

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Розробка мобільного застосунку для мобільного сервісу»

Виконав студент групи КНк-22-1
ГАЙДАЙ Ігор Ярославович

Керівник роботи:
ГРАФ Марина Сергіївна

Рецензент:
СУГОНЯК Інна Іванівна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«25» січня 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»

(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: **ГАЙДАЙ Ігор Ярославович**

Керівник роботи: **ГРАФ Марина Сергіївна**

Тема роботи: «Розробка мобільного застосунку для мобільного сервісу»,
затверджена Наказом закладу вищої освіти від «25» січня 2024 р. №22/с

Вихідні дані для роботи (зазначається предмет і об'єкт дослідження): об'єктом дослідження є розробка мобільного додатку для мобільного сервісу. Предметом дослідження є використання інструментарію Flutter і Firebase для розробки мобільного додатка.

Консультанти випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1,2	Граф М.С.		

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання задачі. Пошук, огляд та аналіз існуючих аналогічних рішень та їх особливостей. Аналіз літературних джерел.	1 лютого 2024 – 29 лютого 2024	Виконано
2	Проектування мобільного додатку	1 марта 2024 – 31 марта 2024	Виконано
3	Розробка додатку	1 квітня 2024 – 3 травня 2024	Виконано
4	Перевірка функціональності додатку	3 травня 2024 – 15 травня 2024	Виконано
5	Оформлення пояснювальної записки	16 травня 2024 – 8 червня 2024	Виконано
6	Попередній захист	15 червня 2024	Виконано

Здобувач вищої освіти **Ігор ГАЙДАЙ**

Керівник **Марина ГРАФ**

РЕФЕРАТ

Кваліфікаційна робота складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатків.

Текстова частина займає 55 сторінок друкованого тексту.

Пояснювальна записка містить 26 сторінок додатків. Список використаних джерел налічує 25 найменувань і займає 2 сторінки. Робота містить 25 ілюстрацій та 5 таблиць. Загальний обсяг роботи становить 86 сторінки.

Метою цього проекту є створення мобільного додатку, який дозволяє користувачам спілкуватися та обмінюватися даними. Для досягнення цієї мети буде використано сучасні технології, зокрема інтеграцію з Firebase. Додаток забезпечить надійний та швидкий обмін повідомленнями.

Архітектура мобільного додатку ґрунтується на використанні платформи Firebase. Ця платформа забезпечує основні функціональні можливості додатку, включаючи управління базами даних, аутентифікацію користувачів, обробку повідомлень та зберігання файлів. Використання Firebase дозволяє створити надійний додаток.

Додаток відрізняється надзвичайно зручним і інтуїтивно зрозумілим інтерфейсом, розробленим для забезпечення максимально комфортного користувацького досвіду.

КЛЮЧОВІ СЛОВА: ДОДАТОК, FIREBASE, ПОВІДОМЛЕННЯ, ОБМІН, ДАНІ, СПІЛКУВАННЯ.

					КН.КР.Б-122-24-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка мобільного додатку для мобільного сервісу	Літ.	Арк.	Аркушів
Розроб.		І.Г. ГАЙДАЙ						
Керівник		М.С. ГРАФ					3	86
Рецензист		І.І. СУГОНЯК				Житомирська політехніка, група КНк-22-1		
Н. Контр.								
Зав. каф.		М.С. ГРАФ						

ABSTRACT

The qualification work consists of a list of abbreviations, an introduction, three chapters, conclusions, a list of references and appendices.

The textual part takes up 55 pages of printed text.

The explanatory note contains 26 pages of appendices. The list of references includes 25 titles and occupies 2 pages. The work contains 25 illustrations and 5 tables. The total volume of the work is 86 pages.

The purpose of this project is to create a mobile application that allows users to communicate and exchange data. To achieve this goal, modern technologies will be used, including integration with Firebase. The application will provide reliable and fast messaging.

The architecture of the mobile application is based on the Firebase platform. This platform provides the basic functionality of the application, including database management, user authentication, message processing, and file storage. Using Firebase allows you to create a reliable application.

The application has an extremely user-friendly and intuitive interface designed to provide the most comfortable user experience.

KEYWORDS: APPLICATION, FIREBASE, MESSAGES, EXCHANGE, DATA, COMMUNICATION.

					КН.КР.Б-122-24-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ МОБІЛЬНОГО СЕРВІСУ	8
1.1. Постановка задачі	8
1.2. Аналіз аналогів програмного продукту.....	10
1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	15
Висновки до першого розділу	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ	20
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	20
2.2. Розробка бази даних системи	27
2.3. Проектування та реалізація алгоритмів роботи системи.....	31
2.4. Реалізація функціоналу системи мобільного сервісу	33
Висновки до другого розділу.....	44
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МОБІЛЬНИМ ДОДАТКОМ	45
3.1. Порядок встановлення та налаштування параметрів платформи.....	45
3.2. Структура інтерфейсу мобільного додатку	46
3.3. Тестування роботи мобільного додатку	55
Висновки до третього розділу	56
ВИСНОВКИ	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	58
ДОДАТКИ.....	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

UI – англ. user interface, укр. користувацький інтерфейс;

SDK – англ. software development kit, укр. набір засобів розробки програмного забезпечення;

UML – англ. unified modeling language, укр. уніфікована мова моделювання;

API – англ. application programming interface, укр. інтерфейс прикладного програмування.

					КН.КР.Б-122-24-ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми розробки мобільного додатку є надзвичайно високою в сучасному світі, де мобільні технології швидко розвиваються і стають невід'ємною частиною повсякденного життя. Мобільні додатки активно використовуються в різних сферах, таких як комунікація, розваги, освіта, бізнес, охорона здоров'я та багато інших. Завдяки своїй доступності та зручності, мобільні додатки дозволяють користувачам виконувати широкий спектр завдань безпосередньо зі своїх смартфонів або планшетів.

У комунікацій мобільні додатки, такі як месенджери та соціальні мережі, надають можливість людям легко спілкуватися один з одним незалежно від відстані. Це сприяє розвитку соціальних зв'язків та полегшує обмін інформацією.

Метою випускної кваліфікаційної роботи бакалавра є розробка мобільного додатку для мобільного сервісу, що відповідає потребам сучасного користувача в ефективному та зручному використанні. Цей додаток буде спрямований на полегшення процесу обміну повідомленнями, файлами, фотографіями та іншими мультимедійними матеріалами між користувачами, незалежно від їхнього місця знаходження та типу пристрою.

Об'єктом дослідження є процес розробки програмного забезпечення для створення мобільного сервісу.

Предметом дослідження є інструментарій Flutter і Firebase для розробки мобільного додатка мобільного сервісу.

Методи дослідження: аналіз літературних джерел, порівняльний аналіз існуючих рішень, проектування архітектури додатку, програмування, тестування та оцінка продуктивності.

					КН.КР.Б-122-24-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ МОБІЛЬНОГО СЕРВІСУ

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є розробка мобільного додатку для мобільного сервісу.

Мобільний додаток для мобільного сервісу — це програмне забезпечення, розроблене для мобільних пристроїв, яке дозволяє користувачам замовляти та отримувати послуги безпосередньо на їхньому місці перебування.

Існує широкий спектр мобільних сервісів, таких як соціальні мережі, стрімінгові платформи, послуги таксі та фітнес, але ми сфокусуємось на дослідженні мобільного сервісу месенджерів.

Існують кілька аргументів, які підтверджують, що месенджер є мобільним сервісом, включаючи його доступність через мобільні додатки, функціональність, зручність використання в будь-який час і в будь-якому місці та широке поширення серед користувачів мобільних пристроїв.

Розглядаючи мобільний сервіс месенджер, ми зосередимось на платформах, що дозволяють обмінюватися повідомленнями, файлами, а також надають різноманітні можливості спілкування та організації групових чатів. Такі месенджери часто надаються як мобільні додатки, які працюють на смартфонах та планшетах під управлінням різних операційних систем, таких як iOS і Android.

Також на створення месенджера надихнула ця історія.

24 серпня 2016 року в Італії стався сильний землетрус, який забрав життя 299 людей і залишив бездомними понад 4 000 людей. У цій трагічній ситуації мобільний месенджер WhatsApp став рятівним колом для багатьох людей. Мобільні мережі були перевантажені, а телефонні лінії пошкоджені. WhatsApp, завдяки своїй стійкості до перевантажень, став єдиним способом

					КН.КР.Б-122-24-ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

зв'язку для багатьох людей. Люди використовували WhatsApp, щоб дізнатися, чи живі й здорові їхні близькі, щоб повідомити про своє місцезнаходження і щоб попросити про допомогу. WhatsApp також використовувався для координації рятувальних робіт і для поширення інформації про ситуацію.

Для успішної розробки мобільного додатку треба визначити етапи розробки:

1. Аналіз вимог:

- етап передбачає ретельне визначення функціональних та технічних вимог до платформи;
- аналіз аналогів програмного продукту;

2. Вибір та інтеграція технологій:

- дослідження та вибір платформи розробки;
- розгляд інтеграції з існуючими системами чи сервісами;

3. Розробка:

- розробка макету інтерфейсу користувача, визначаються основні елементи дизайну та навігації;
- реалізація функціоналу платформи на основі визначених вимог;

4. Тестування і виправлення помилок:

- після реалізацію функціоналу важливо провести ретельне тестування, виявити та виправити помилки;
- впровадження процесів забезпечення якості для виявлення та вирішення будь-яких проблем або помилок, які можуть виникнути на етапах розробки та тестування;

5. Впровадження і підтримка:

- після тестування додаток можна випустити в експлуатацію;
- забезпечити підтримку та обслуговування платформи;
- збирати фідбек користувачів та вносити необхідні зміни та покращення;

					КН.КР.Б-122-24-ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

Для реалізації функціональності мобільного месенджера можуть бути розроблені різні програмні модулі, які забезпечують різноманітні функції та можливості для користувачів. Деякі з можливих програмних модулів включають:

1. Модуль авторизації та реєстрації: забезпечує можливість створення облікового запису користувача та авторизації за допомогою мобільного телефону для використання месенджера.
2. Модуль обміну повідомленнями: забезпечує функціонал для надсилання та отримання текстових, голосових, відео та інших типів повідомлень.
3. Модуль створення групових чатів: Дозволяє користувачам створювати групові чати, до яких можуть приєднуватися інші користувачі.
4. Модуль пошуку користувачів: надає можливість пошуку та знаходження інших користувачів для подальшої взаємодії.
5. Модуль історії чатів: зберігає історію спілкування користувачів для подальшого перегляду та відновлення.

В результаті розробки мобільного додатка буде створено месенджер, що надасть користувачам не лише обмінюватись текстовими повідомленнями, але й використовувати різноманітні додаткові функції для комунікації.

1.2. Аналіз аналогів програмного продукту

Існує безліч месенджерів для платформи Android, які відрізняються функціоналом, зручністю використання, рівнем безпеки та іншими параметрами, проведемо аналіз наявних популярних месенджерів.

Деякі з найпопулярніших аналогів включають:

Telegram відомий своєю високою швидкістю передачі повідомлень та широким функціоналом. Він підтримує групові чати, канали, шифрування повідомлень, можливість відправки файлів різних форматів (рис. 1.1.).

					КН.КР.Б-122-24-ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

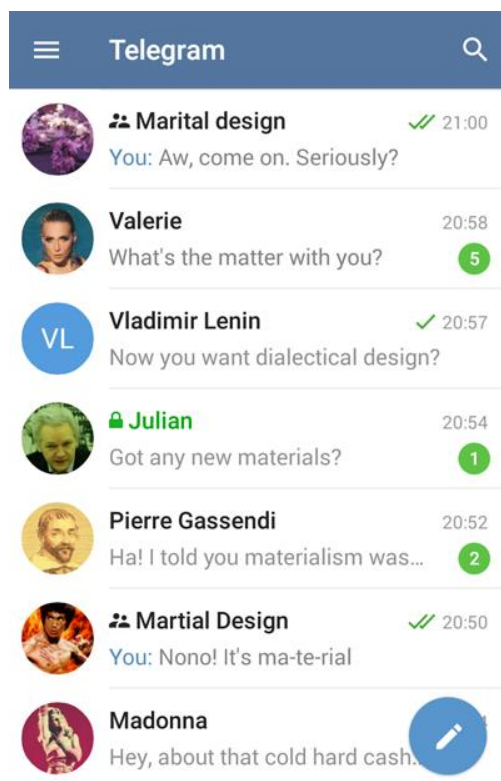


Рисунок 1.1 - Месенджер Telegram

Переваги Telegram:

- висока швидкість роботи;
- функціональність;
- хмарне зберігання;
- безпека;
- приватні чати;
- доступний на різних платформах (IOS, Android, PC);

Недоліки Telegram:

- платний контент
- відсутня цензура
- незручний інтерфейс

WhatsApp є надійним та популярним месенджером, який надає широкий набір функцій для зручного та ефективного спілкування. Використовуючи шифрування end-to-end користувачі можуть не переживати за свою конфіденційність (рис. 1.2.).

					КН.КР.Б-122-24-ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок 1.2 - Месенджер WhatsApp

Переваги WhatsApp:

- шифрування end-to-end означає що лише відправник і одержувач можуть читати повідомлення;
- простий і зрозумілий інтерфейс;
- підтримка на кількох пристроях;
- доступний на різних платформах (IOS, Android, PC);

Недоліки WhatsApp:

- обмежений набір функцій;
- обмеження розміру файлів;
- обмежена персоналізація;
- поширення спаму;

Viber - це потужний та багатофункціональний месенджер, який надає широкий спектр інструментів для спілкування користувачів по всьому світу. Широкий функціонал та простота використання роблять Viber популярним

					КН.КР.Б-122-24-ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

вибором для спілкування як для особистого, так і для корпоративного використання (рис. 1.3.).

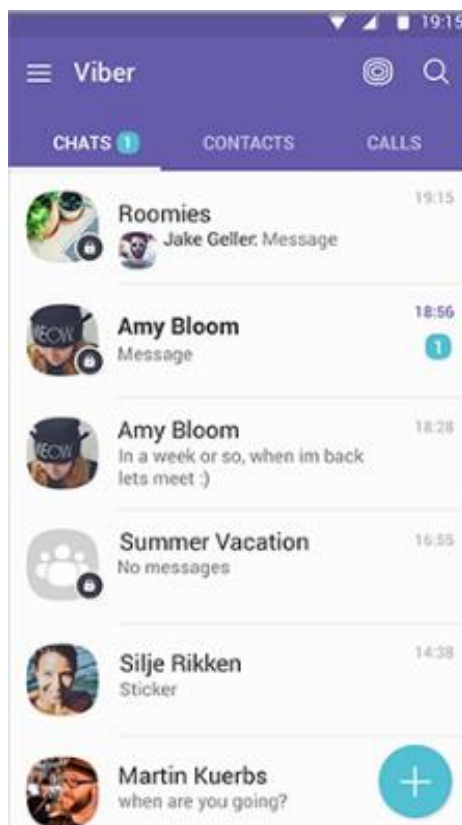


Рисунок 1.3 - Месенджер Viber

Переваги Viber:

- повністю безкоштовний;
- публічні облікові записи;
- зручний інтерфейс;
- доступний на різних платформах (IOS, Android, PC);
- часто використовують компанії для створення робочих груп;

Недоліки Viber:

- реклама;
- відсутність деяких функцій;
- поширення спаму;

Messenger - це месенджер, розроблений компанією Facebook, який користується популярністю серед мільйонів людей по всьому світу. Він пропонує широкий спектр функцій для комунікації, включаючи текстові та

					КН.КР.Б-122-24-ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

голосові повідомлення, відео-дзвінки, групові чати та інші можливості (рис 1.4).

