурація середовища використання є важливою для забезпечення безперебійної роботи у режимі реального часу.

4.2.1 Апаратні та програмні вимоги

Щоб додаток працював стабільно та швидко, потрібно врахувати основні апаратні та програмні вимоги. Вони допомагають забезпечити надійність і зручність використання.

Основні параметри, які варто враховувати:

– мобільні пристрої: рекомендовано використовувати пристрої з операційними системами Android (версія 5.0 або новіша) або iOS (версія 10 або новіша). Для комфортного використання додаток бажано запускати на пристроях з не менше ніж 2 Гб оперативної пам'яті;

– інтернет-з'єднання: оскільки чат-додаток передбачає обмін повідомленнями у режимі реального часу, особливо важливо забезпечити стабільне високошвидкісне з'єднання з Інтернетом. Затримки у з'єднанні можуть впливати на швидкість доставки повідомлень та завантаження медіаконтенту.

4.2.2 Умови експлуатації

Щоб чат-додаток працював стабільно та з максимальною продуктивністю, важливо дотримуватись певних умов використання. Основні рекомендації, які сприяють покращенню роботи системи:

- оновлення програмного середовища: для забезпечення сумісності з оновленнями хмарних сервісів Firebase та Flutter рекомендується регулярно перевіряти та встановлювати оновлення застосунку, операційної системи і драйверів пристрою;
- оптимізація енергоспоживання: з огляду на характер роботи додатку, де обмін даними відбувається постійно, бажано стежити за належним рівнем заряду акумулятора пристрою або використовувати зарядний пристрій для тривалої роботи у разі мобільного використання;
- мережеві умови: оптимальна робота передбачає наявність стабільного Wi-Fi-з'єднання або мобільного інтернету з достатньою пропускну здатністю, що дозволить уникнути затримок при завантаженні фото чи обміну повідомленнями.

Загалом, додаток терпимий до широкого спектру пристроїв і ОС завдяки ефективній організації коду та використанню хмарних сервісів. Проте дотримання наведених вище вимог і умов експлуатації є важливим для забезпечення високої продуктивності та надійності роботи чат-додатку в реальному часі.

4.3 Демонстрація екранних рішень

На першому екрані користувач зустрічається із чистим та мінімалістичним дизайном, що відповідає принципу «less is more». Екран містить поля для введення електронної пошти та пароля, а також окрему кнопку для входу в систему. Нижче в виділеному тексті є посилання для переходу до сторінки реєстрації нового облікового запису.

Приклад інтерфейсу наведено на рисунку 4.1.

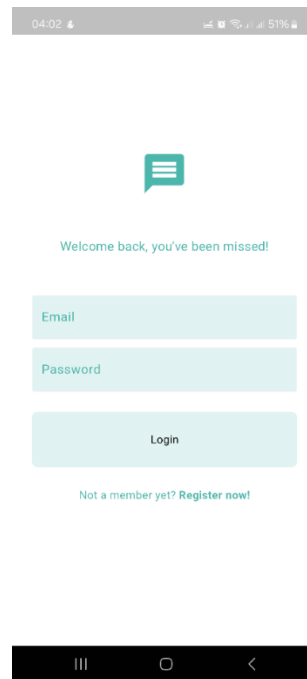


Рисунок 4.1 – Сторінка авторизації

Після успішної авторизації користувач потрапляє на домашню сторінку.

На наступному екрані представлено сторінку реєстрації. Сторінка містить форму для введення особистих даних, таких як електронна пошта, бажаний пароль та його підтвердження. Елементи розміщені логічно та інтуїтивно, що дозволяє користувачу швидко заповнити необхідні поля без зайвих зусиль. Нижче форми виділено велику кнопку для завершення реєстрації, натискання на яку створює новий обліковий запис.

Приклад інтерфейсу наведено на рисунку 4.2.

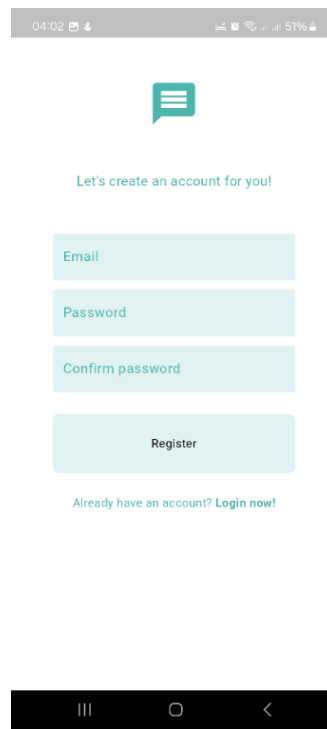


Рисунок 4.2 – Сторінка реєстрації нового облікового запису

Після успішної реєстрації користувач далі переходить до домашньої сторінки додатку.

Також, нижче в виділеному тексті є посилання для переходу до сторінки авторизації, якщо обліковий запис уже створено.

На наступному екрані користувача очікує головний інтерфейс, де представлено список чатів. Список містить картки кожного чату, в яких відображено електронну пошту співрозмовника. Приклад інтерфейсу наведено на рисунку 4.3.

У верхній лівій частині екрана розміщується додаткова кнопка (бургер) для відкриття бокової панелі з обціями переходу до налаштувань та виходу з облікового запису. Бокову панель показано на рисунку нижче. Приклад інтерфейсу наведено на рисунку 4.4.

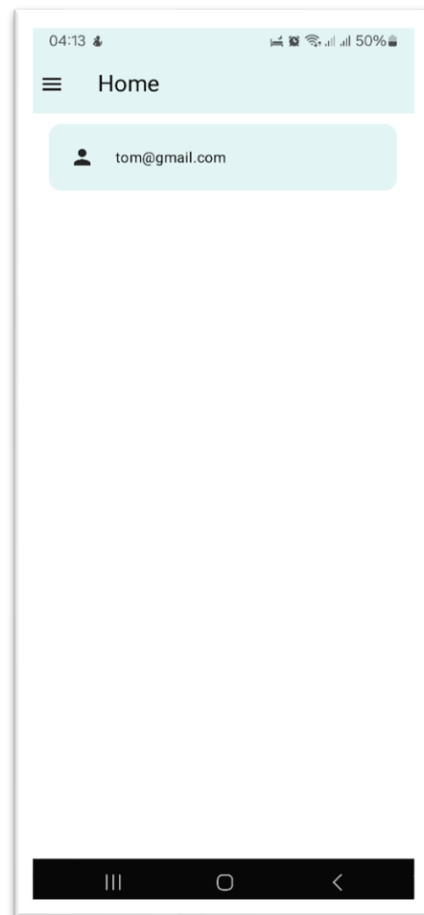


Рисунок 4.3 – Домашня сторінка

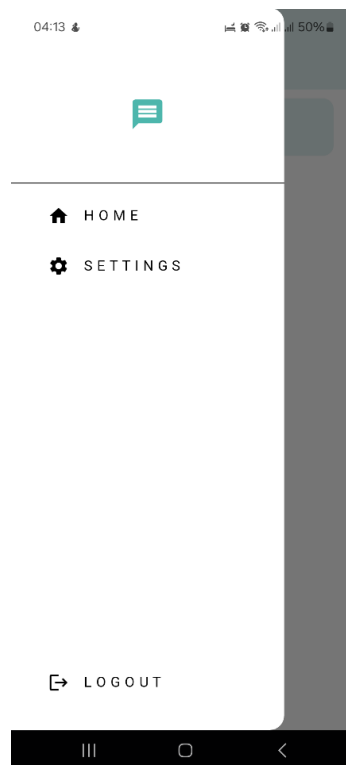


Рисунок 4.4 – Бокова панель (бургер панель)

На наступному екрані представлено інтерфейс приватного чату. У верхній частині розміщується панель з інформацією про співрозмовника, де вказано його електронну пошту, а також можливість повернутися до домашньої сторінки. Основна частина заповнена історією повідомлень. В нижній частині екрану розташоване текстове поле для введення повідомлення, а також кнопки для додавання медіаконтенту і відправлення повідомлення. Зображення відправляються у вигляді стислого передпоказу. Приклад інтерфейсу наведено на рисунку 4.5.

При натисканні на фото воно відкривається в оригінальному розмірі. Сторінка з відкритим фото наведена на рисунку 4.6.

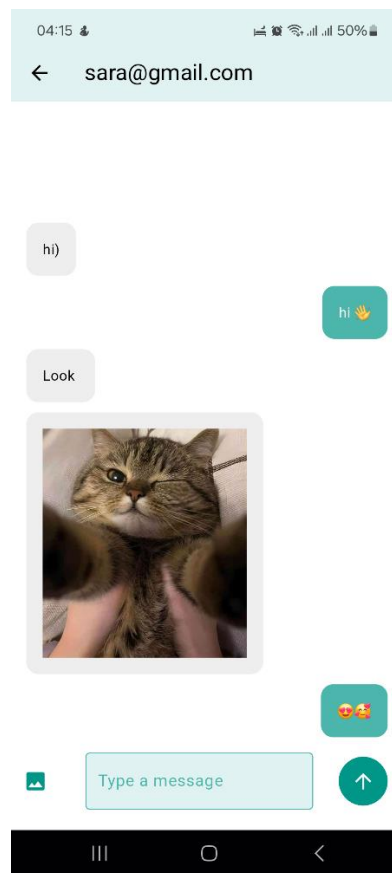


Рисунок 4.5 – Сторінка чату



Рисунок 4.6 – Відкрите фото

При затисканні на конкретне повідомлення свого співрозмовника на екрані чату знизу відкривається контекстне меню, яке надає швидкий доступ до додаткових дій. Меню оформлено у вигляді спливаючого блоку з кількома опціями, кожна з яких має інтуїтивно зрозумілі іконки та підписи. Основні дії, доступні у цьому меню, включають: подати скаргу на повідомлення, заблокувати користувача та закрити меню. Приклад інтерфейсу наведено на рисунку 4.7.

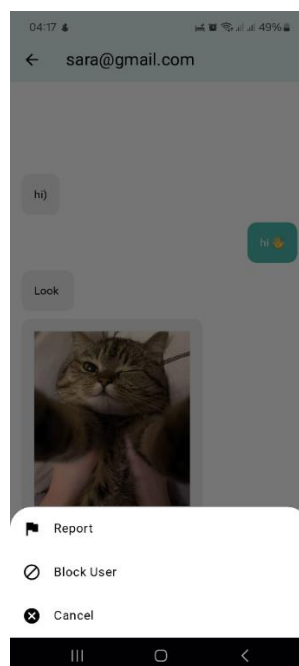


Рисунок 4.7 – Меню при затисканні на повідомлення

На наступному екрані налаштувань, на рисунку 4.8, представлено всі опції для налаштування роботи додатку та управління обліковим записом. Основна частина екрану містить перелік функціональних параметрів: зміна теми та перехід на сторінку зі списком заблокованих користувачів. Темна тема додатку зображена на рисунку 4.9.

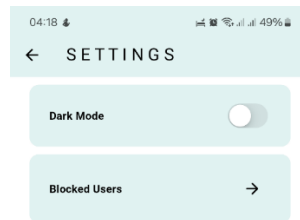


Рисунок 4.8 – Сторінка налаштувань

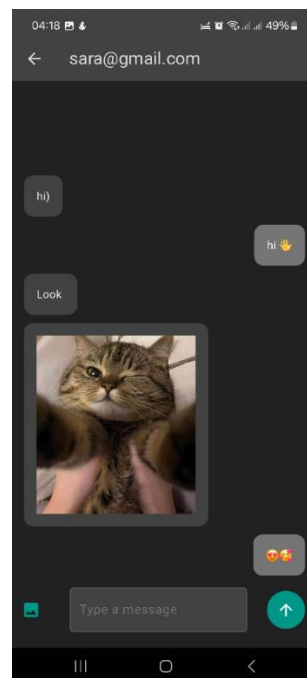


Рисунок 4.9 – Темна тема додатку

На останньому екрані представлено список користувачів, яких було заблоковано для обмеження небажаного спілкування. Кожна картка в списку відображає електронну пошту заблокованого користувача, а також при натисканні на неї можна виконати розблокування. Приклад інтерфейсу наведено на рисунку 4.10.