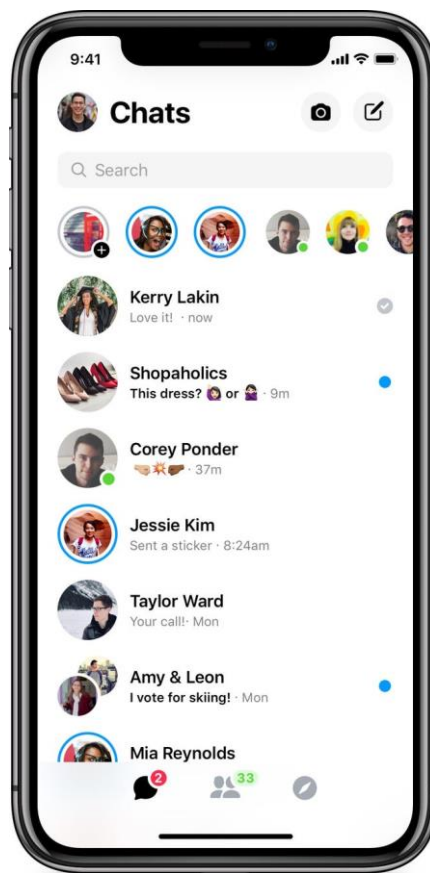


Рисунок 1.4 - Месенджер Messenger

Переваги Messenger:

- повністю безкоштовний
- інтеграція з Facebook
- функціональність;
- зручний інтерфейс;
- доступний на різних платформах (IOS, Android, PC);

Недоліки Messenger:

- реклама;
- необхідність мати обліковий запис Facebook;
- поширення спаму;

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1.).

					КН.КР.Б-122-24-ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця порівняння месенджерів

Характеристика	Telegram	WhatsApp	Viber	Messenger
Платний функціонал	+	-	-	-
Зручність використання(UI)	-	+	+	+-
Кросплатформенність	+	+	+	+
Обмін будь-яких файлів	+	+-	+	+-
Боти	+	-	-	+
Спам	-	+	+	+
Персоналізація	+	-	-	-

При огляді месенджерів можна помітити основні відмінності: деякі мають розгалужений функціонал, інші відрізняються зручним та простим у використанні інтерфейсом, а також деякі забезпечують шифрування типу end-to-end. Крім того, вони різняться за швидкістю роботи, а деякі мають спеціалізовану аудиторію.

Аналіз існуючих месенджерів дозволяє визначити ключові аспекти, які потрібно враховувати при розробці нового програмного продукту. Врахування цих аспектів допоможе забезпечити успішне створення нового месенджера.

Отже основні вимоги до цього додатку включають простоту використання з інтуїтивним дизайном, великим функціоналом. Додаток також має орієнтуватись на широке коло аудиторії, задовольняти потреби користувачів, та забезпечувати миттєвий обмін повідомленнями.

1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.

Месенджер буде побудований за допомогою фреймворк Flutter, який забезпечить швидку та ефективну розробку зручного користувацького інтерфейсу для мобільних пристроїв. В наслідок використання даного фреймворку мова програмування буде використовуватись Dart, синтаксис дуже подібний до JS, при цьому Dart має безліч переваг. До цього якісного поєднання додаємо ще Firebase, на сьогоднішній день це є одним з найбільш успішних поєднань, оскільки використання Firebase дозволяє уникнути потреби у створенні серверного коду.

Flutter – це розроблений Google фреймворк з відкритим програмним кодом, який дозволяє просто і швидко створювати мобільні додатки для iOS і Android. При цьому Flutter не використовує нативні компоненти. Замість цього всі інтерфейс елементи у фреймворку створюються за допомогою власного движка [1].

Переваги Flutter:

- Економічність: дає можливість створити одну програму відразу для декількох ОС.
- Відкритий вихідний код: дає розробникам можливість використовувати його у своїх проектах абсолютно безкоштовно, а також вирішувати проблеми, набагато швидше, за рахунок пакетів ком'юніті.
- Віджети: величезний набір сучасних віджетів, які можна налаштовувати та адаптувати під потреби окремого проекту. Також містить інструменти для розробки власних елементів дизайну.
- Ефективність: Додатки, розроблені за допомогою Flutter, працюють дуже ефективно та плавно. Це досягається завдяки компіляції у нативний ARM код, що забезпечує високу продуктивність навіть при великому графічному навантаженні.
- Миттєве оновлення: Вносите зміни до коду вашого додатку, і результат одразу ж з'явиться на екрані – без перезавантаження, без довгих очікувань. Завдяки Hot Reload ви можете блискавично тестувати нові ідеї, експериментувати з дизайном та функціональністю, економити час і ресурси.

					КН.КР.Б-122-24-ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

- Простота використання: Flutter використовує мову програмування Dart, яка проста в освоєнні та використанні.

Flutter є інструментом високої потужності та гнучкості для розробки мобільних додатків, який пропонує безліч переваг, зокрема кросплатформенність, високу продуктивність та розмаїття бібліотек компонентів.

Мова Dart - це всебічний інструмент, розроблений Google як альтернатива JavaScript. Вона дозволяє не лише писати веб-сайти та мобільні додатки для iOS і Android, а також розробляти утиліти командного рядка і серверні застосунки. В даний момент Dart найчастіше використовується для написання кросплатформенних додатків на Flutter [2,3].

Переваги Dart:

- Гнучкість: код, написаний на Dart, можна компілювати для різних платформ, включаючи iOS, Android, веб, сервери та командний рядок.

- Вбудовані інструменти тестування: Dart має вбудовані інструменти для модульного тестування, що робить процес тестування коду простим і зручним. Вам не потрібно використовувати сторонні бібліотеки чи фреймворки для юніт-тестування.

- Dart – це об'єктно-орієнтована мова, що робить її стійкою до помилок та легкою у налагодженні. Dart підтримує успадкування та інтерфейси, що дозволяє вам використовувати код повторно та створювати більш складні програми.

- Легкість синтаксис: Синтаксис Dart схожий на JavaScript, тому його легко вивчити розробникам, які вже знають ці мови. Нам не потрібно витрачати багато часу, щоб навчитися писати код на Dart. Для початку роботи з Dart нам достатньо вивчити лише базові принципи мови.

Отже, Dart — універсальна і функціональна, але при цьому проста мова програмування, для якої збільшення популярності лише питання часу. На сьогоднішній день, Dart широко використовується для розробки різноманітних програмних продуктів, включаючи серверні, десктопні,

					КН.КР.Б-122-24-ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

мобільні та веб-додатки. Ця мова програмування відрізняється тим, що код, написаний на ній, компілюється в байтовий код JAVA, і на рівні машини фактично майже не відрізняється від нативного коду, що відкриває перед розробниками безліч можливостей.

Firebase – це платформа для розробки мобільних додатків з величезним функціоналом. Починалася вона як стартап, а сьогодні її використовують при розробці кращих кросплатформених додатків. Головна перевага платформи в тому, що вона дозволяє розробнику не відволікатися на створення backend, тобто прихованої від користувача програмної частини проекту, наприклад, серверного коду [4].

Переваги Firebase:

- Параметри бази даних: Firestore і Realtime — це дві можливі бази даних, які ви можете вибрати під час створення свого продукту. Обидві бази даних базуються на хмарі та є базами даних NoSQL, що робить їх гнучкими та масштабованими з точки зору розміру.
- Швидкість розробки: Firebase надає готові рішення для реалізації різних функцій, таких як аутентифікація, збереження даних, аналітика, хостинг та багато іншого. Це дозволяє розробникам прискорити процес розробки та швидко випустити додаток на ринок.
- Аналітика та звіти: Firebase пропонує вбудований інструмент аналітики, який дозволяє відстежувати взаємодію користувачів з додатком та отримувати цінні звіти для подальшого вдосконалення додатку.
- Безкоштовний тариф: Firebase пропонує безкоштовний тариф, який підходить для багатьох проектів, для початківців. Це дозволяє розробникам експериментувати з платформою, не несучи жодних фінансових витрат.

Використання платформи Firebase – це чудове Backend as a Service, яке надає масу можливостей: сервер, базу даних, хостинг, аутенфікацію в одній платформі. Дозволяє розробнику не відволікатись на створення бекенду, а

					КН.КР.Б-122-24-ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

зосередитися на створенні зручного користувацького інтерфейсу чи функцій додатку.

Висновки до першого розділу

У першому розділі визначено ціль кваліфікаційної роботи - розробка мобільного додатку для мобільного сервісу. Було досліджено сутність мобільного сервісу та наведено приклади, а також обґрунтовано вибір месенджера як об'єкту розробки з наведенням аргументів. Описано етапи розробки додатку та визначено можливий функціонал.

Було проведено дослідження аналогічних продуктів, були проаналізовані та порівняні різні месенджери з різними функціональними можливостями та особливостями використання. З цього аналізу були витягнуті основні вимоги до нашого додатку.

На основі цих вимог був складений набір ключових технологій та інструментів для розробки додатку, що забезпечить зручну та ефективну спілкування між користувачами.

					КН.КР.Б-122-24-ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

У цьому розділі ретельно проаналізовано вимоги до програмного продукту, з великим акцентом на виявленні як функціональних, так і нефункціональних вимог. Також були чітко визначені вимоги до часових характеристик реакції додатку на дії користувача, і описані різноманітні сценарії його використання. Для глибшого розуміння структури системи, було включено детальну діаграму елементів системи разом з їх взаємозв'язками та методами [5].

Основна мета аналізу вимог полягає у вивченні потреб та очікувань користувачів щодо функціональності та характеристик мобільного месенджера. За допомогою зворотного зв'язку були з'ясовані функціональні вимоги, які визначають унікальні особливості та можливості, якими має володіти програмний продукт. Крім того, були узяті до уваги нефункціональні вимоги, що охоплюють аспекти, такі як продуктивність, стабільність роботи та зручність використання [6].

Функціональні вимоги:

- надсилання повідомлення: користувачі можуть надсилати текстові, голосові, відео- та фото-повідомлення один одному;
- редагування профілю: користувач може змінити ім'я, аватар, статус;
- групові чати: можливість створювати групові чати для обговорення тем або планування подій, можливість налаштувати групу під свої потреби;
- додавання друзів: можливість відправляти або приймати запити на дружбу з іншими користувачами для подальшої взаємодії;

					КН.КР.Б-122-24-ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

- інтерактивність: підтримка емодзі, стікерів, можливість залишати реакцію під повідомлення, видаляти і пересилати повідомлення;
- перегляд статусу повідомлення: користувач може переглядати статус свої відправлених повідомлень (прочитано, доставлено);
- реєстрація та авторизація користувача: можливість користувачам реєструватись і авторизуватись за допомогою мобільного номеру телефона;

Нефункціональні вимоги:

- ефективне використання ресурсів: забезпечення оптимального використання ресурсів пристрою (батареї, процесор, пам'яті), щоб месенджер не споживав занадто багато енергії та не перенавантажував пристрій;
- сумісність: забезпечення сумісності та однакової функціональності на різних мобільних платформах (iOS, Android), щоб користувачі з будь-якого пристрою могли зручно використовувати месенджер;
- стабільність: система повинна швидко реагувати на дії користувача, забезпечуючи безперебійну та чуйну роботу;
- зручність використання: інтуїтивний та зрозумілий інтерфейс для користувачів різного рівня технічної підготовки;
- сумісність з форматами повідомленнями (фото- і відео-повідомленнями): додаток повинен підтримувати різні формати зображень і відео, щоб користувачі могли легко відправляти свої повідомлення;

Часові характеристики реакції системи:

- миттєва доставка повідомлень: забезпечення надсилання та отримання повідомлень з найменшою можливою затримкою для сприяння швидкій та безперервній комунікації.;
- мінімальна затримка при завантаженні чатів та медіа файлів: чати та медіа файли повинні завантажуватися швидко;

					КН.КР.Б-122-24-ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

- час відгуку на дії користувача: забезпечення миттєвої відповіді на будь-які вхідні сигнали або дії користувача, такі як натискання кнопок або прокручування екрану, для покращення взаємодії та забезпечення плавного та безперервного користувацького досвіду;
- швидкість відкриття додатку: максимально швидке відкриття месенджера для користувача без відчутної затримки, щоб забезпечити швидкий доступ до функціональності та зручний початок взаємодії з додатком;
- час виконання операцій: мінімізація часу, необхідного для виконання різних операцій у месенджері, створення групових чатів, додавання в друзі, реакція на повідомлення та інші дії, щоб користувачі могли швидко та ефективно взаємодіяти з додатком;

Важливо враховувати, що реакція додатку залежить від різноманітних факторів, таких як операційна система, характеристики пристрою та розмір екрану. Передбачення реакції на всі ці фактори може бути складним завданням, оскільки вони впливають на виконання додатку по-різному [7].

За допомогою тестування проводиться перевірка всіх характеристик реакції додатку, включаючи швидкість відкриття, час виконання операцій та миттєву відповідь на дії користувача. Тестування дозволяє ідентифікувати можливі проблеми з продуктивністю та швидкодією додатку на різних операційних системах, пристроях та екранах. Шляхом виконання різних тестових сценаріїв на різних платформах та пристроях, тестування допомагає виявити та виправити можливі проблеми, що впливають на реакцію додатку, забезпечуючи кращий користувацький досвід [8].

Варіанти використання моделюють послуги, завдання та функції, які система повинна виконувати. Вони представляють функціональні можливості та демонструють, як користувач буде взаємодіяти з системою. Варіанти використання є основою уніфікованої мови моделювання та відображають ключові концепції системи [9].

					КН.КР.Б-122-24-ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Діаграма використання месенджера показує, як система взаємодіє з акторами, тобто користувачами месенджера, і що саме вона робить, але не розкриває внутрішньої роботи системи. Ця діаграма допомагає визначити контекст та вимоги до всієї системи або до її критичних компонентів, а актори виступають у ролі користувачів месенджера, які взаємодіють з системою через різні сценарії використання [10].

Діаграма використання для нашого проекту (рис. 2.1) – взаємодія між актором (користувач) і системою та специфікація прецедентів (дод. А).



Рисунок 2.1 – Діаграма варіантів використання.

Користувач:

- перегляд профілю іншого користувача: перегляд профілю, взаємодія з користувачем, такі як відправити або прийняти запит на дружбу;
- перегляд профілю поточного користувача: перегляд свого профілю, зміна налаштування додатку, зміна даних профілю, перегляд списку друзів;
- надсилання повідомлення: можна відправити текстові, відео-, аудіо- і фото-повідомлення, додавання реакції на повідомлення використовуючи бібліотеку з емодзі;
- створення групового чату: обов'язково треба вказати ім'я і опис групи, також треба витрати тип групи (приватний, публічний), додатково можна додати користувачів і налаштувати групу під свої потреби;
- авторизація за допомогою мобільного номеру;
- реєстрація за допомогою мобільного номеру;
- вихід з облікового запису;

Виокремлені функціональні можливості для користувача утворюють повноцінне рішення, яке відповідає різноманітним потребам.

Загалом, діаграма варіантів використання надає докладне та зрозуміле уявлення про функціональні можливості. Шляхом визначення та документування різних варіантів використання ця діаграма стає корисним інструментом для аналізу системи, збору вимог та проектування.

Структура дозволяє розробляти систему відповідно до модульного підходу, де кожен елемент виступає як об'єкт з чітко визначеними характеристиками та можливостями.

Діаграма класів - це графічна нотація, яка використовується для побудови та візуалізації об'єктно-орієнтованих систем. Ця статична структурна діаграма демонструє властивості системи, класи, операції та зв'язки між об'єктами для опису структури системи [11]. В наведеній діаграмі класів (рис 2.2) використовуються такі зв'язки:

					КН.КР.Б-122-24-ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

- Узагальнення - це відношення в таксономії, яке об'єднує більш загальний класифікатор з більш конкретним класифікатором. Конкретний класифікатор успадковує характеристики більш загального класифікатора [12].
- Асоціації - це зв'язки між класами на діаграмі класів UML. Вони відображаються суцільною лінією між класами. Зазвичай асоціації називаються за допомогою дієслова або дієслівної фрази [12].
- Композиція - зв'язок відображається суцільною лінією із зафарбованим ромбом на кінці асоціації, яка пов'язана з класом, який представляє ціле або складене. Особливий тип агрегації, де частини руйнуються, коли руйнується ціле [12].

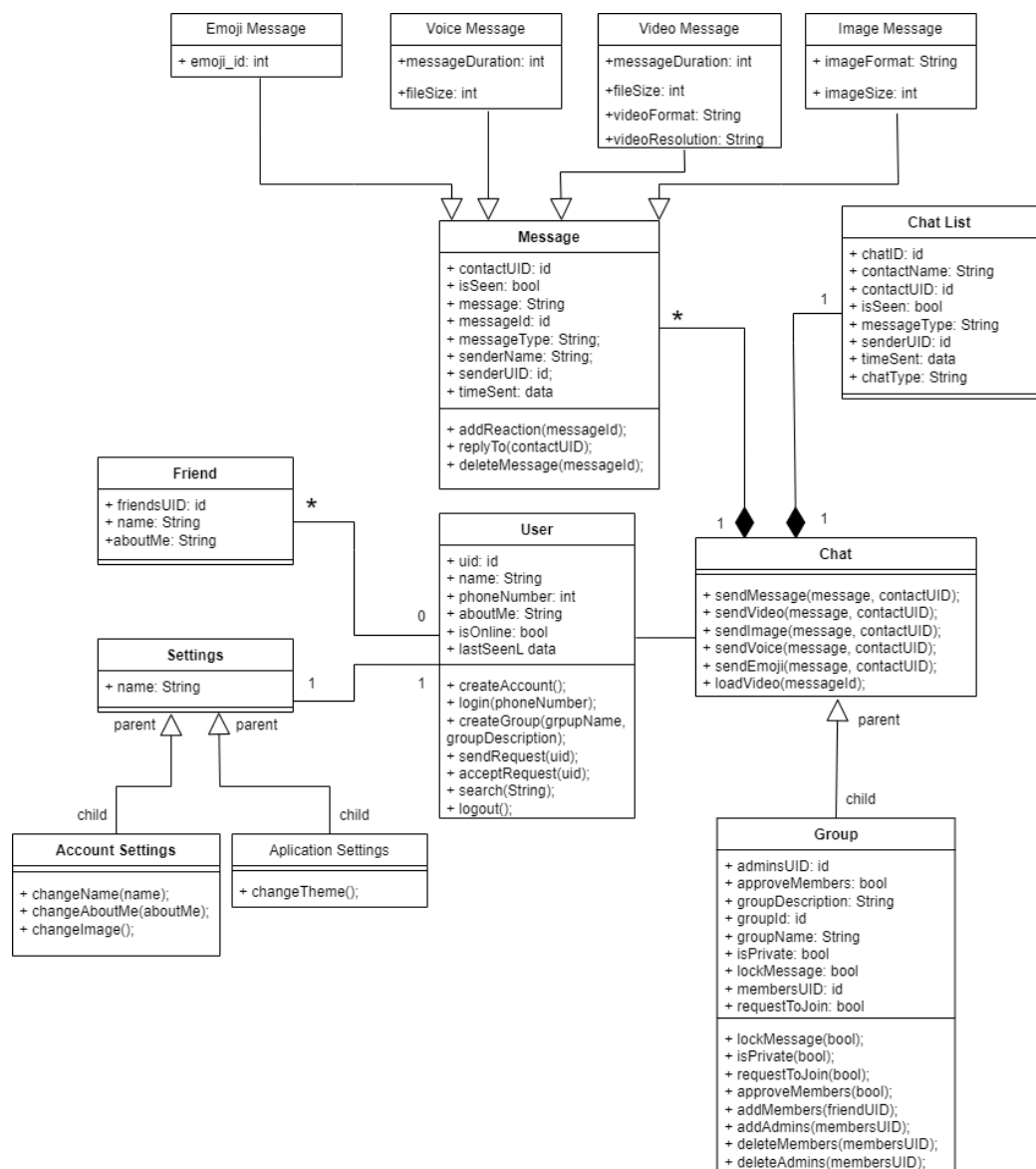


Рисунок 2.2 – Діаграма класів

- User - це клас, який відповідає за користувача. Методи включають createAccount() для створення облікового запису, login() для входу в обліковий запис, createGroup() для створення групового чату, sendRequest() для відправлення запиту в друзі або вступу в груповий чат, acceptRequest() для прийняття запиту в друзі, search() для пошуку користувача або існуючого чату в історії і logout() для виходу з облікового запису.
- Settings - це клас для налаштування облікового запису користувача і додатку.
- Chat - клас, що використовується для взаємодії з чатом, надсилання різних типів повідомлень і взаємодії з ними.
- Chat list - це клас, який повертає список чатів користувача.
- Message - клас, який відповідає за повідомлення в чаті. Методи включають addReaction() для додавання реакції до повідомлення, replyTo() для відправлення повідомлення отримувачу, deleteMessage() для видалення поточного повідомлення.
- Group - клас, який відповідає за роботу з груповим чатом і його налаштування. Методи включають lockMessage() для блокування надсилання повідомлень іншим користувачам, isPrivate() для зміни типу групи, requestToJoin() для вимоги від користувача надіслати запит на вступ, approveMembers() для перевірки, чи є користувач членом групи, addMembers() для додавання членів групи зі списку друзів користувача, addAdmins() для додавання адміністраторів зі списку членів групи, deleteMembers() для видалення членів групи та deleteAdmins() для видалення адміністраторів групи.

Патерн проектування - це стандартний метод вирішення типових проблем, які часто виникають при проектуванні архітектури програмного забезпечення [13].

Патерни які були використані в проектування:

Фасад - це зручний інтерфейс, який спрощує роботу зі складною підсистемою, що складається з великої кількості класів [14]. Він

					КН.КР.Б-122-24-ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

використовується в класі “Chat” надаючи зручний спосіб взаємодії з системою повідомлень.

Ланцюжок обов'язків - це поведінковий патерн проектування, який дозволяє передавати запити послідовно через ланцюжок обробників [15]. Він використовується в класі “Message” для обробки різних типів повідомлень.

2.2. Розробка бази даних системи

Для збереження інформації в додатку використовується Firestore Database. Firestore є NoSQL базою даних, що означає, що вона зберігає дані в документах, які складаються з колекцій ключ-значення, це робить її гнучкою та придатною для зберігання різних типів даних [16].

База даних складається з колекцій і підколекцій які створюють ієрархічні структуру даних:

1. Users – колекція, яка зберігає дані користувачів;
 - Chat – підколекція, яка зберігає дані чатів користувачів;
 - Messages – підколекція, яка зберігає дані повідомлень в чаті;
2. Groups – колекція, яка зберігає дані групових чатів;
 - Messages – підколекція, яка зберігає дані повідомлення в групових чатах;

Опис кожної з колекцій бази даних наведений у таблицях 2.1 – 2.4.

Таблиця 2.1

Опис колекції Users:

Поле	Тип	Призначення
uid	string	Ідентифікатор
sentFriendRequestUID	array	Масив ідентифікаторів користувачів, яким надіслав запити на дружбу
phoneNumber	int	Телефонний номер користувача
name	string	Ім'я користувача

lastSeen	data	Дата та час останньої активності користувача
isOnline	bool	Чи користувач зараз онлайн
image	string	URL-адреса зображення профілю користувача
friendsUIDs	array	Масив ідентифікаторів користувачів, які є друзями
friendsRequesrsUIDs	array	Масив ідентифікаторів користувачів, які надіслали запити на дружбу
created	data	Дата та час створення користувача
aboutMe	string	Короткий опис користувача

Таблиця 2.2

Опис колекції Chat

Поле	Тип	Призначення
contactImage	string	Зображення контакту
contactName	string	Ім'я контакту
contactUID	string	Ідентифікатор контакту
isSeen	bool	Чи повідомлення побачене
message	string	Останнє повідомлення
messageType	string	Тип повідомлення
senderUID	string	Ідентифікатор користувача який надіслав повідомлення
timeSent	data	Дата та час надсилання повідомлення

Таблиця 2.3

Опис колекції Message

Поле	Тип	Призначення
contactUID	string	Ідентифікатор отримувача
deleteBy	array	Масив користувачів, які видалили повідомлення

message	string	Вміст повідомлення
messageId	string	Ідентифікатор повідомлення
messageType	string	Тип повідомлення
isSeen	bool	Чи побачили повідомлення
reactions	array	Масив реакцій
senderImage	string	Зображення відправника
senderName	string	Ім'я відправника
senderUID	string	Ідентифікатор відправника
timeSent	data	Дата та час надсилання повідомлення

Таблиця 2.4

Опис колекції Groups

Поле	Тип	Призначення
adminUID	string	Масив ідентифікаторів, які є адміністраторами групи
approveMembers	bool	Чи є користувач учасником групи
awaitingApprovalUID	array	Масив ідентифікаторів користувачів, які подали заявки на вступ до групи
created	data	Дата та час створення
creatorUID	string	Ідентифікатор користувача який створив групу
editSettings	bool	Чи дозволено членам групи редагувати налаштування групи
gpoupDescription	string	Опис групи
groupId	string	Ідентифікатор групи
groupImage	string	Зображення групи
groupName	string	Назва групи
isPrivate	bool	Чи група приватна
lastMessage	string	Останнє повідомлення
membersUID	array	Масив учасників групи
messageId	string	Ідентифікатор останнього повідомлення

messageType	string	Тип останнього повідомлення
requestToJoin	bool	Чи є запит на вступ до групи
senderUID	string	Ідентифікатор відправника
timeSent	data	Дата та час останнього повідомлення

Користувачі можуть створювати облікові записи з ім'ям та номером телефону. При створенні облікового запису генерується унікальний ідентифікатор користувача. Для кожного користувача зберігається інформація про його профіль, таку як ім'я, номер телефону, посилання на зображення профілю, короткий опис (aboutMe), дата та час створення облікового запису, також йде перевірка статус онлайн (isOnline). Користувачі можуть надсилати та отримувати запити на дружбу. Зберігаються також масиви унікальних ідентифікаторів користувачів, які є друзями, а також масиви унікальних ідентифікаторів користувачів, яким були надіслані запити на дружбу.

Користувачі можуть обмінюватися текстовими повідомленнями, зображеннями та відео. Кожне повідомлення має свій унікальний ідентифікатор (messageId). Повідомлення містять інформацію про відправника (senderName, senderUID, senderImage), отримувача (contactName, contactUID, contactImage), тип повідомлення (messageType), зміст повідомлення (message), час надсилання (timeSent). Повідомлення також може мати відповідь на інше повідомлення (replyMessage), в якому зберігається зміст відповіді, тип повідомлення-відповіді (replyMessageType) та унікальний ідентифікатор вихідного повідомлення (replyTo).

Користувачі можуть створювати групи для спільного обговорення. Кожна група має свій унікальний ідентифікатор (groupId). Для кожної групи зберігається інформація про назву (groupName), опис (groupDescription), зображення групи (groupImage), дату та час створення (createdAt), статус приватності (isPrivate), можливість редагування налаштувань групи членами ("editSettings"), блокування надсилання повідомлень (lockMessages). Для

					КН.КР.Б-122-24-ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

кожної групи зберігаються інформація про адміністраторів (adminsUID), членів групи (membersUID), а також користувачів, що подали заявки на вступ (awaitingApprovalUID). Кожна група має останнє повідомлення (lastMessage), яке зберігається разом з інформацією про відправника (senderUID) та час надсилання (timeSent).

2.3. Проектування та реалізація алгоритмів роботи системи

Додаток легкий у використанні і має всі основні функції платформи, що дозволяє користувачам легко взаємодіяти з ним. Алгоритм роботи з платформою показано на рисунці 2.3 і детально описано його кроки.

1. Вхід до системи:

- якщо користувач має обліковий запис він може авторизуватись;
- якщо користувач не має облікового запису йому потрібно реєструватись.

2. Авторизація:

- користувач вводить свої дані для входу в систему;
- якщо дані правильні, користувач авторизується в системі і переходить в головне меню;
- якщо дані неправильні, користувач залишається у вікні авторизації.

3. Реєстрація:

- користувач вводить дані для реєстрації в систему;
- якщо дані правильні, користувач реєструється в системі і переходить в головне меню;
- якщо дані неправильні, користувач залишається у вікні реєстрації.

4. Після входу в систему або реєстрації користувач обирає в меню один із варіантів: чат, групи або користувачі, для подальшої взаємодії з додатком.

5. Чати:

- користувач може надіслати повідомлення в різних форматах (текстові, відео, аудіо, зображень) іншим користувачам;

					КН.КР.Б-122-24-ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

- користувач може додати реакцію на повідомлення.
6. Групи:
- користувач має можливість створити власну групу;
 - користувач може приєднатись до існуючої групи.
7. Користувачі:
- користувач може переглянути профіль іншого користувача;
 - користувач може надіслати запит на дружбу іншому користувачеві;
 - користувач може видалити друга зі списку друзів.