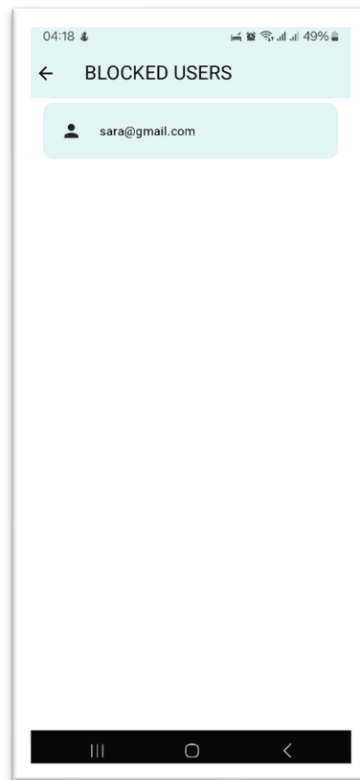


Рисунок 4.10 – Сторінка зі списком заблокованих користувачів

4.4 Тестування функцій додатку

Для перевірки функціональних можливостей застосунку було проведено серію ручних тестів, спрямованих на оцінку роботи ключових функцій. Тестування охоплювало такі сценарії: автентифікація користувачів, обмін текстовими повідомленнями, робота з медіаконтентом (надсилення зображень з припоказом та перегляд оригіналу), зміна теми інтерфейсу, а також управління контактами – блокування, розблокування і подання скарг на повідомлення. Тести виконувалися на справжньому мобільному пристрої на базі Android та на

емуляторі пристрою з операційною системою iOS, із застосуванням стандартних тестових сценаріїв.

Результати тестування основних функцій додатка наведено в таблиці 4.1.

Таблиця 4.1 – Результати тестування функціоналу додатка

Функція	Опис тестування	Результат (Android)	Результат (iOS)
Автентифікація користувачів	Перевірка коректності входу та реєстрації з обробкою введених даних.	Успішно. Дані обробляються коректно.	Успішно. Дані обробляються коректно.
Приватний чат	Обмін текстовими повідомленнями у режимі реального часу між двома користувачами.	Успішно. Повідомлення доставляються швидко.	Успішно. Повідомлення доставляються швидко.
Обмін медіаконтентом	Надсилання зображень, які спочатку відображаються як стислий передпоказ, а при натисканні відкриваються в оригінальному розмірі.	Успішно. Передпоказ завантажується коректно, відкриття оригіналу відбувається без збоїв.	Успішно. Передпоказ завантажується коректно, відкриття оригіналу відбувається без збоїв.

Продовження таблиці 4.1

Функція	Опис тестування	Результат (Android)	Результат (iOS)
Зміна теми інтерфейсу	Перемикання між світлим та темним режимами інтерфейсу.	Успішно. Перемикання відбувається миттєво та без помилок.	Успішно. Перемикання відбувається миттєво та без помилок.
Управління контактами – блокування	Перевірка можливості блокування користувачів.	Успішно. Заблоковані користувачі відображаються у списку заблокованих.	Успішно. Заблоковані користувачі відображаються у списку заблокованих.
Управління контактами – розблокування	Тестування можливості відновлення спілкування після блокування контакту.	Успішно. Розблокування виконується коректно та забезпечує відновлення зв'язку та історії чату.	Успішно. Розблокування виконується коректно та забезпечує відновлення зв'язку та історії чату.
Скарги на повідомлення	Подання скарг на повідомлення.	Успішно. Скарги правильно фіксуються системою.	Успішно. Скарги правильно фіксуються системою.

Таким чином, результати тестування підтверджують, що всі основні функції додатку працюють стабільно та відповідають встановленим вимогам як

на Android, так і на iOS, що дозволяє рекомендувати застосунок для подальшої експлуатації.

4.5 Оцінка зручності використання

Оцінка зручності використання додатку проводилася шляхом аналізу внутрішніх критеріїв, спрямованих на визначення рівня комфорту взаємодії з інтерфейсом. В процесі аналізу було встановлено, що функції, кнопки та розташування елементів виконані інтуїтивно зрозуміло і відповідають загальноприйнятим стандартам. Популярні аналоги мають схожі шаблони, до яких вже звикли користувачі, тому цей додаток адаптовано саме за цим принципом, що дозволяє швидко орієнтуватися в системі.

Результати оцінювання зручності використання наведено в таблиці 4.2.

Таблиця 4.2 – Результати оцінювання зручності використання

Критерій	Оцінка (Android)	Оцінка (iOS)	Коментарі
Інтуїтивність інтерфейсу	4.7/5	4.8/5	Швидка орієнтація в системі та доступ до основних функцій завдяки знайомій структурі.
Зручність навігації	4.6/5	4.5/5	Мінімалістичний дизайн сприяє легкому перемиканню між екранними рішеннями.

Продовження таблиці 4.2

Критерій	Оцінка (Android)	Оцінка (iOS)	Коментарі
Зрозумілість елементів управління	4.8/5	4.7/5	Чітко позначені кнопки та поля введення забезпечують простоту використання.
Загальний досвід використання	4.7/5	4.8/5	Загальна оцінка підтверджує високий рівень комфорту.

Отже, завдяки використанню знайомих шаблонів та інтуїтивно організованих елементів, додаток забезпечує високий рівень зручності використання, що сприяє позитивному користувацькому досвіду на обох платформах.

4.6 Висновки за розділом

Підсумовуючи представлені результати, можна стверджувати, що розроблений чат-додаток успішно відповідає встановленим вимогам як з точки зору функціональності, так і з точки зору зручності використання. Основні сценарії застосування – від автентифікації та реєстрації до управління чатами та контактами – реалізовано згідно з принципом "less is more", що забезпечує чистий, мінімалістичний та інтуїтивно зрозумілий інтерфейс.

Демонстрація екранних рішень засвідчує, що кожен елемент дизайну (від домашнього екрану до контекстного меню при взаємодії з повідомленнями) побудований за загальновизнаною структурою, до якої більшість користувачів

вже звикла в подібних застосунках. Це сприяє швидкому освоєнню та ефективному використанню додатку.

Також тестування функцій та юзабіліті показало високий рівень стабільності, швидкості відгуку та комфорт роботи на обох основних платформах – Android та iOS. Отримані результати дозволяють рекомендувати додаток для подальшої експлуатації з подальшим удосконаленням деяких аспектів, орієнтованих на підвищення зручності та адаптивності інтерфейсу.

Отже, результати свідчать, що дизайн та функціональність додатку ретельно оптимізовані для забезпечення ефективної та інтуїтивно зрозумілої роботи. Реалізація основних сценаріїв у поєднанні з чітким та мінімалістичним інтерфейсом гарантує комфортну взаємодію, що робить додаток перспективним для подальшої експлуатації та розвитку.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було розроблено кросплатформний мобільний застосунок для чатування, що забезпечує ефективний обмін текстовими повідомленнями та медіаконтентом.