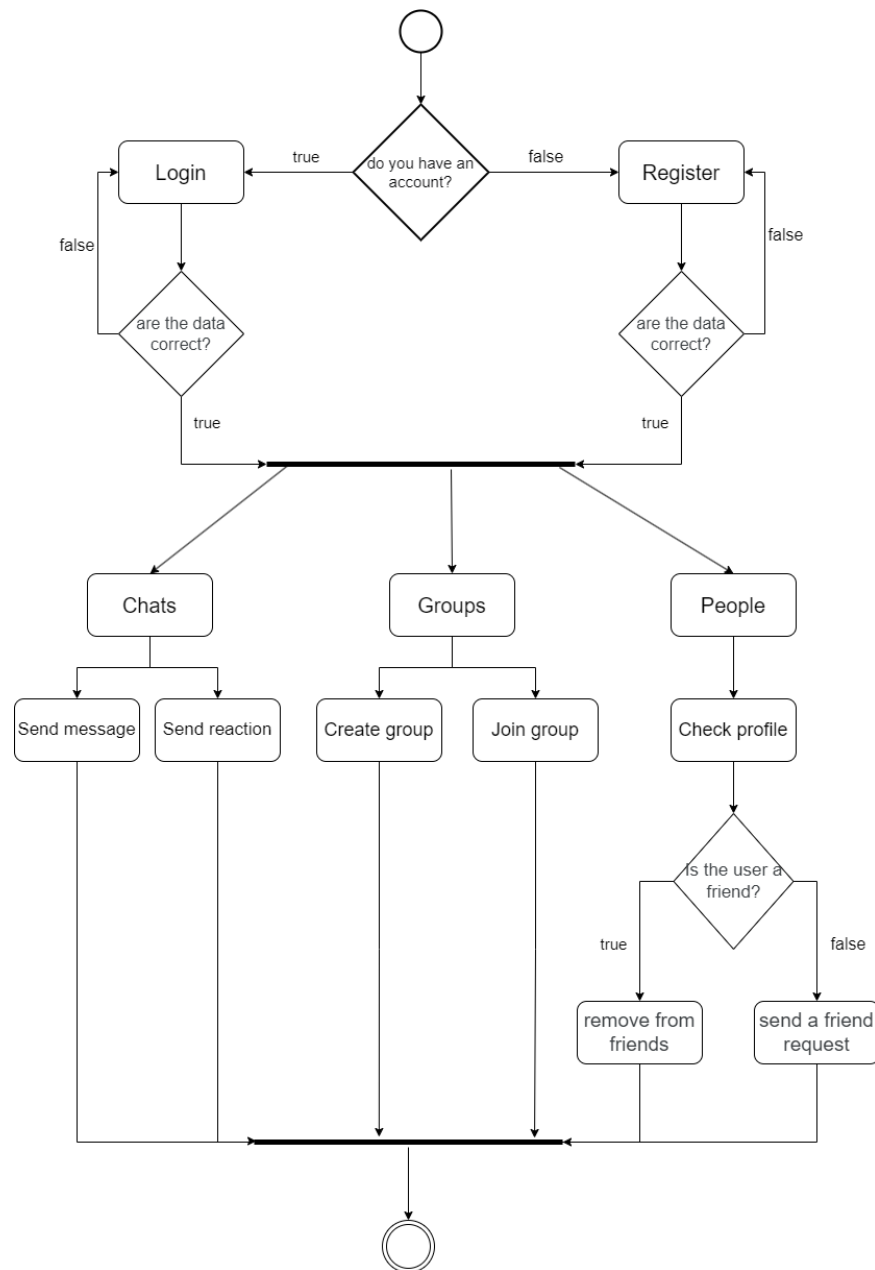


Рисунок 2.3 – Діаграма активності

2.4. Реалізація функціоналу системи мобільного сервісу

В наших функціях ми постійно використовуємо Constants для зберігання констант, таких як назви колекцій, полів у базі даних, назви різних типів повідомлень тощо. Використання Constants дозволяє уникнути хардкоду і полегшити його рефакторинг[17].

Щодо шифрування та безпеки даних, Firebase бере на себе цю відповідальність, і для передачі використовується протокол HTTPS [18].

HTTPS - це протокол передачі гіпертексту, який забезпечує безпечний обмін даними між веб-браузером користувача та веб-сайтом. Онлайн-комунікації за допомогою HTTPS зашифровані шифром SSL або TLS, що забезпечує конфіденційність та цілісність даних [19].

Тепер розглянемо основні функції додатку. По-перше, користувач повинен зареєструвати новий акаунт. Зареєструватись можна тільки за номером телефону. Використання номера телефону для реєстрації дозволяє забезпечити додатковий рівень безпеки, оскільки потрібно мати доступ до конкретного номера телефону для підтвердження особистості. Firebase Authentication надає прості у використанні SDK для автентифікації користувачів [20]. Користувач вводить номер телефону після чого викликається метод `_auth.verifyPhoneNumber()`, який перевіряє номер телефону. У разі успішної перевірки `uid` і номер телефону зберігаються. Якщо код підтвердження було відправлено функція перенаправить користувача на екран введення коду для підтвердження. Функція для входу за допомогою мобільного телефону:

```
Future<void> signInWithPhoneNumber({
  required String phoneNumber,
  required BuildContext context,
}) async {
  _isLoading = true;
  notifyListeners();

  await _auth.verifyPhoneNumber(
    phoneNumber: phoneNumber,
```

```

verificationCompleted: (PhoneAuthCredential credential) async {
    await _auth.signInWithCredential(credential).then((value) async {
        _uid = value.user!.uid;
        _phoneNumber = value.user!.phoneNumber;
        _isSuccessfull = true;
        _isLoading = false;
        notifyListeners();
    });
},
verificationFailed: (FirebaseAuthException e) {
    _isSuccessfull = false;
    _isLoading = false;
    notifyListeners();
    showSnackBar(context, e.toString());
},
codeSent: (String verificationId, int? resendToken) async {
    _isLoading = false;
    _resendToken = resendToken;
    notifyListeners();
    Navigator.of(context).pushNamed(
        Constants.otpScreen,
        arguments: {
            Constants.verificationId: verificationId,
            Constants.phoneNumber: phoneNumber,
        },
    );
},
codeAutoRetrievalTimeout: (String verificationId) {},
timeout: const Duration(seconds: 60),
forceResendingToken: resendToken,
);
}

```

На екрані підтвердження користувач вводить код, який йому прийшов. Після вводу функція перевіряє чи є такий користувач в БД, якщо ні, то користувач переходить на екран створення облікового запису.

Функція для підтвердження:

```

Future<void> verifyOTPCode({
    required String verificationId,
    required String otpCode,
    required BuildContext context,
    required Function onSuccess,
}) async {
    _isLoading = true;
    notifyListeners();

    final credential = PhoneAuthProvider.credential(
        verificationId: verificationId,

```

					КН.КР.Б-122-24-ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        smsCode: otpCode,
    );

    await _auth.signInWithCredential(credential).then((value) async {
        _uid = value.user!.uid;
        _phoneNumber = value.user!.phoneNumber;
        _isSuccessfull = true;
        _isLoading = false;
        onSuccess();
        notifyListeners();
    }).catchError((e) {
        _isSuccessfull = false;
        _isLoading = false;
        notifyListeners();
        showSnackBar(context, e.toString());
    });
}

```

Після реєстрації він може розпочати спілкування, додаючи інших користувачів до свого списку друзів. Для цього він переходить на профіль іншого користувача та надсилає запит на дружбу.

Функція звертається до колекції користувачів у Firestore за допомогою ідентифікатора друга (friendID). Після цього оновлюється документ користувача з ідентифікатором friendID, додаючи ідентифікатор користувача, який відправляє запит на дружбу, до списку запитів на дружбу цього користувача. Для цього використовується метод arrayUnion, щоб додати новий ідентифікатор користувача до існуючого списку. Потім функція оновлює документ користувача, який відправляє запит на дружбу, додаючи ідентифікатор друга до списку відправлених запитів:

```

Future<void> sendFriendRequest({
    required String friendID,
}) async {
    try {
        // add our uid to friend`s request list
        await _firestore.collection(Constants.users).doc(friendID).update({
            Constants.friendRequestsUIDs: FieldValue.arrayUnion([_uid])
        });

        // add friend uid to our friend requests send list
        await _firestore.collection(Constants.users).doc(_uid).update({
            Constants.sentFriendRequestsUIDs: FieldValue.arrayUnion([friendID])
        });
    } on FirebaseException catch (e) {

```

					КН.КР.Б-122-24-ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

```
print(e);
```

Після надсилання запиту інший користувач повинен його прийняти.

Функція додає ідентифікатор друга до списку друзів поточного користувача:

```
Future<void> acceptFriendRequest({required String friendID}) async {  
  await _firestore.collection(Constants.users).doc(friendID).update({  
    Constants.friendsUIDs: FieldValue.arrayUnion([_uid])  
  });  
  await _firestore.collection(Constants.users).doc(_uid).update({  
    Constants.friendsUIDs: FieldValue.arrayUnion([friendID])  
  });  
  await _firestore.collection(Constants.users).doc(friendID).update({  
    Constants.sentFriendRequestsUIDs: FieldValue.arrayRemove([_uid])  
  });  
  await _firestore.collection(Constants.users).doc(_uid).update({  
    Constants.friendRequestsUIDs: FieldValue.arrayRemove([friendID])  
  });  
}
```

Коли інший користувач приймає запит на дружбу, ми можемо переглянути список друзів на своєму профілі і побачити нового друга. Після цього ми можемо зайти на його профіль і розпочати взаємодію. У користувача є кілька варіантів: видалити з друзів або перейти до чату з цим другом.

Коли користувач заходить у чат, він може надсилати різні типи повідомлень, такі як відео, аудіо, фото та текстові повідомлення. Для цього розроблені відповідні функції.

За допомогою бібліотеки `uuid` генерується унікальний `messageId`. Потім створюється об'єкт `messageModel` з необхідними даними для повідомлення. Функція `handleContactMessage` викликається, передаючи потрібні дані. Функцію для надсилання файлових повідомлень можна переглянути за посиланням у додатку Б. Функція для надсилання текстових повідомлень:

```
Future<void> sendTextMessage({  
  required UserModel sender,  
  required String contactUID,  
  required String contactName,  
  required String contactImage,  
  required String message,  
  required MessageEnum messageType,  
  required String groupId,  
  required Function onSuccess,  
  required Function(String) onError,  
})
```

					КН.КР.Б-122-24-ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

```

})) async {
  // set loading to true
  setLoading(false);
  try {
    var messageId = const Uuid().v4();
    String replyMessage = _messageReplyModel?.message ?? '';
    String replyTo = _messageReplyModel == null
      ? ''
      : _messageReplyModel!.isMe
        ? 'Тя'
        : _messageReplyModel!.senderName;
    MessageEnum replyMessageType =
      _messageReplyModel?.messageType ?? MessageEnum.text;
    final messageModel = MessageModel(
      message: message,
      senderUID: sender.uid,
      senderName: sender.name,
      senderImage: sender.image,
      contactUID: contactUID,
      messageType: messageType,
      timeSent: DateTime.now(),
      messageId: messageId,
      isSeen: false,
      replyMessage: replyMessage,
      replyTo: replyTo,
      replyMessageType: replyMessageType,
      reactions: [],
      isSeenBy: [sender.uid],
      deletedBy: [],
    );
    if (groupId.isNotEmpty) {
      await _firestore
        .collection(Constants.groups)
        .doc(groupId)
        .collection(Constants.messages)
        .doc(messageId)
        .set(messageModel.toMap());
      await _firestore.collection(Constants.groups).doc(groupId).update({
        Constants.lastMessage: message,
        Constants.timeSent: DateTime.now().millisecondsSinceEpoch,
        Constants.senderUID: sender.uid,
        Constants.messageType: messageType.name,
      });
      setMessageReplyModel(null);
    } else {
      await handleContactMessage(
        messageModel: messageModel,
        contactUID: contactUID,
        contactName: contactName,
        contactImage: contactImage,

```

					КН.КР.Б-122-24-ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        onSuccess: onSuccess,
        onError: onError,
    );
    setMessageReplyModel(null);
}
} catch (e) {
    setLoading(false);
    onError(e.toString());
}
}
}

```

Також користувач може пересилати повідомлення і його видаляти.

Функція видаляє повідомлення з бази даних Firestore. Вона приймає на вхід інформацію про користувача (`currentUserId`), інформацію про контакт, з яким відбувається спілкування (`contactUID`), ідентифікатор повідомлення (`messageId`), тип повідомлення (`messageType`) та інші необхідні параметри. Спочатку вона оновлює інформацію про видалення повідомлення в колекції `messages` для користувача з `currentUserId`. Якщо `deleteForAll` - `false`, функція завершує роботу. Якщо `deleteForAll` - `true`, вона оновлює інформацію про видалення повідомлення для контакту з `contactUID`. Якщо `messageType` не є текстовим, функція також видаляє файли збережені на сервері, викликавши іншу функцію `deleteFileFromStorage`.

Функція для видалення повідомлень:

```

Future<void> deleteMessage({
  required String currentUserId,
  required String contactUID,
  required String messageId,
  required bool isGroup,
  required String messageType,
  required bool deleteForAll,
}) async {
  setLoading(true);
  if (isGroup) {
    await _firestore
      .collection(Constants.groups)
      .doc(contactUID)
      .collection(Constants.messages)
      .doc(messageId)
      .update({
        Constants.deletedBy: FieldValue.arrayUnion([currentUserId])
      });
  }
  if (deleteForAll) {

```

					КН.КР.Б-122-24-ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        final groupData =
            await
            _firestore.collection(Constants.groups).doc(contactUID).get();
        final List<String> groupMembers =
            List<String>.from(groupData.data()![Constants.membersUID]);
        await _firestore
            .collection(Constants.groups)
            .doc(contactUID)
            .collection(Constants.messages)
            .doc(messageId)
            .update({Constants.deletedBy:
FieldValue.arrayUnion(groupMembers)});
        if (messageType != MessageEnum.text.name) {
            await deleteFileFromStorage(
                currentUserId: currentUserId,
                contactUID: contactUID,
                messageId: messageId,
                messageType: messageType,
            );
        }
    }
    setLoading(false);
} else {
    await _firestore
        .collection(Constants.users)
        .doc(currentUserId)
        .collection(Constants.chats)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({
            Constants.deletedBy: FieldValue.arrayUnion([currentUserId])
        });
    if (deleteForAll) {
        await _firestore
            .collection(Constants.users)
            .doc(contactUID)
            .collection(Constants.chats)
            .doc(currentUserId)
            .collection(Constants.messages)
            .doc(messageId)
            .update({
                Constants.deletedBy: FieldValue.arrayUnion([contactUID])
            });
    }
    if (messageType != MessageEnum.text.name) {
        await deleteFileFromStorage(
            currentUserId: currentUserId,
            contactUID: contactUID,
            messageId: messageId,

```

					КН.КР.Б-122-24-ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        messageType: messageType,
    );
}
setLoading(false)

```

Функція setMessageReplyModel приймає аргумент messageReply, який може бути об'єктом типу MessageReplyModel або значенням null. Змінна _messageReplyModel, яка зберігає поточну модель відповіді на повідомлення, встановлюється рівною значенню messageReply. Після встановлення значення _messageReplyModel викликається метод notifyListeners(). Функція для пересилання повідомлення:

```

void setMessageReplyModel(MessageReplyModel? messageReply) {
    _messageReplyModel = messageReply;
    notifyListeners();
}

```

Давайте розглянемо процес створення груп та їх налаштування більш детально. Для створення групи необхідно вказати назву та опис обов'язково, а додавання зображення є опціональним. Також потрібно додати учасників групи, які обираються виключно із списку друзів.

Процес створення групи розпочинається з встановлення значення isLoading в true для відображення початку процесу. За допомогою пакету uuid генерується унікальний ідентифікатор для нової групи. Якщо зображення групи існує, воно завантажується до сховища, і URL зберігається в об'єкті newGroup. Творець групи та інші адміністратори та учасники додаються до списку адміністраторів та учасників групи. Об'єкт newGroup зберігається в колекції groups з ідентифікатором groupId.

Функція створення групи:

```

Future<void> createGroup({
    required GroupModel newGroup,
    required File? fileImage,
    required Function onSuccess,
    required Function(String) onFall,
}) async {
    setIsLoading(value: true);
    try {
        var groupId = const Uuid().v4();
        newGroup.groupId = groupId;
        if (fileImage != null) {

```

```

        final String imageUrl = await storeFileToStorage(
            file: fileImage, reference:
            '${Constants.groupImages}/${groupId}');
        newGroup.groupImage = imageUrl;
    }
    newGroup.adminsUID = [newGroup.creatorUID, ...getGroupAdminsUid()];
    newGroup.membersUID = [newGroup.creatorUID, ...getGroupMembersUid()];
    setGroupModel(groupModel: newGroup);
    await _firestore
        .collection(Constants.groups)
        .doc(groupId)
        .set(groupModel.toMap());
    setIsLoading(value: false);
    onSuccess();
} catch (e) {
    setIsLoading(value: false);
    onFall(e.toString());
}
}
}

```

Після створення групи, можна додавати нових користувачів і змінювати налаштування.

Функція додавання нових учасників і адміністраторів працює ідентично. При додаванні користувача його uid додається до списку membersUID, а при додаванні адміністратора - до списку adminsUID.

Функції додавання адміністраторів і учасників групи:

```

void addGroupMember({required UserModel groupMember}) {
    _groupMembers.add(groupMember);
    _groupModel.membersUID.add(groupMember.uid);
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

void addGroupAdmin({required UserModel groupAdmin}) {
    _groupAdmins.add(groupAdmin);
    _groupModel.adminsUID.add(groupAdmin.uid);
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

```

Функція setEditGroupSettings встановлює налаштування для редагування налаштувань групи. Якщо параметр value дорівнює true,

редагування налаштувань дозволено. Якщо value дорівнює false, редагування налаштувань заборонено.

Функція setApproveNewMembers встановлює налаштування, яке дозволяє адміністраторам групи затверджувати нових учасників. Якщо параметр value дорівнює true, затвердження нових учасників дозволено. Якщо value дорівнює false, затвердження нових учасників заборонено.

Функція setRequestToJoinGroup встановлює налаштування, яке дозволяє користувачам надсилати запит на вступ до групи. Якщо параметр value дорівнює true, це означає, що користувачам дозволено надсилати запит на вступ до групи. Якщо value дорівнює false, користувачам заборонено надсилати запити на вступ до групи.

Функція setLockMessages встановлює налаштування, яке блокує можливість надсилання повідомлень у групу. Якщо параметр value дорівнює true, надсилання повідомлень у групу заблоковане. Якщо value дорівнює false, надсилання повідомлень у групу дозволене.

Функції налаштування групи:

```
void setEditGroupSettings({required bool value}) {
    _groupModel.editSettings = value;
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

void setApproveNewMembers({required bool value}) {
    _groupModel.approveMembers = value;
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

void setRequestToJoinGroup({required bool value}) {
    _groupModel.requestToJoin = value;
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

void setLockMessages({required bool value}) {
    _groupModel.lockMessages = value;
```

```

notifyListeners();
if (_groupModel.groupId.isEmpty) return;
updateSettings();
}

```

Додаток також має додаткові функції, такі як перевірка онлайн статусу користувача, яка визначає, чи знаходиться користувач у додатку. Якщо користувач у додатку, то він вважається онлайн, і цю інформацію можуть бачити інші користувачі. Також є функція перевірки, чи було прочитано повідомлення отримувачем. Якщо так, то цю інформацію показують в інтерфейсі відправнику. Присутня функція виходу з облікового запису. З всіма функціями можна ознайомитись в додатку В.

Підсумовуючи, було створено діаграму компонентів мобільного месенджера (рис 2.13) і визначено основні компоненти месенджера.

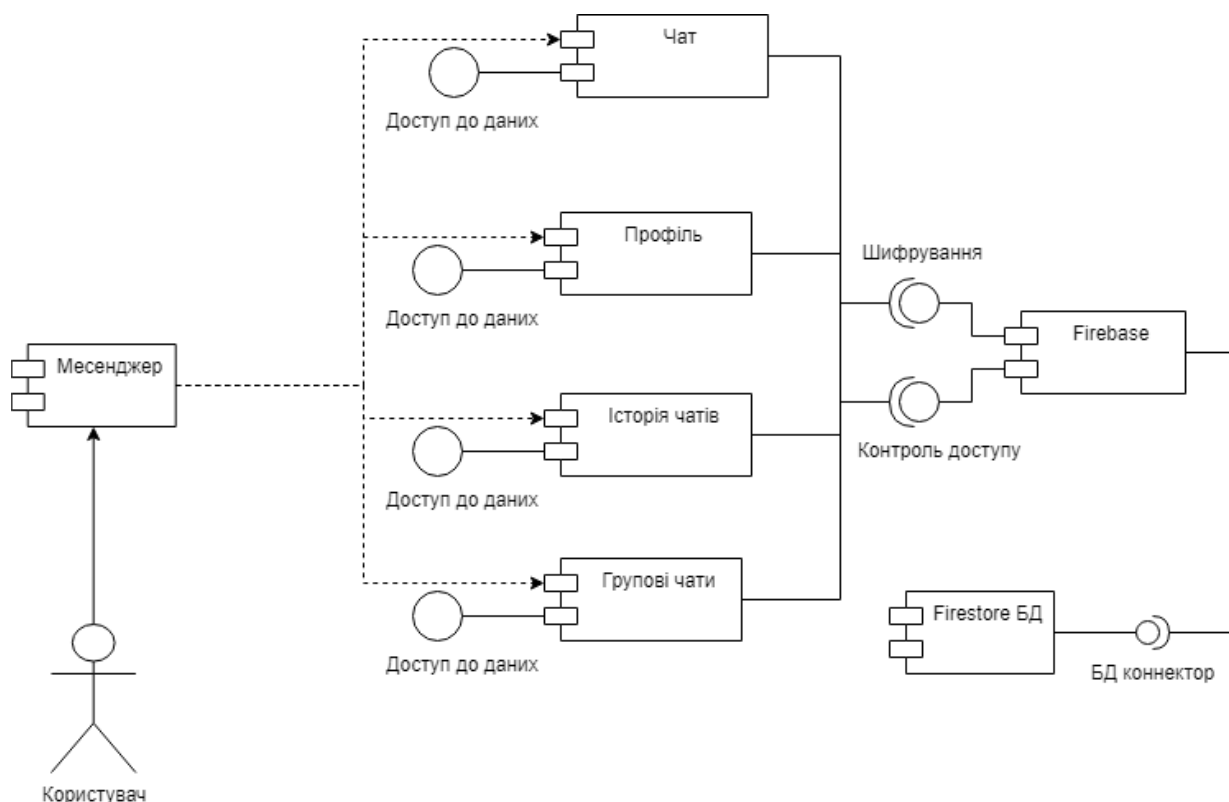


Рисунок 2.4 – Діаграма компонентів

На діаграмі визначені такі елементи:

- користувач – людина яка використовує додаток;
- месенджер – мобільний додаток, за допомогою якого користувачі взаємодіють із функціями.

- доступ до даних – механізм за допомогою якого месенджер взаємодіє з сервером для доступу до даних та їх керування;
- firebase – хмарна платформа, що пропонує сервіси для мобільних додатків, через неї йде підключення до БД і шифрування даних;
- firestore – тут зберігаються дані про користувача, групи і чати;

Висновки до другого розділу

У другому розділі було визначено вимоги до програмного продукту, проаналізовано функціональні та нефункціональні вимоги, а також встановлено часові характеристики.

Були створені та описані діаграми використання та класів з урахуванням усіх використаних патернів проектування.

Розроблено базу даних, включаючи опис колекцій, що вона включає.

Проведено проектування алгоритмів роботи за допомогою діаграм активності, створено функціонал мобільного додатку та презентовано реалізовані функції. На основі розроблених функцій була побудована та продемонстрована діаграма компонентів.

					КН.КР.Б-122-24-ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З МОБІЛЬНИМ ДОДАТКОМ

3.1. Порядок встановлення та налаштування параметрів платформи

Для встановлення та налаштування додатку необхідно розгорнути всі компоненти і налаштувати кожен елемент окремо, а також систему в цілому.

Для опису архітектури додатку була створена діаграма розгортання (рис. 3.1).

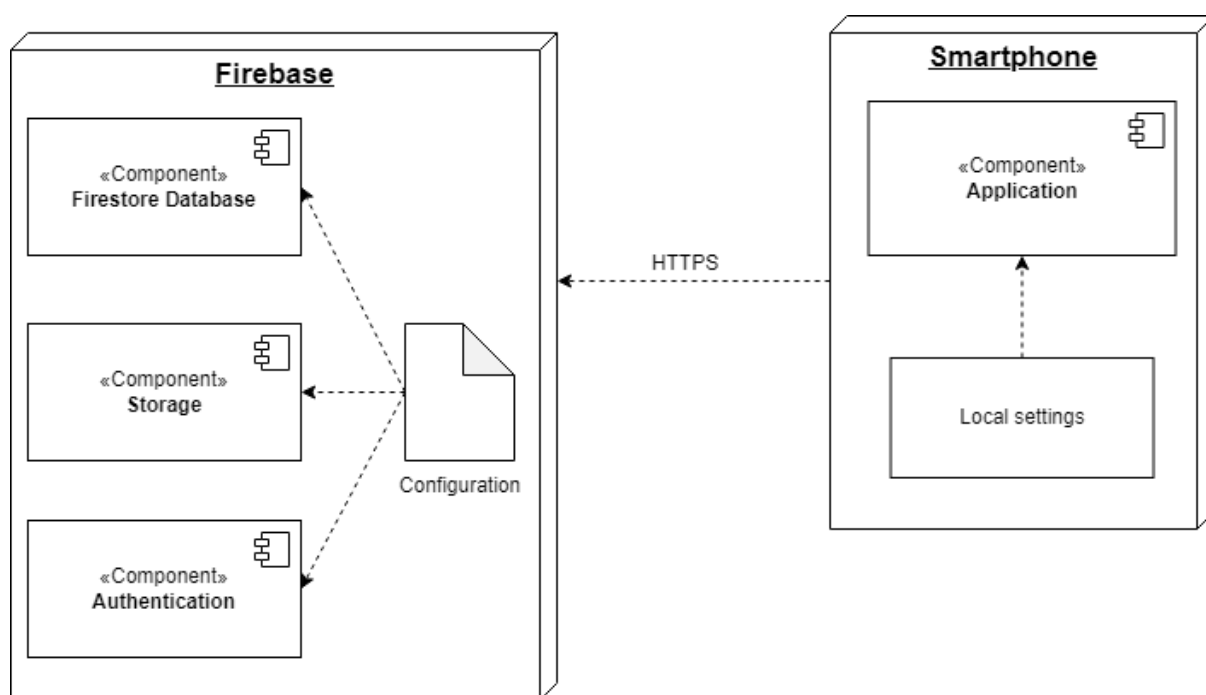


Рисунок 3.1 – Діаграма розгортання мобільного додатку

Опис процесу розгортання:

1. Smartphone – мобільний пристрій на якому запускається додаток, являється клієнтською частиною програми:
 - application – мобільний додаток, який взаємодіє з Firebase через протокол для безпечного зв'язку HTTPS;
 - local settings – налаштування програми, які зберігаються в смартфоні;

2. Firebase – сервіс для розробки мобільних додатків, використовується як серверна частина:

- firestore database – використовується для зберігання та отримання даних;
- storage – хмарне сховище для зберігання файлів, вона дозволяє зберігати та розповсюджувати файли користувачі [21];
- authentication – дозволяє реалізувати механізми аутентифікації користувачів, підтримує різні методи аутентифікації;
- configuration – список правил, використовуються для визначення того, хто може звертатись до яких даних та яким чином [22];

3.2. Структура інтерфейсу мобільного додатку

В цьому розділі ми розглянемо UI та робочий процес нашого мобільного додатка. Переглянемо різні скріншоти і надамо короткі описи до кожного з них.

Flutter надає SDK який містить усе необхідне для розробки UI, включаючи набір віджетів і анімацій. Віджети побудовані з використанням Material Design та Cupertino [23].

Material Design - це система дизайну, розроблена та підтримується дизайнерами та розробниками Google. На веб-сайті Material.io надаються докладні вказівки з UX та реалізації компонентів користувацького інтерфейсу для Android, Flutter та інших платформ [24].

Cupertino - це дизайн-довідник, розроблений компанією Apple, який використовується для створення інтерфейсів користувача в додатках для платформи iOS. Цей дизайн відображає стандартні елементи та стиль, характерні для продуктів Apple [25].

При першому відкритті додатка на мобільному пристрої (як Android, так і iOS), користувач потрапляє на екран авторизації (рис.3.2).

					КН.КР.Б-122-24-ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

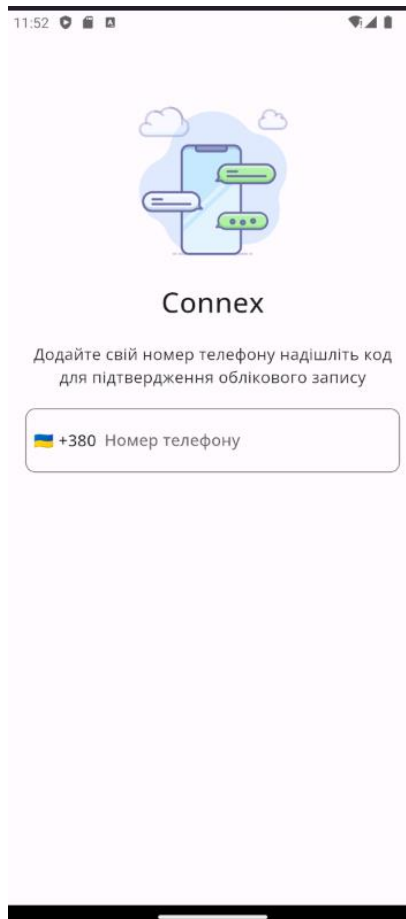


Рисунок 3.2 – Екран авторизації

Користувач може вибрати код своєї країни, натиснувши на відповідний код країни в полі введення. Внизу екрана з'явиться меню з кодами країн і їх назвами (рис.3.3).

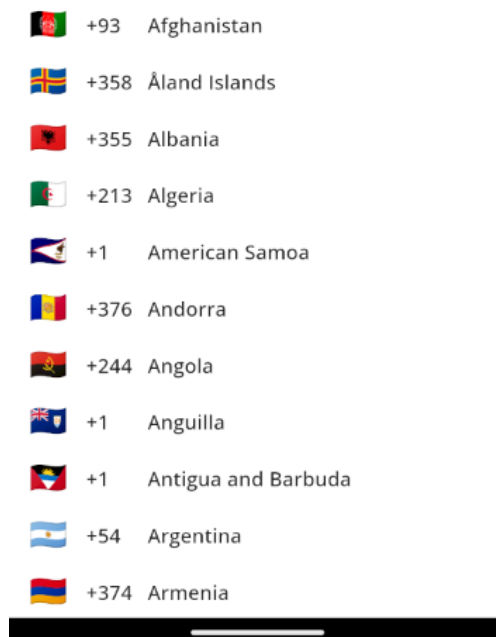


Рисунок 3.3 – Меню з кодами країни

					КН.КР.Б-122-24-ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

Після введення номера в поле введення з'явиться підтверджувальна галочка, показуючи, що номер введено правильно. Після цього користувач натискає на галочку, щоб підтвердити введений номер (рис.3.4).