Для цього було виконано ряд завдань:

- проведено аналіз предметної області та досліджено сучасні технології комунікації;
- розроблено загальну архітектуру системи та спроектовано її основні компоненти (клієнтська і серверна частини, база даних, інтерфейс користувача);
- реалізовано функціонал авторизації, управління контактами, чатування, а також оптимізації передачі медіаконтенту шляхом стиснення зображень;
- здійснено тестування застосунку на платформах Android та iOS для перевірки його стабільності, швидкодії та зручності використання.

Розроблений застосунок був реалізований з використанням сучасного інструментарію – фреймворку Flutter для створення кросплатформних рішень та сервісу Firebase для зберігання даних і автентифікації користувачів.

Застосунок було протестовано на основних мобільних платформах, що дозволило перевірити його працездатність, оптимізувати швидкість завантаження медіаконтенту та забезпечити високий рівень юзабіліті.

Застосунок реалізує такі функціональні вимоги:

- забезпечення швидкого і стабільного обміну повідомленнями;
- підтримку передачі як тексту, так і фото;
- оптимізацію передачі даних завдяки реалізації функції стиснення зображень;
- управління обліковим записом користувача, включаючи налаштування інтерфейсу та можливості роботи з контактами (функції чорного списку та подання скарг).

В результаті розробки було створено сучасний, стабільний та зручний чат-додаток, що відповідає вимогам щодо продуктивності та ефективного використання ресурсів мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Flutter vs. React Native: A Comparison of Cross-Platform Frameworks [Electronic resource]. – Access mode: <https://clarity.microsoft.com/blog/flutter-vs-react-native-a-comparison-of-cross-platform-frameworks-2/>.
2. An image format for the Web [Electronic resource]. – Access mode: <https://developers.google.com/speed/webp>.
3. Number of messaging app users worldwide 2023 | Statista [Electronic resource]. – Access mode: <https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide>.
4. Firebase Documentation – Build apps fast, without managing infrastructure [Electronic resource]. – Access mode: <https://firebase.google.com/docs>.
5. Flutter Case Studies [Electronic resource]. – Access mode: <https://flutter.dev/showcase>.
6. No More Blurry Images: Your Texts Are Getting an Upgrade [Electronic resource]. – Access mode: <https://www.wsj.com/tech/personal-tech/iphone-android-rcs-texting-features-78ba502a>.
7. The Fight to Unite iPhone and Android Users Is Far From Over [Electronic resource]. – Access mode: <https://www.wired.com/story/iphone-android-apple-imessage-beeper/>.
8. Ultimate Guide to In-App Messaging 2024 [Electronic resource]. – Access mode: <https://bizbot.com/blog/ultimate-guide-to-in-app-messaging-2024/>.
9. WhatsApp Message privately [Electronic resource]. – Access mode: <https://www.whatsapp.com/privacy>.
10. Top 6 WhatsApp features: From AI Studio, built-in dial pad, custom photo stickers and more [Electronic resource]. – Access mode: <https://www.thehansindia.com/technology/tech-news/whatsapps-top-6-features-of-2025-ai-chat-custom-stickers-dial-pad-and-more-941840>.
11. Telegram Messenger [Electronic resource]. – Access mode: <https://apps.apple.com/us/app/telegram-messenger/id686449807>.

12. Telegram (software) [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Telegram_%28software%29.
13. Why use Signal? [Electronic resource]. – Access mode: <https://signal.org/>.
14. What is Signal? 7 features that make it a go-to app for private, secure messaging [Electronic resource]. – Access mode: <https://www.zdnet.com/article/what-is-signal-7-features-that-make-it-a-go-to-app-for-private-secure-messaging/>.
15. Dart vs Kotlin: Top Differences [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/dart-vs-kotlin/#dart-vs-kotlin-key-differences>.
16. Dart vs. JavaScript: Which Language Should You Choose? [Electronic resource]. – Access mode: <https://clouddevs.com/dart/vs-javascript/>.
17. Distinctive Features and Advantages of Flutter Apps, and What Are the Disadvantages [Electronic resource]. – Access mode: <https://intent-solutions.com/en/blog/advantages-of-flutter-development>.
18. Store and sync data in real time [Electronic resource]. – Access mode: <https://firebase.google.com/products/realtime-database/>.
19. Compare AWS Amplify vs. Firebase vs. Supabase [Electronic resource]. – Access mode: <https://slashdot.org/software/comparison/AWS-Amplify-vs-Firebase-vs-Supabase/>.
20. Exploring Backend as a Service (BaaS) Platforms: Firebase vs. Supabase vs. AWS Amplify [Electronic resource]. – Access mode: <https://www.hyper-leap.com/2024/05/08/exploring-backend-as-a-service-baas-platforms-firebase-vs-supabase-vs-aws-amplify/>.
21. Розробка мобільних додатків [Електрон. ресурс]. – Режим доступу: <https://brander.ua/technologies/mvvm>.
22. Мінімалістичні тенденції дизайну інтерфейсу користувача [Електрон. ресурс]. – Режим доступу: <https://dizz.in.ua/uk/minimalistichni-tendenczii-dizajnu-interfejsu-koristuvacha/>.

ДОДАТОК А
Технічне завдання

Вступ

Сучасні мобільні чат-застосунки набувають все більшої популярності завдяки зростанню мобільних технологій та збільшенню вимог до оперативної комунікації. Використання таких сервісів дозволяє забезпечити швидкий та зручний обмін не лише текстовими повідомленнями, але й медіаконтентом, що є критично важливим для сучасного ринку цифрових технологій. Розробка кросплатформного чат-застосунку з урахуванням таких факторів спрямована на оптимізацію процесів передачі даних, економію мобільного трафіку та підвищення якості взаємодії між користувачами. Для досягнення зазначених цілей застосовуються сучасні технології, такі як Flutter і Firebase, які дозволяють створити стабільний, безпечний та інтуїтивно зрозумілий продукт.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація кросплатформного мобільного застосунку чатування з оптимізацією обміну медіаконтентом», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

А.2 Основні вимоги до програми, що розробляється

А.2.1 Вимоги до функціональних характеристик

Вимоги до програми:

- автентифікація користувачів: забезпечення надійної верифікації через використання сучасних методів автентифікації (наприклад, через Firebase Authentication);
- обмін повідомленнями та медіаконтентом: реалізація можливості текстового чатування, а також передачі зображень;

- стиснення зображень: впровадження функції стиснення зображень для оптимізації об'єму переданих даних з метою економії трафіку та пришвидшення завантаження контенту;
- управління користувачами: забезпечення функціоналу управління контактами, можливості блокування користувачів та подання скарг.