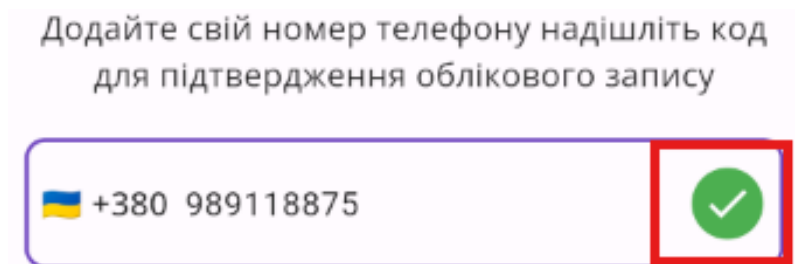


Рисунок 3.4 – Підтвердження номеру телефону

Після цього користувач переходить на екран верифікації телефону, де йому потрібно ввести 6-значний код, який приходить на мобільний телефон (рис.3.5).

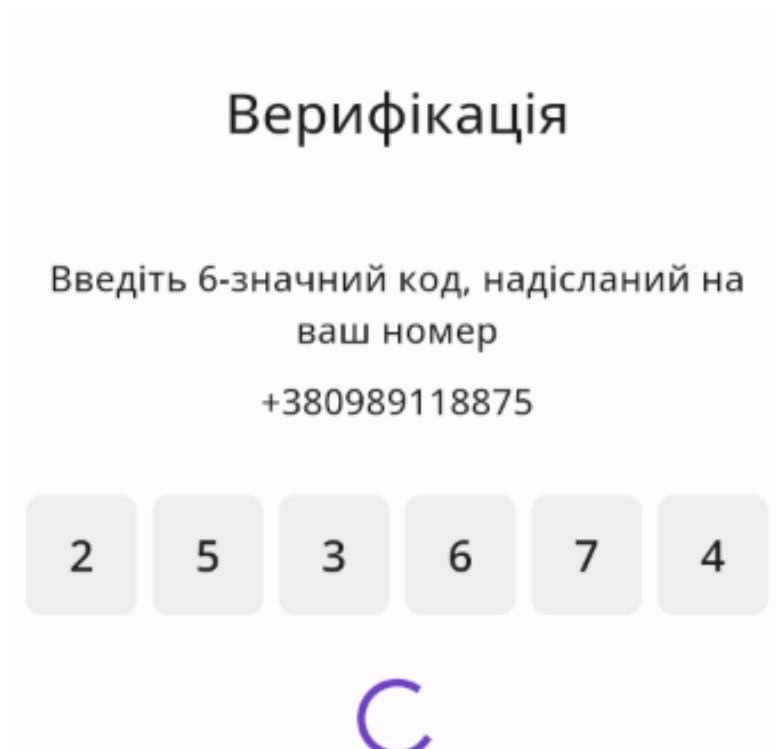


Рисунок 3.5 – Верифікація

Потім користувач переходить до надання своєї інформації яка буде відображатись в профілю, включаючи введення свого імені та додавання фото. Для додавання фотографії користувач натискає на кнопку поруч з полем для зображення і обирає фото з камери або галереї. Красним виділено кнопку для додавання фото (рис.3.6).

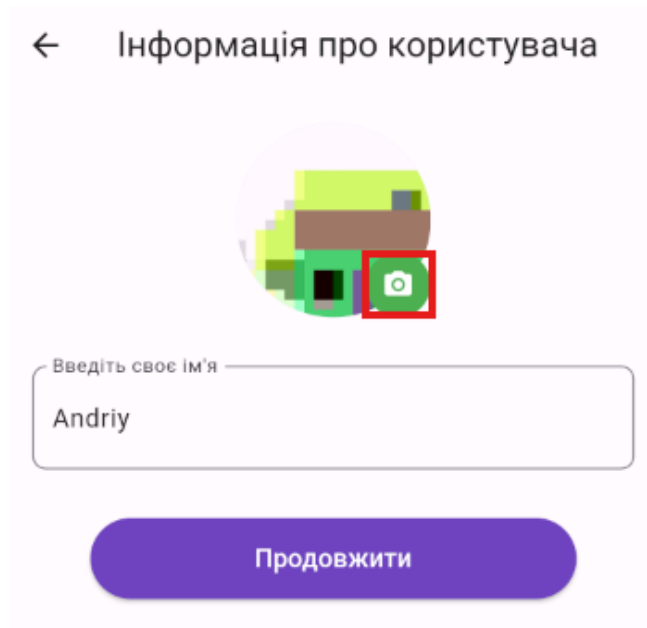


Рисунок 3.6 – Інформація про користувача

Після авторизації користувач буде переправлений на головний екран додатка (рис. 3.7).

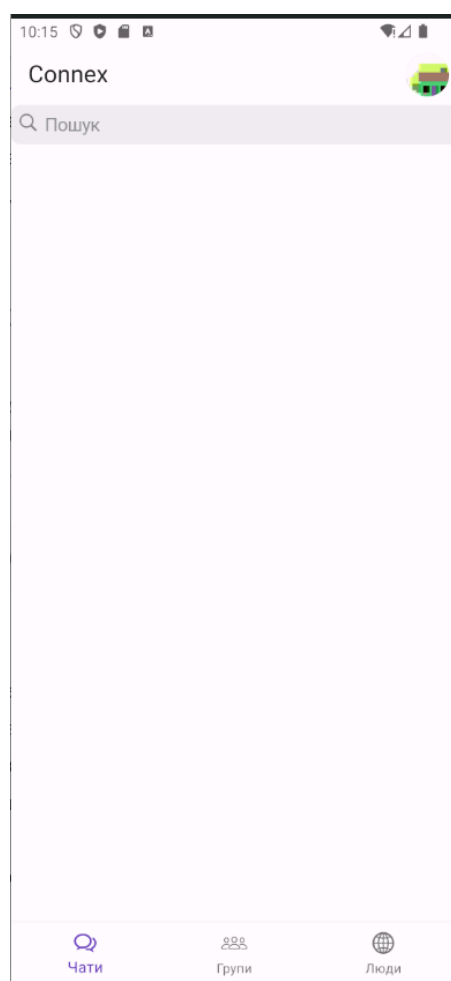


Рисунок 3.7 – Екран “Чати”

					КН.КР.Б-122-24-ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

Новий користувач не матиме чатів, тому для початку спілкування йому потрібно додати іншого користувача в друзі. Для цього необхідно перейти на екран "Люди", де він зможе знайти інших користувачів додатку (рис 3.8).

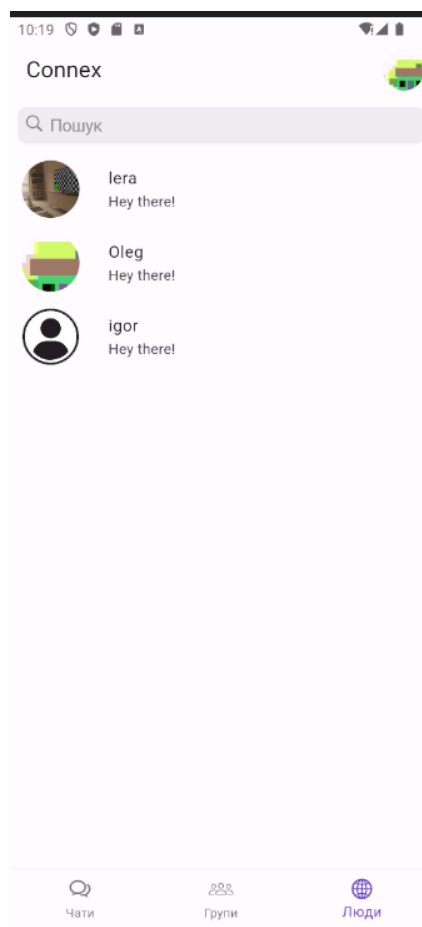


Рисунок 3.8 – Екран “Люди”

Потім потрібно відвідати профіль будь-якого користувача, якого ви хочете додати, і натиснути кнопку "Відправити запит" у його профілі. Ці кроки надішлють іншому користувачеві запит на дружбу (рис 3.9).

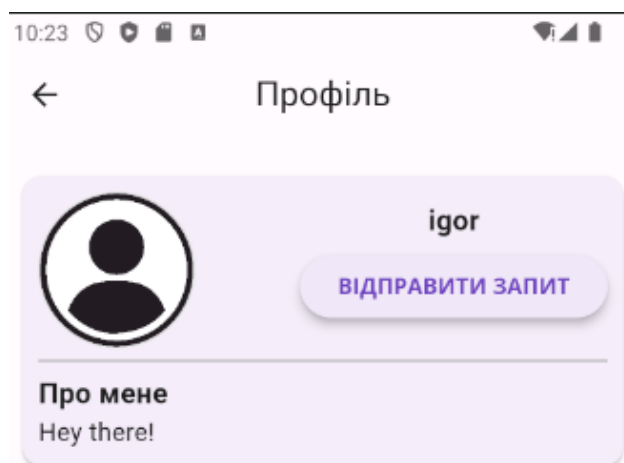


Рисунок 3.9 – Профіль користувача

Користувачу якому надіслали запит на дружбу появиться кнопка “Запити” (рис.3.10).

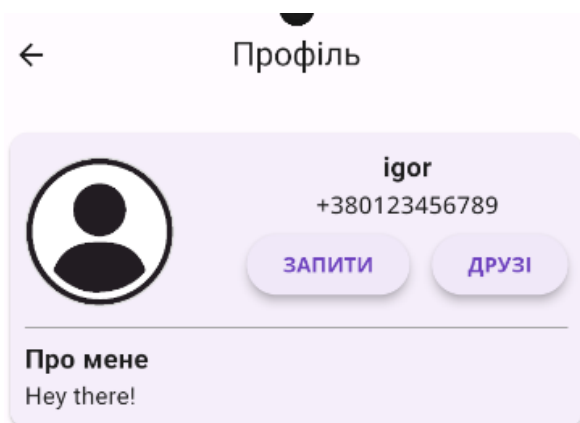


Рисунок 3.10 – Профіль із запитом на дружбу

Натиснувши кнопку "Запити", ви перейдете на екран, де можна побачити всі отримані запити. Для додавання користувача в друзі потрібно натиснути кнопку "Додати" (рис 3.11).

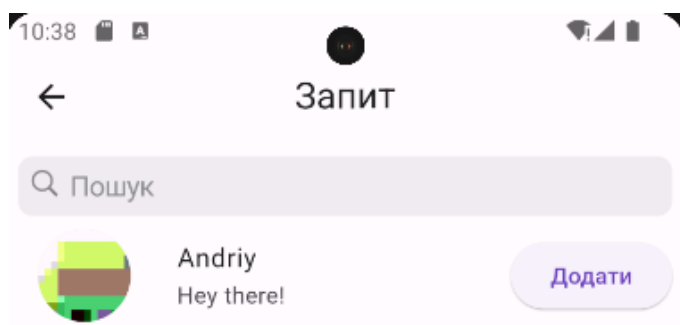


Рисунок 3.11 – Запити на дружбу

Коли у користувача з'явиться друг, він зможе перейти у свій профіль і натиснути кнопку "Друзі", щоб побачити список своїх друзів. Натиснувши на будь-якого з них, користувач перейде на екран чату з цим другом (рис.3.12).

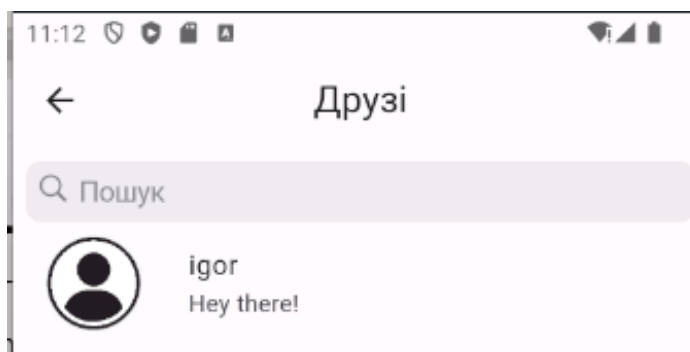


Рисунок 3.12 – Список друзів

					КН.КР.Б-122-24-ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

На екрані чату знаходиться поле введення повідомлення внизу екрана. Кнопка запису аудіо автоматично змінюється на кнопку відправки повідомлення в залежності від наявності тексту у полі введення. Також тут є кнопка для відправлення файлових повідомлень і кнопка меню емодзі. На екрані можна побачити фото та ім'я користувача, з яким ви спілкуєтеся, а також індикатор його онлайн-статусу (рис.3.13).

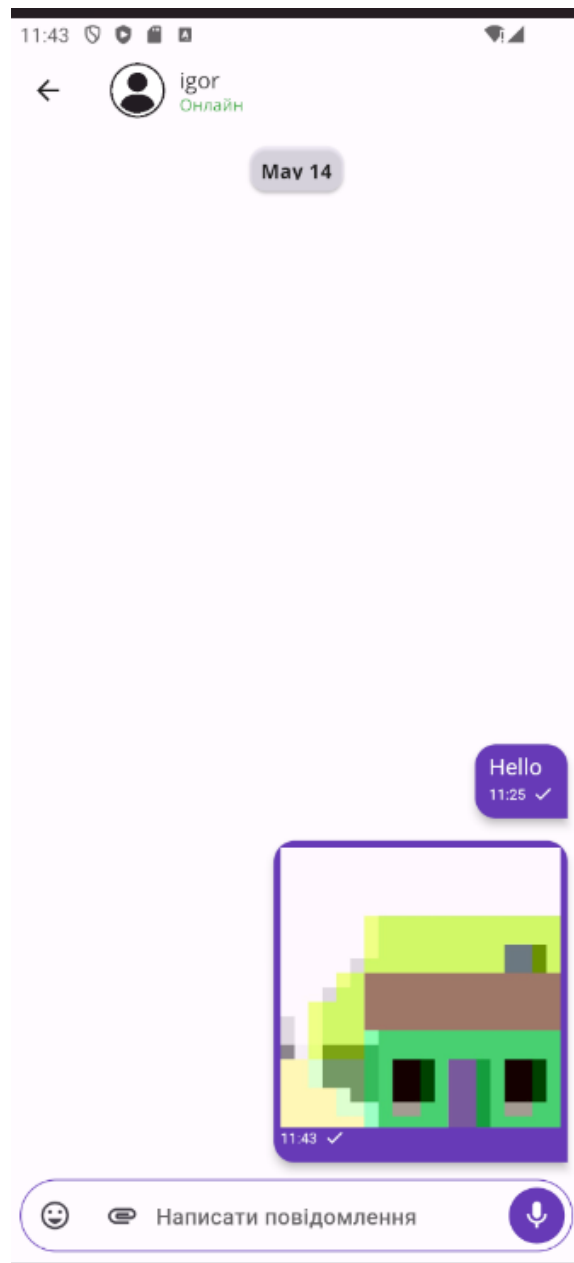


Рисунок 3.13 – Екран чату

Ми можемо взаємодіяти з повідомленнями, натискаючи на них. Після цього відкривається меню взаємодії, в якому доступні різні опції, такі як переслати, видалити та додати реакцію з переліку (рис.3.14).

					КН.КР.Б-122-24-ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

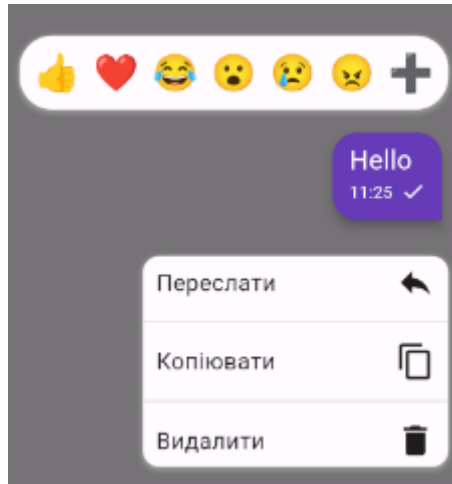


Рисунок 3.14 – Меню взаємодії

Переглянемо екран "Групи". Тут можна бачити як публічні, так і приватні групи. Для створення нової групи необхідно натиснути кнопку в правому нижньому куті (рис.3.15).