A.2.2 Вимоги до інтерфейсу програми

Програма повинна мати:

- інтуїтивність: інтерфейс має бути простим та зрозумілим, забезпечуючи інтуїтивну навігацію між розділами застосунку;
- мінімалістичний дизайн: дотримання принципу «less is more» з чітким розмежуванням інформаційних блоків;
- підтримка режимів: забезпечення підтримки як світлого, так і темного режимів, що дозволяє користувачу обрати оптимальний варіант в залежності від освітлення;
- сумісність: інтерфейс повинен коректно відображатися на різних розмірах екранів мобільних пристроїв.

A.2.3 Вимоги до надійності

До вимог щодо надійності належить гарантування безперебійної та стабільної роботи як клієнтської, так і серверної частин застосунку. Система повинна працювати з мінімальними зупинками, забезпечуючи збереження даних та унеможлиблюючи непередбачувані витоки пам'яті, що сприяє надійній експлуатації програми.

A.2.4 Вимоги до складу та параметрів технічних засобів

Мобільні пристрої: застосунок має коректно працювати на сучасних смартфонах із операційними системами Android та iOS.

Мінімальні вимоги включають:

- 2 Гб оперативної пам'яті;
- 8 Гб вбудованої пам'яті;
- підтримку Wi-Fi та мобільного зв'язку.

Серверні ресурси: серверна частина має бути розгорнута на надійній хмарній платформі, що забезпечує масштабованість, високу доступність та безперервну роботу.

Додаткові засоби: необхідна наявність засобів резервного копіювання та моніторингу роботи системи для запобігання втраті даних і оперативного реагування на потенційні проблеми.

ДОДАТОК Б
Текст програми

Б.1 Текст програми з файлу main.dart

```
import 'package:Chattix/services/auth/auth_controller.dart';
import 'package:Chattix/firebase_options.dart';
import 'package:Chattix/theme/theme_provider.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );

  runApp(
    ChangeNotifierProvider(
      create: (context) => ThemeProvider(),
      child: const MainApp(),
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: const AuthController(),
      theme: Provider.of<ThemeProvider>(context).themeData,
    );
  }
}
```

Б.2 Текст програми з файлу chat_message.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';

class ChatMessage {
  final String senderID;
  final String senderEmail;
  final String receiverID;
  final String? textContent;
  final String? previewImageUrl;
  final String? fullImageUrl;
  final Timestamp timestamp;

  ChatMessage({
    required this.senderID,
    required this.senderEmail,
    required this.receiverID,
    this.textContent,
    this.previewImageUrl,
    this.fullImageUrl,
    required this.timestamp,
  });

  Map<String, dynamic> toMap() {
    return {
      'senderID': senderID,
      'senderEmail': senderEmail,
      'receiverID': receiverID,
      'message': textContent,
      'smallImageUrl': previewImageUrl,
      'originalImageUrl': fullImageUrl,
      'timestamp': timestamp,
    };
  }
}
```

Б.3 Текст програми з файлу `messaging_service.dart`

```
import 'package:Chattix/models/chat_message.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class ChatService extends ChangeNotifier {
  // get instance of firestore & auth
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  final FirebaseAuth _auth = FirebaseAuth.instance;

  // get all users stream
  Stream<List<Map<String, dynamic>>> getAllUsersStream() {
    return _firestore.collection("Users").snapshots().map((snapshot) {
      return snapshot.docs.map((doc) {
        // go through each individual user
        final user = doc.data();

        // return user
        return user;
      }).toList();
    });
  }

  // get all users stream except blocked users
  Stream<List<Map<String, dynamic>>> getUsersStreamExcludingBlocked() {
    final currentUser = _auth.currentUser;

    return _firestore
      .collection("Users")
      .doc(currentUser!.uid)
      .collection('BlockedUsers')
      .snapshots()
      .asyncMap((snapshot) async {
```

```

// get blocked user IDs
final blockedUserIds = snapshot.docs.map((doc) => doc.id).toList();

// get all users
final userSnapshot = await _firestore.collection("Users").get();

// return as stream list, excluding current user and blocked users
return userSnapshot.docs
    .where((doc) =>
        doc.data()['email'] != currentUser.email &&
        !blockedUserIds.contains(doc.id))
    .map((doc) => doc.data())
    .toList();
});
}

// send message
Future<void> sendTextMessage(String receiverID, message) async {
    // get current user info
    final String currentUserID = _auth.currentUser!.uid;
    final String currentUserEmail = _auth.currentUser!.email!;
    final Timestamp timestamp = Timestamp.now();

    // create a new message
    ChatMessage newMessage = ChatMessage(
        senderID: currentUserID,
        senderEmail: currentUserEmail,
        receiverID: receiverID,
        textContent: message,
        timestamp: timestamp,
    );

    // construct chat room ID for the two users (sorted to ensure uniqueness)
    List<String> ids = [currentUserID, receiverID];
    ids.sort(); // sort the IDs (this ensures the chatroomID is the same for any
two users)

```

```

String chatRoomID = ids.join('_');

// add message to database
await _firestore
    .collection("chat_rooms")
    .doc(chatRoomID)
    .collection("messages")
    .add(newMessage.toMap());
}

// send Image
Future<void> sendImageMessage(
    String receiverID, String smallImageUrl, String originalImageUrl) async {
    final String currentUserID = _auth.currentUser!.uid;
    final String currentUserEmail = _auth.currentUser!.email!;
    final Timestamp timestamp = Timestamp.now();

    ChatMessage newMessage = ChatMessage(
        senderID: currentUserID,
        senderEmail: currentUserEmail,
        receiverID: receiverID,
        textContent: null,
        previewImageUrl: smallImageUrl.isNotEmpty ? smallImageUrl : null,
        fullImageUrl: originalImageUrl.isNotEmpty ? originalImageUrl : null,
        timestamp: timestamp,
    );

    List<String> ids = [currentUserID, receiverID];
    ids.sort();
    String chatRoomID = ids.join('_');

    await _firestore
        .collection("chat_rooms")
        .doc(chatRoomID)
        .collection("messages")
        .add(newMessage.toMap());
}

```