Рисунок 3.15 – Екран “Групи”

					КН.КР.Б-122-24-ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

Коли натискаємо кнопку для створення групи, ми переходимо на екран створення групи. Тут ми обираємо тип групи (публічна чи приватна), вводимо назву та опис групи, додаємо аватар і членів групи (рис.3.16).

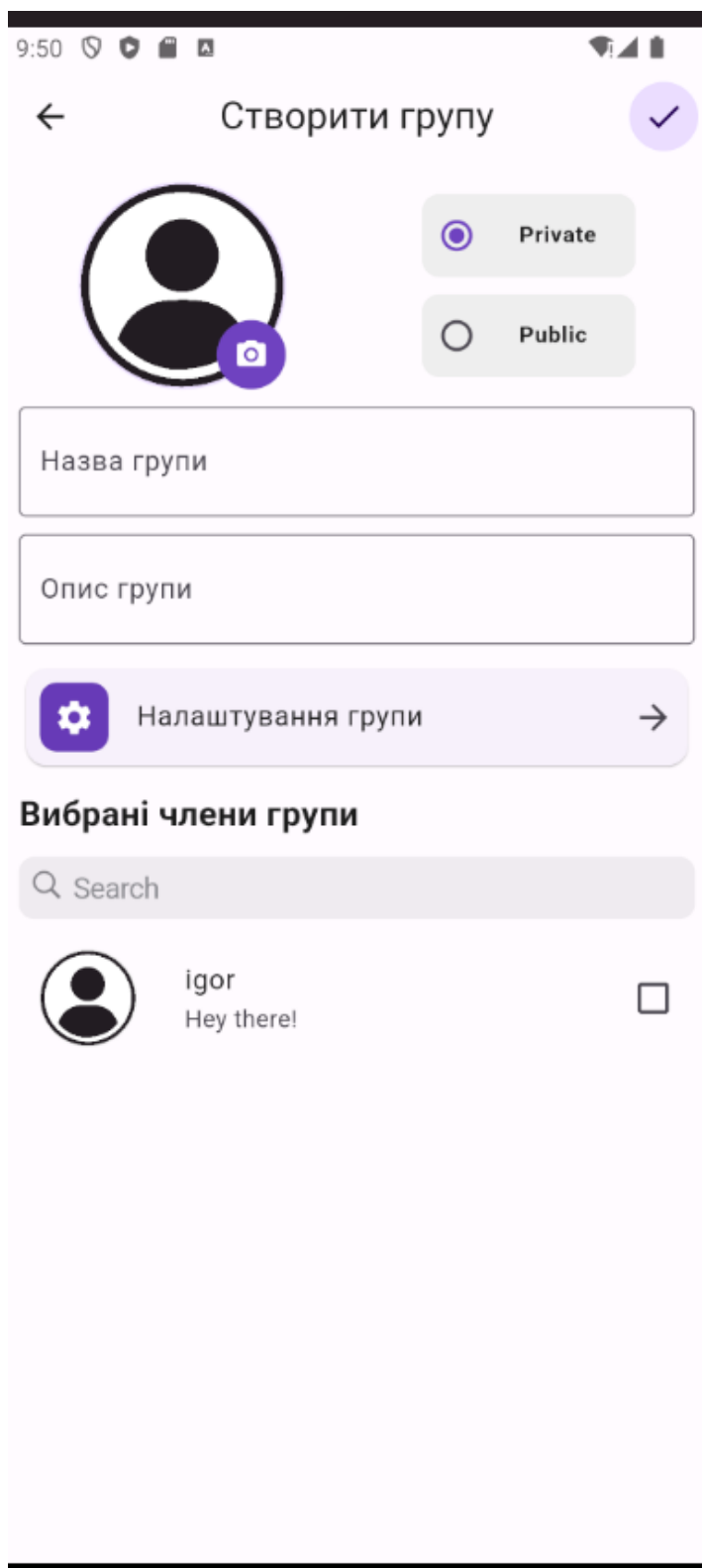


Рисунок 3.16 – Екран створення групи

					КН.КР.Б-122-24-ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

При натисканні кнопки "Налаштування групи" ми переходимо на екран налаштувань. Тут відображені всі опції, кожна з яких має свій опис, щоб користувач міг краще зрозуміти їх призначення (рис 3.17).

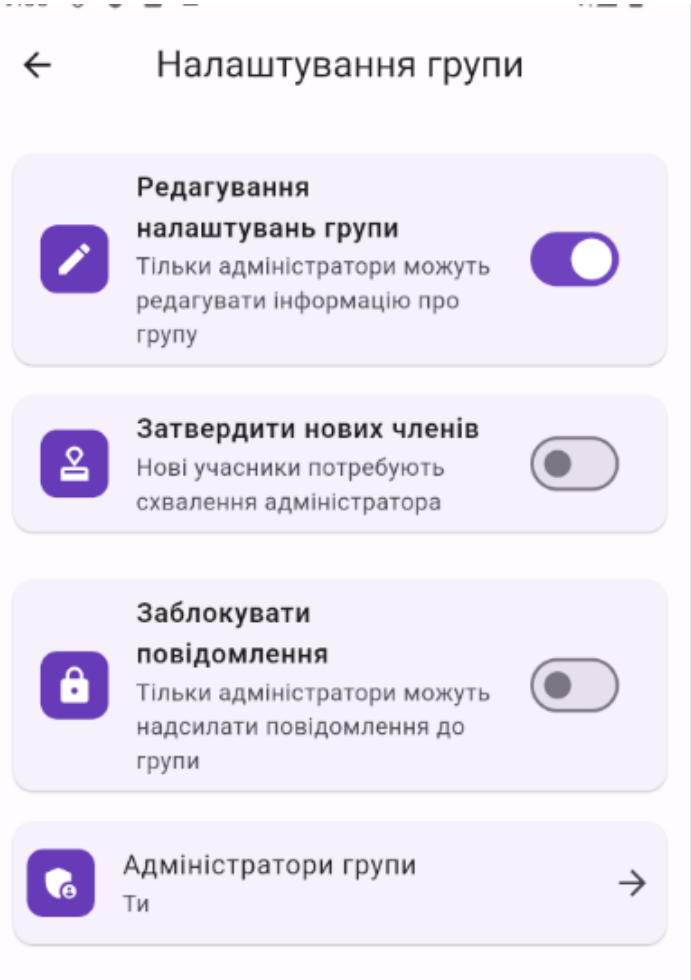


Рисунок 3.17 – Екран налаштувань

3.3. Тестування роботи мобільного додатку

Тестування роботи месенджера може складатися з кількох етапів, що перевіряють різні аспекти його функціональності та продуктивності. Важливими аспектами, які слід враховувати під час тестування, є:

- Функціональне тестування: відправка та отримання повідомлень, групові чати тощо;
- Тестування продуктивності: швидкість доставки повідомлень, ресурсоємність;

- Кроссплатформне тестування: підтримка різних операційних систем;
- Тестування інтерфейсу користувача: зручність навігації;

Проводячи тестування програмного забезпечення, ми можемо переконатися, що розроблюваний продукт відповідає необхідним функціональним вимогам і працює оптимально в різних умовах. Ці тести допомагають виявити та усунути потенційні проблеми або помилки, забезпечуючи стабільну роботу месенджера.

Висновки до третього розділу

У цьому розділі розглянуто схему розгортання платформи. Описано кожен структурний елемент системи та показано, як вони взаємодіють з іншими частинами платформи, що допомогло зрозуміти архітектуру системи та взаємозв'язки між її компонентами.

У другій частині розділу було присвячено увагу структурі інтерфейсу. Представлено огляд різних компонентів, що складають інтерфейс системи, що дозволяє користувачам комплексно зрозуміти, як вони взаємодіють із системою.

Нарешті, розглянуто важливий аспект тестування роботи додатка. Підкреслено важливість проведення тестів для забезпечення функціональності, надійності та продуктивності системи.

					КН.КР.Б-122-24-ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У рамках виконання кваліфікаційної роботи було створено мобільний додаток, який відповідає всім визначеним функціональним і нефункціональним вимогам. Користувачі можуть відправляти різні типи повідомлень: текстові, аудіо та відео. Додаток дозволяє створювати групові чати з можливістю їх налаштування, взаємодіяти з іншими користувачами, додавати їх у друзі та переглядати їхні профілі. Інтерфейс користувача був розроблений таким чином, щоб бути привабливим та інтуїтивно зрозумілим як для новачків, так і для досвідчених користувачів.

Для досягнення такого результату було виконано наступні кроки:

Спочатку було визначено сутність мобільного сервісу, наведено приклади, такі як соціальні мережі та стрімінгові платформи. Було обрано конкретний сервіс, а саме месенджер, і наведено аргументи на користь цього вибору. Досліджено аналогічні продукти, проаналізовано та порівняно різні месенджери: Telegram, Viber, WhatsApp, Messenger. На основі цих даних визначено основні вимоги до нашого додатку. Виходячи з цих вимог, обрано набір технологій та інструментів, зокрема фреймворк Flutter та платформу для розробки мобільних додатків Firebase.

Проаналізовано функціональні та нефункціональні вимоги, створено діаграми використання та класи. Описано використані патерни програмування: "Фасад" і "Ланцюжок обов'язків". Розроблено базу даних і функціонал мобільного додатку.

Було створено користувацький інтерфейс та розглянуто різні компоненти, що складають інтерфейс системи. Проведено тестування роботи додатку та визначено важливість проведення тестів

Програмний продукт, реалізований під час кваліфікаційного проекту, готовий до використання та може бути оновлений з додаванням нових функцій.

					КН.КР.Б-122-24-ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка додатків на фреймворку Flutter. *Avada Media*. 2024 URL: <https://avada-media.ua/ua/services/flutter/>
2. Technologies Dart. *Brander*. 2023 URL: <https://brander.ua/technologies/dart>
3. Гузенко С. Dart — мову створення застосунків на Flutter. *HighLoad* – URL: <https://highload.today/uk/blogs/za-shho-my-lyubymo-dart-golovnu-ta-yedynu-movu-stvorenniya-dodatkov-na-flutter/>
4. Що таке Firebase?. *Avada Media*. 2024. URL: <https://avada-media.ua/ua/services/firebase/>
5. Грицюк Ю. Аналіз вимог до програмного забезпечення. Вид-во Львів. політехніки, 2018.
6. Що таке атрибути якості?. *Myservername*. 2023. URL: <https://uk.myservername.com/what-are-quality-attributes>
7. Optimizing Software Development. *JetSoft*. 2023. URL: <https://jetsoftpro.com/blog/optimizing-software-development-for-mobile-cross-platforms/>
8. Семенов С. Тестування продуктивності додатків. *Hillelblog*. 2020. URL: <https://blog.ithillel.ua/articles/testuvannia-produktyvnosti-dodatkov.-chastyna-1>
9. Вокер А. Діаграма варіантів використання UML. *Guru*. 2024. URL: <https://www.guru99.com/uk/use-case-diagrams-example.html>
10. Моралес Д. Що таке діаграма варіантів використання UML. *MindOnMap*. 2023. URL: <https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/>
11. Моралес Д. Що таке діаграма класів UML. *MindOnMap*. 2023. URL: <https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/>
12. UML Class Diagram Tutorial. *Visual Parading*. 2024. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>

					КН.КР.Б-122-24-ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

13. Що таке Патерн?. *Refactoring and Design Patterns*. 2024. URL: <https://refactoring.guru/uk/design-patterns/what-is-pattern>
14. Фасад. *Refactoring and Design Patterns*. 2024 URL: <https://refactoring.guru/uk/design-patterns/facade>
15. Ланцюжок обов'язків. *Refactoring and Design Patterns*. 2024 URL: <https://refactoring.guru/uk/design-patterns/chain-of-responsibility>
16. Cloud Firestore. *Firebase*. 2024. URL: <https://firebase.google.com/docs/firestore>
17. Dart Constants. *Javatpoint*. 2024. URL: <https://www.javatpoint.com/dart-constants>
18. Privacy and Security in Firebase. *Firebase*. 2024. URL: <https://firebase.google.com/support/privacy>
19. Що таке HTTPS. *Cityhost*. 2024. URL: <https://cityhost.ua/uk/blog/chto-takoe-https.html>
20. Firebase Authentication. *Firebase*. 2024. URL: <https://firebase.google.com/docs/auth>
21. Cloud Storage for Firebase. *Firebase*. 2024. URL: <https://firebase.google.com/docs/storage>
22. Firebase Security Rules. *Firebase*. 2024. URL: <https://firebase.google.com/docs/rules>
23. Building user interfaces with Flutter. *Docs | Flutter*. 2024. URL: <https://docs.flutter.dev/ui>
24. Material Design. *Material Design*. 2024. URL: <https://m3.material.io/get-started>
25. Cupertino (iOS-style) widgets. *Docs | Flutter*. 2024. URL: <https://docs.flutter.dev/ui/widgets/cupertino>

ДОДАТКИ

					КН.КР.Б-122-24-ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Опис варіанта використання (прецедентів).

Опис прецеденту “Авторизація/Реєстрація”

Назва варіанту використання	Авторизація/Реєстрація
Основні актори	Користувач
Короткий опис	Користувачі можуть проходити авторизацію або реєстрацію через свій мобільний номер, підтверджуючи це кодом, отриманим через SMS.

Опис прецеденту “Переглянути профіль користувача”

Назва варіанту використання	Переглянути профіль користувача
Основні актори	Користувач
Короткий опис	Користувачі мають можливість переглядати профілі інших користувачів, а також надсилати та приймати запити на дружбу.

Опис прецеденту “Переглянути свій профіль”

Назва варіанту використання	Переглянути свій профіль
Основні актори	Користувач
Короткий опис	Користувачі можуть переглядати свій профіль, змінювати налаштування та інформацію про себе, а також переглядати список друзів.

Опис прецеденту “Відправити повідомлення”

Назва варіанту використання	Відправити повідомлення
Основні актори	Користувач
Короткий опис	Користувачі можуть надсилати текстові та файлові повідомлення, а також додавати до них реакції.

Опис прецеденту “Створити груповий чат”

Назва варіанту використання	Створити груповий чат
Основні актори	Користувач
Короткий опис	Користувачі можуть створити груповий чат, налаштувати його, додати назву і опис, запросити друзів зі списку контактів, а також вибрати тип групи.