```

}

Future<QuerySnapshot> getMessagesOnce(
    String userID, String otherUserID) async {
    List<String> ids = [userID, otherUserID];
    ids.sort();
    String chatRoomID = ids.join('_');

    return await _firestore
        .collection("chat_rooms")
        .doc(chatRoomID)
        .collection("messages")
        .orderBy("timestamp", descending: false)
        .get();
}

// get messages
Stream<QuerySnapshot> getMessages(String userID, otherUserID) {
    // construct a chatroom ID for the two users
    List<String> ids = [userID, otherUserID];
    ids.sort();
    String chatRoomID = ids.join('_');

    return _firestore
        .collection("chat_rooms")
        .doc(chatRoomID)
        .collection("messages")
        .orderBy("timestamp", descending: false)
        .snapshots();
}

// report user
Future<void> reportUser(String messageId, String userId) async {
    // get current user ID
    final currentUser = _auth.currentUser;
    final report = {

```

```

        'reportedBy': currentUser!.uid,
        'messageId': messageId,
        'messageOwnerId': userId,
        'timestamp': FieldValue.serverTimestamp(),
    };

    await _firestore.collection('reports').add(report);
}

// block user
Future<void> blockUser(String userId) async {
    final currentUser = _auth.currentUser;

    await _firestore
        .collection('Users')
        .doc(currentUser!.uid)
        .collection('BlockedUsers')
        .doc(userId)
        .set({});
    notifyListeners();
}

// unblock user
Future<void> unblockUser(String blockedUserId) async {
    final currentUser = _auth.currentUser;

    await _firestore
        .collection('Users')
        .doc(currentUser!.uid)
        .collection('BlockedUsers')
        .doc(blockedUserId)
        .delete();
    notifyListeners();
}

// get blocked users stream

```

```

Stream<List<Map<String, dynamic>>> getBlockedUsersStreamByUserId(
    String userId) {
    return _firestore
        .collection('Users')
        .doc(userId)
        .collection('BlockedUsers')
        .snapshots()
        .asyncMap((snapshot) async {
            // get list of blocked user IDs
            final blockedUserIds = snapshot.docs.map((doc) => doc.id).toList();

            final userDocs = await Future.wait(
                blockedUserIds
                    .map((id) => _firestore.collection('Users').doc(id).get()),
            );

            // return as a list
            return userDocs.map((doc) => doc.data() as Map<String, dynamic>).toList();
        });
    }
}

```

Б.4 Текст програми з файлу `image_service.dart`

```

import 'dart:io';
import 'package:image_picker/image_picker.dart';
import 'package:image/image.dart' as img;
import 'package:firebase_storage/firebase_storage.dart';

class ImageService {
    final ImagePicker _picker = ImagePicker();

    Future<Map<String, File?>> pickAndCompressImage() async {
        final pickedFile = await _picker.pickImage(source: ImageSource.gallery);
    }
}

```

```

if (pickedFile != null) {
    File imageFile = File(pickedFile.path);

    // Compress an image for display in chat
    img.Image? image = img.decodeImage(imageFile.readAsBytesSync());
    if (image != null) {
        File compressedImage = File('${imageFile.path}_compressed.jpg')
            ..writeAsBytesSync(img.encodeJpg(image, quality: 30));

        return {
            'small': compressedImage,
            'original': imageFile,
        };
    }
}
return {};
}

Future<String> uploadImageToFirebase(File imageFile) async {
    try {
        FirebaseStorage storage = FirebaseStorage.instance;
        Reference ref = storage
            .ref()
            .child('chat_images/${DateTime.now().millisecondsSinceEpoch}.jpg');
        await ref.putFile(imageFile);
        String downloadURL = await ref.getDownloadURL();
        return downloadURL;
    } catch (e) {
        return '';
    }
}
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

Програмна реалізація кросплатформного мобільного застосунку чатування з оптимізацією обміну медіаконтентом

Виконав:
ст. групи КНТ-221

Керівник:
д-р.філос., доцент

Андрій ДАНИЛІН

Сергій ЛЕОЩЕНКО



Рисунок В.1 – Слайд 1

2

Об'єкт предмет та мата роботи

- 01** Об'єкт дослідження – процес обміну медіаконтентом у мобільних чат-додатках.
- 02** Предмет дослідження – методи оптимізації передавання зображень у мобільних застосунках.
- 03** Мета роботи – розробка кросплатформного мобільного застосунку чатування з можливістю оптимізованого обміну медіаконтентом.

Рисунок В.2 – Слайд 2

Постановка завдання

Метою дипломної кваліфікаційної роботи бакалавра є дослідження особливостей програмної реалізації кросплатформного мобільного застосунку чатування з оптимізацією обміну медіаконтентом.

Завдання:

- провести аналіз предметної області та наявних програмних рішень у сфері чат-застосунків;
- здійснити проектування архітектури та основних компонентів мобільного застосунку;
- створити програмне забезпечення чат-додатку;
- провести тестування та оцінку функціональності, стабільності й зручності використання розробленого програмного забезпечення.

Рисунок В.3 – Слайд 3

Функціональні вимоги

Система має підтримувати:

- реєстрацію та авторизацію користувачів;
- обмін текстовими повідомленнями в реальному часі;
- надсилання та отримання зображень (в оригінальній та стиснутій якості);
- збереження історії чату;
- відправку скарг на повідомлення;
- блокування та розблокування користувачів;
- зміну теми додатку на світлу та темну.

Рисунок В.4 – Слайд 4

Порівняння існуючих аналогів

Функція	WhatsApp	Telegram	Signal
Наскрізне шифрування	Так	Лише в "Таємних чатах"	Так
Максимальна кількість учасників у групі	До 1 024 учасників	До 200 000 учасників	До 1 000 учасників
Підтримка каналів	Ні	Так	Ні
Редагування повідомлень	Ні	Так	Ні
Відправлення великих файлів	До 2 ГБ	До 2 ГБ	Обмежено
Функція зникнення повідомлень	Так	Так	Так
Персоналізація стікерів	Так	Так	Ні

Рисунок В.5 – Слайд 5

Вибір мови програмування

Критерій	Dart	JavaScript	Kotlin
Продуктивність	Висока	Середня	Висока
Читабельність та синтаксис	Чистий та зрозумілий синтаксис, схожий на C-подібні мови	Гнучкий, але може бути неуніфікованим через динамічну природу	Консистентний та лаконічний синтаксис, схожий на Java
Легкість або швидкість навчання	Легкий для вивчення, особливо для тих, хто знайомий з JavaScript	Легкий для початку, але складний для глибокого опанування через особливості	Помірна складність, особливо для тих, хто знайомий з Java
Підтримка спільноти та екосистема	Зростаюча спільнота, активний розвиток Flutter	Велика спільнота, багата екосистема бібліотек та фреймворків	Широка спільнота, підтримка JetBrains
Сумісність та інтеграція	Добра інтеграція з Flutter, але обмежена з іншими платформами	Висока сумісність з веб-технологіями та серверними платформами	Відмінна інтеграція з Java та Android, підтримка мультиплатформенності
Кросплатформеність	Підтримує мобільні, веб та десктопні платформи через Flutter	Основна сфера – веб, обмежена підтримка мобільних платформ	Підтримує Android та iOS через Kotlin Multiplatform
Наявність інструментів та середовищ розробки	Добра підтримка в Visual Studio Code та Android Studio	Широкий вибір інструментів, включаючи Visual Studio Code, WebStorm	Відмінна інтеграція з Android Studio та IntelliJ IDEA

Рисунок В.6 – Слайд 6

Вибір середовища розробки

Критерій	Visual Studio Code	Android Studio	IntelliJ IDEA
Ресурсоємність	Низька	Висока	Висока
Підтримка Flutter	Через розширення	Є	Є
Продуктивність	Висока	Середня	Середня
Налагодження	Вбудоване	Розширене	Розширене
Вартість	Безкоштовне	Безкоштовне	Платна
Плагіни	Широка підтримка	Обмежена	Розширена

Рисунок В.7 – Слайд 7

Вибір бекенд платформи

Критерій	Firebase	AWS Amplify	Supabase
Функціональність	Широкий спектр BaaS-сервісів, включаючи базу даних реального часу, автентифікацію та хмарні функції	Надає інструменти для розробки та розгортання бекенду з інтеграцією AWS-сервісів	Open-source альтернатива Firebase з підтримкою PostgreSQL та реального часу
Сумісність із Flutter/Dart	Офіційні FlutterFire плагіни забезпечують просту та надійну інтеграцію з Flutter	Підтримка Flutter через Amplify Flutter; потребує додаткового налаштування для повної сумісності	Плагін supabase_flutter надає засоби для інтеграції з Flutter
Простота інтеграції	Легка інтеграція з мобільними та веб-застосунками	Потребує налаштування AWS-сервісів, що може ускладнити інтеграцію	Простий RESTful API для інтеграції
Вартість	Безкоштовний тариф з обмеженнями	Платформа з оплатою за використання AWS-ресурсів	Відкрите ПЗ, витрати на хостинг та обслуговування бази даних
Спільнота та підтримка	Велика спільнота та офіційна підтримка від Google	Активна спільнота AWS та офіційна підтримка	Зростаюча спільнота, підтримка через open-source канали

Рисунок В.8 – Слайд 8

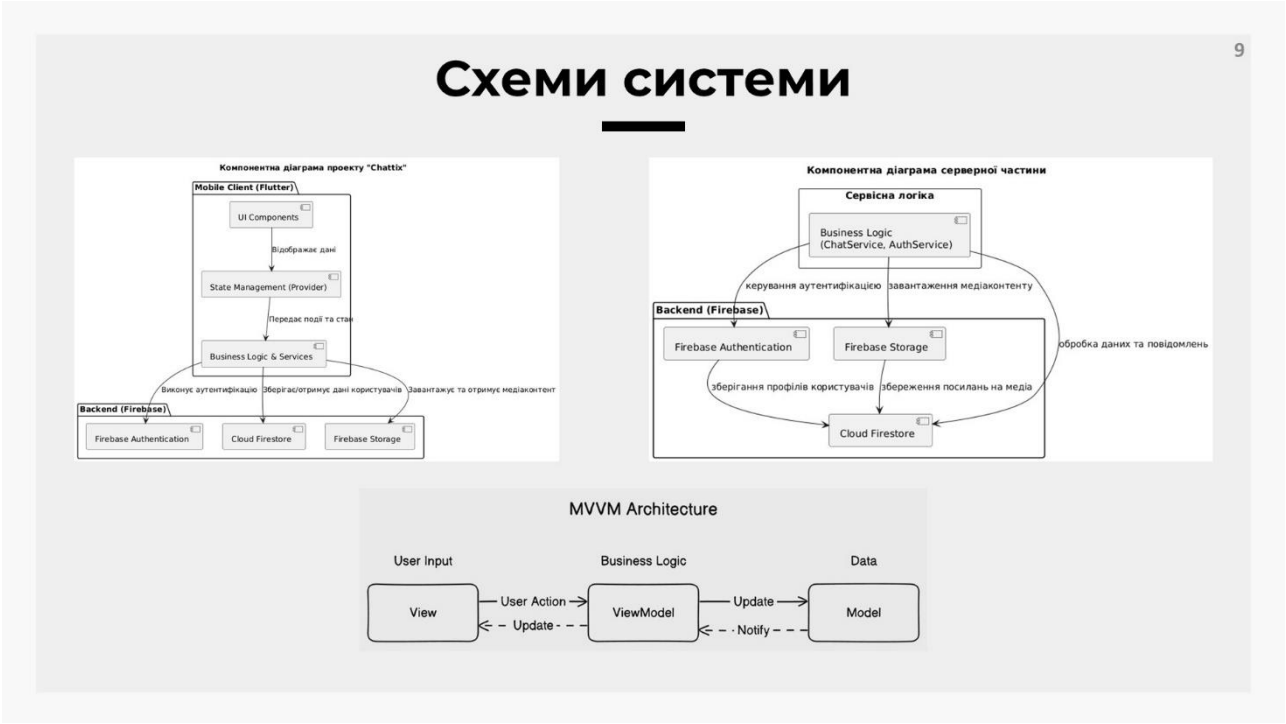


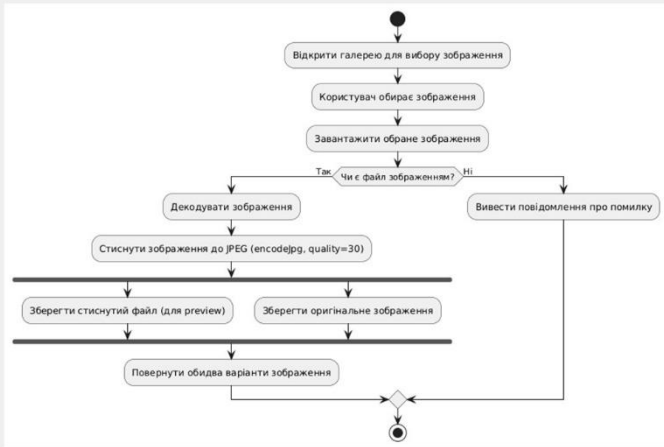
Рисунок В.9 – Слайд 9



Рисунок В.10 – Слайд 10

Фізичне проєктування

11



Блок-схема алгоритму стискання фото

Рисунок В.11 – Слайд 11

Логічне проєктування

12

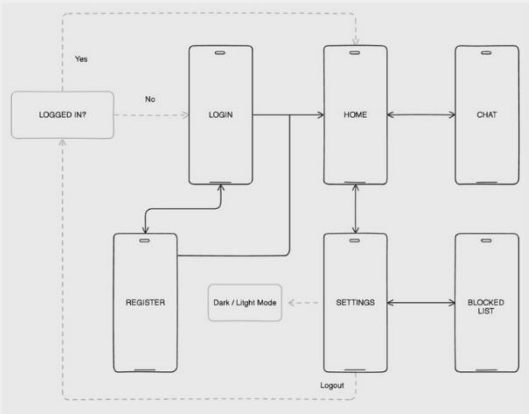


Схема бази даних

Рисунок В.12 – Слайд 12

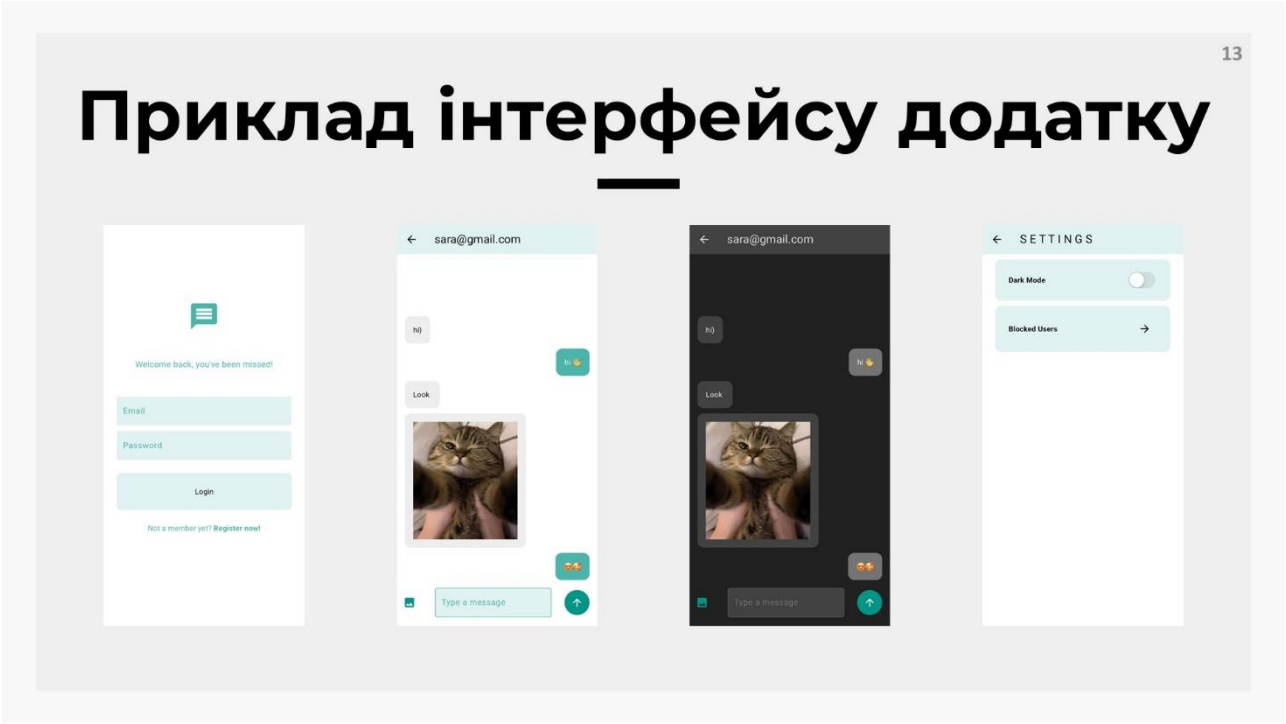


Рисунок В.13 – Слайд 13

14

Функціональне тестування

Функція	Опис тестування	Результат (Android)	Результат (iOS)
Автентифікація користувачів	Перевірка коректності входу та реєстрації з обробкою введених даних	Успішно	Успішно
Приватний чат	Обмін текстовими повідомленнями	Успішно	Успішно
Обмін медіаконтентом	Надсилання зображень, які спочатку відображаються як стислий передпоказ, а при натисканні відкриваються в оригінальному розмірі	Успішно	Успішно
Зміна теми інтерфейсу	Перемикання між світлим та темним режимами інтерфейсу	Успішно	Успішно
Управління контактами – блокування	Перевірка можливості блокування користувачів	Успішно	Успішно
Управління контактами – розблокування	Тестування можливості відновлення спілкування після блокування контакту	Успішно	Успішно
Скарги на повідомлення	Подання скарг на повідомлення	Успішно	Успішно

Рисунок В.14 – Слайд 14

Висновки

В ході виконання дипломної кваліфікаційної роботи бакалавра було реалізовано головну ціль - дослідити особливості програмної реалізації кросплатформного мобільного застосунку чатування з оптимізацією обміну медіаконтентом.

Для цього було виконано такі завдання:

- проведено аналіз предметної області та наявних програмних рішень у сфері чат-застосунків;
- здійснено проектування архітектури та основних компонентів мобільного застосунку;
- створено програмне забезпечення чат-додатку;
- проведено тестування та оцінку функціональності, стабільності й зручності використання розробленого програмного забезпечення.

В результаті виконання завдання було отримано систему, що відповідає всім вимогам представленим в технічному завданні, зручна для використання та виконує всі необхідні функції.

Майбутні вдосконалення системи вбачаються в додаванні групових чатів, підтримці голосових повідомлень, а також покращенні захисту персональних даних та впровадженні штучного інтелекту для розширення функціональності.

Усі завдання випускної дипломної роботи бакалавра повністю виконано.

Рисунок В.15 – Слайд 15