ДОДАТОК Б

Лістинг файлу “authentication_provider.dart”

```
import 'dart:convert';
import 'dart:io';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gayday_chat/constants.dart';
import 'package:flutter_gayday_chat/models/user_model.dart';
import 'package:flutter_gayday_chat/utilities/global_methods.dart';
import 'package:shared_preferences/shared_preferences.dart';
class AuthenticationProvider extends ChangeNotifier {
  bool _isLoading = false;
  bool _isSuccessfull = false;
  int? _resendToken;
  String? _uid;
  String? _phoneNumber;
  UserModel? _userModel;
  bool get isLoading => _isLoading;
```

					КН.КР.Б-122-24-ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

```

bool get isSuccessfull => _isSuccessfull;
int? get resendToken => _resendToken;
String? get uid => _uid;
String? get phoneNumber => _phoneNumber;
UserModel? get userModel => _userModel;

final FirebaseAuth _auth = FirebaseAuth.instance;
final FirebaseFirestore _firestore = FirebaseFirestore.instance;
final FirebaseStorage _storage = FirebaseStorage.instance;

Future<bool> checkAuthenticationState() async {
  bool isSignedIn = false;
  await Future.delayed(const Duration(seconds: 2));

  if (_auth.currentUser != null) {
    _uid = _auth.currentUser!.uid;
    await getUserDataFromFireStore();

    await saveUserDataToSharedPreference();

    notifyListeners();

    isSignedIn = true;
  } else {
    isSignedIn = false;
  }

  return isSignedIn;
}

Future<bool> checkUserExists() async {
  DocumentSnapshot documentSnapshot =
    await _firestore.collection(Constants.users).doc(_uid).get();
  if (documentSnapshot.exists) {
    return true;
  } else {
    return false;
  }
}

Future<void> updateUserStatus({required bool value}) async {
  await _firestore
    .collection(Constants.users)
    .doc(_auth.currentUser!.uid)
    .update({Constants.isOnline: value});
}

Future<void> getUserDataFromFireStore() async {
  DocumentSnapshot documentSnapshot =
    await _firestore.collection(Constants.users).doc(_uid).get();
  _userModel =
    UserModel.fromMap(documentSnapshot.data() as Map<String, dynamic>);
  notifyListeners();
}

```

					КН.КР.Б-122-24-ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

```

}
Future<void> saveUserDataToSharedPreference() async {
  SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
  await sharedPreferences.setString(
    Constants.userModel, jsonEncode(_userModel!.toMap()));
}
Future<void> getUserDataFromSharedPreference() async {
  SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
  String userModelString =
    sharedPreferences.getString(Constants.userModel) ?? '';
  _userModel = UserModel.fromMap(jsonDecode(userModelString));
  _uid = _userModel!.uid;
  notifyListeners();
}
Future<void> signInWithPhoneNumber({
  required String phoneNumber,
  required BuildContext context,
}) async {
  _isLoading = true;
  notifyListeners();

  await _auth.verifyPhoneNumber(
    phoneNumber: phoneNumber,
    verificationCompleted: (PhoneAuthCredential credential) async {
      await _auth.signInWithCredential(credential).then((value) async {
        _uid = value.user!.uid;
        _phoneNumber = value.user!.phoneNumber;
        _isSuccessfull = true;
        _isLoading = false;
        notifyListeners();
      });
    },
    verificationFailed: (FirebaseAuthException e) {
      _isSuccessfull = false;
      _isLoading = false;
      notifyListeners();
      showSnackBar(context, e.toString());
    },
    codeSent: (String verificationId, int? resendToken) async {
      _isLoading = false;
      _resendToken = resendToken;
      notifyListeners();
      Navigator.of(context).pushNamed(
        Constants.otpScreen,
        arguments: {
          Constants.verificationId: verificationId,
          Constants.phoneNumber: phoneNumber,
        },
      );
    },
  );
}

```

					КН.КР.Б-122-24-ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        codeAutoRetrievalTimeout: (String verificationId) {},
        timeout: const Duration(seconds: 60),
        forceResendingToken: resendToken,
    );
}
Future<void> verifyOTPCode({
    required String verificationId,
    required String otpCode,
    required BuildContext context,
    required Function onSuccess,
}) async {
    _isLoading = true;
    notifyListeners();

    final credential = PhoneAuthProvider.credential(
        verificationId: verificationId,
        smsCode: otpCode,
    );

    await _auth.signInWithCredential(credential).then((value) async {
        _uid = value.user!.uid;
        _phoneNumber = value.user!.phoneNumber;
        _isSuccessfull = true;
        _isLoading = false;
        onSuccess();
        notifyListeners();
    }).catchError((e) {
        _isSuccessfull = false;
        _isLoading = false;
        notifyListeners();
        showSnackBar(context, e.toString());
    });
}

void saveUserDataToFirestore({
    required UserModel userModel,
    required File? fileImage,
    required Function onSuccess,
    required Function onFail,
}) async {
    _isLoading = true;
    notifyListeners();
    try {
        if (fileImage != null) {
            String imageUrl = await storeFileToStorage(
                file: fileImage,
                reference: '${Constants.userImages}/${userModel.uid}');

            userModel.image = imageUrl;
        }
        userModel.lastSeen = DateTime.now().microsecondsSinceEpoch.toString();
    }
}

```

					КН.КР.Б-122-24-ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        userModel.createdAt = DateTime.now().microsecondsSinceEpoch.toString();

        _userModel = userModel;
        _uid = userModel.uid;
        await _firestore
            .collection(Constants.users)
            .doc(userModel.uid)
            .set(userModel.toMap());

        _isLoading = false;
        onSuccess();
        notifyListeners();
    } on FirebaseException catch (e) {
        _isLoading = false;
        notifyListeners();
        onFail(e.toString());
    }
}

Stream<DocumentSnapshot> userStream({required String userID}) {
    return _firestore.collection(Constants.users).doc(userID).snapshots();
}

Stream<QuerySnapshot> getAllUsersStream({required String userID}) {
    return _firestore
        .collection(Constants.users)
        .where(Constants.uid, isNotEqualTo: userID)
        .snapshots();
}

Future<void> sendFriendRequest({
    required String friendID,
}) async {
    try {
        await _firestore.collection(Constants.users).doc(friendID).update({
            Constants.friendRequestsUIDs: FieldValue.arrayUnion([_uid])
        });
        await _firestore.collection(Constants.users).doc(_uid).update({
            Constants.sentFriendRequestsUIDs: FieldValue.arrayUnion([friendID])
        });
    } on FirebaseException catch (e) {
        print(e);
    }
}

Future<void> cancelFriendRequest({required String friendID}) async {
    try {
        await _firestore.collection(Constants.users).doc(friendID).update({
            Constants.friendRequestsUIDs: FieldValue.arrayRemove([_uid])
        });
        await _firestore.collection(Constants.users).doc(_uid).update({
            Constants.sentFriendRequestsUIDs: FieldValue.arrayRemove([friendID])
        });
    } on FirebaseException catch (e) {

```

```

        print(e);
    }
}
Future<void> acceptFriendRequest({required String friendID}) async {
    await _firestore.collection(Constants.users).doc(friendID).update({
        Constants.friendsUIDs: FieldValue.arrayUnion([_uid])
    });
    await _firestore.collection(Constants.users).doc(_uid).update({
        Constants.friendsUIDs: FieldValue.arrayUnion([friendID])
    });
    await _firestore.collection(Constants.users).doc(friendID).update({
        Constants.sentFriendRequestsUIDs: FieldValue.arrayRemove([_uid])
    });
    await _firestore.collection(Constants.users).doc(_uid).update({
        Constants.friendRequestsUIDs: FieldValue.arrayRemove([friendID])
    });
}
Future<void> removeFriend({required String friendID}) async {
    await _firestore.collection(Constants.users).doc(friendID).update({
        Constants.friendsUIDs: FieldValue.arrayRemove([_uid])
    });
    await _firestore.collection(Constants.users).doc(_uid).update({
        Constants.friendsUIDs: FieldValue.arrayRemove([friendID])
    });
}
Future<List<UserModel>> getFriendsList(
    String uid, List<String> groupMembersUID) async {
    List<UserModel> friendsList = [];
    DocumentSnapshot documentSnapshot =
        await _firestore.collection(Constants.users).doc(uid).get();
    List<dynamic> friendsUIDs = documentSnapshot[Constants.friendsUIDs];

    for (String friendID in friendsUIDs) {
        if (groupMembersUID.isNotEmpty && groupMembersUID.contains(friendID)) {
            continue;
        }
        DocumentSnapshot documentSnapshot =
            await _firestore.collection(Constants.users).doc(friendID).get();
        UserModel friend =
            UserModel.fromMap(documentSnapshot.data() as Map<String, dynamic>);
        friendsList.add(friend);
    }
    return friendsList;
}
Future<List<UserModel>> getFriendRequestsList(
    {required String uid, required String groupID}) async {
    List<UserModel> friendRequestsList = [];
    if (groupID.isNotEmpty) {
        DocumentSnapshot documentSnapshot =
            await _firestore.collection(Constants.groups).doc(groupID).get();
    }
}

```

```

List<dynamic> requestsUID =
    documentSnapshot.get(Constants.awaitingApprovalUID);
for (String friendRequestsID in requestsUID) {
    DocumentSnapshot documentSnapshot = await _firestore
        .collection(Constants.users)
        .doc(friendRequestsID)
        .get();
    UserModel friendRequests =
        UserModel.fromMap(documentSnapshot.data() as Map<String, dynamic>);
    friendRequestsList.add(friendRequests);
}
}
DocumentSnapshot documentSnapshot =
    await _firestore.collection(Constants.users).doc(uid).get();
List<dynamic> friendRequestsUIDs =
    documentSnapshot.get(Constants.friendRequestsUIDs);

for (String friendRequestsID in friendRequestsUIDs) {
    DocumentSnapshot documentSnapshot = await _firestore
        .collection(Constants.users)
        .doc(friendRequestsID)
        .get();
    UserModel friendRequests =
        UserModel.fromMap(documentSnapshot.data() as Map<String, dynamic>);
    friendRequestsList.add(friendRequests);
}
return friendRequestsList;
}
Future logout() async {
    await _auth.signOut();
    SharedPreferences sharedPreferences = await SharedPreferences.getInstance();
    await sharedPreferences.clear();
    notifyListeners();
}
}

```

Лістинг файлу “chat_provider.dart”

```

import 'dart:io';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gayday_chat/constants.dart';
import 'package:flutter_gayday_chat/enum/enum.dart';
import 'package:flutter_gayday_chat/models/last_message_model.dart';
import 'package:flutter_gayday_chat/models/message_model.dart';
import 'package:flutter_gayday_chat/models/message_reply_model.dart';
import 'package:flutter_gayday_chat/models/user_model.dart';
import 'package:flutter_gayday_chat/utilities/global_methods.dart';
import 'package:uuid/uuid.dart';

```

					КН.КР.Б-122-24-ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

```

class ChatProvider extends ChangeNotifier {
  bool _isLoading = false;
  MessageReplyModel? _messageReplyModel;

  String _searchQuery = '';
  String get searchQuery => _searchQuery;

  bool get isLoading => _isLoading;
  MessageReplyModel? get messageReplyModel => _messageReplyModel;

  void setSearchQuery(String value) {
    _searchQuery = value;
    notifyListeners();
  }

  void setLoading(bool value) {
    _isLoading = value;
    notifyListeners();
  }

  void setMessageReplyModel(MessageReplyModel? messageReply) {
    _messageReplyModel = messageReply;
    notifyListeners();
  }

  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  Future<void> sendTextMessage({
    required UserModel sender,
    required String contactUID,
    required String contactName,
    required String contactImage,
    required String message,
    required MessageEnum messageType,
    required String groupId,
    required Function onSuccess,
    required Function(String) onError,
  }) async {
    setLoading(false);
    try {
      var messageId = const Uuid().v4();
      String replyMessage = _messageReplyModel?.message ?? '';
      String replyTo = _messageReplyModel == null
        ? ''
        : _messageReplyModel!.isMe
        ? 'Ти'
        : _messageReplyModel!.senderName;
      MessageEnum replyMessageType =
        _messageReplyModel?.messageType ?? MessageEnum.text;
      final messageModel = MessageModel(
        message: message,
        senderUID: sender.uid,

```

					КН.КР.Б-122-24-ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        senderName: sender.name,
        senderImage: sender.image,
        contactUID: contactUID,
        messageType: messageType,
        timeSent: DateTime.now(),
        messageId: messageId,
        isSeen: false,
        replyMessage: replyMessage,
        replyTo: replyTo,
        replyMessageType: replyMessageType,
        reactions: [],
        isSeenBy: [sender.uid],
        deletedBy: [],
    );
    if (groupId.isNotEmpty) {
        await _firestore
            .collection(Constants.groups)
            .doc(groupId)
            .collection(Constants.messages)
            .doc(messageId)
            .set(messageModel.toMap());
        await _firestore.collection(Constants.groups).doc(groupId).update({
            Constants.lastMessage: message,
            Constants.timeSent: DateTime.now().millisecondsSinceEpoch,
            Constants.senderUID: sender.uid,
            Constants.messageType: messageType.name,
        });
        setMessageReplyModel(null);
    } else {
        await handleContactMessage(
            messageModel: messageModel,
            contactUID: contactUID,
            contactName: contactName,
            contactImage: contactImage,
            onSuccess: onSuccess,
            onError: onError,
        );
        setMessageReplyModel(null);
    }
} catch (e) {
    setLoading(false);
    onError(e.toString());
}
}

Future<void> sendFileMessage({
    required UserModel sender,
    required String contactUID,
    required String contactName,
    required String contactImage,
    required File file,

```

					КН.КР.Б-122-24-ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

required MessageEnum messageType,
required String groupId,
required Function onSuccess,
required Function(String) onError,
}) async {
  setLoading(true);
  try {
    var messageId = const Uuid().v4();
    String replyMessage = _messageReplyModel?.message ?? '';
    String replyTo = _messageReplyModel == null
      ? ''
      : _messageReplyModel!.isMe
        ? 'Ти'
        : _messageReplyModel!.senderName;
    MessageEnum replyMessageType =
      _messageReplyModel?.messageType ?? MessageEnum.text;
    final ref =
      '${Constants.chatFiles}/${messageType.name}/${sender.uid}/${contactUID}/${messageId}';
    String fileUrl = await storeFileToStorage(file: file, reference: ref);
    final messageModel = MessageModel(
      message: fileUrl,
      senderUID: sender.uid,
      senderName: sender.name,
      senderImage: sender.image,
      contactUID: contactUID,
      messageType: messageType,
      timeSent: DateTime.now(),
      messageId: messageId,
      isSeen: false,
      replyMessage: replyMessage,
      replyTo: replyTo,
      replyMessageType: replyMessageType,
      reactions: [],
      isSeenBy: [sender.uid],
      deletedBy: [],
    );
    if (groupId.isNotEmpty) {
      await _firestore
        .collection(Constants.groups)
        .doc(groupId)
        .collection(Constants.messages)
        .doc(messageId)
        .set(messageModel.toMap());
      await _firestore.collection(Constants.groups).doc(groupId).update({
        Constants.lastMessage: fileUrl,
        Constants.timeSent: DateTime.now().millisecondsSinceEpoch,
        Constants.senderUID: sender.uid,
        Constants.messageType: messageType.name,
      });
    }
  }
}

```

					КН.КР.Б-122-24-ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        setLoading(false);
        onSuccess();
        setMessageReplyModel(null);
    } else {
        await handleContactMessage(
            messageModel: messageModel,
            contactUID: contactUID,
            contactName: contactName,
            contactImage: contactImage,
            onSuccess: onSuccess,
            onError: onError,
        );
        setMessageReplyModel(null);
    }
} catch (e) {
    setLoading(false);
    onError(e.toString());
}
}

Future<void> handleContactMessage({
    required MessageModel messageModel,
    required String contactUID,
    required String contactName,
    required String contactImage,
    required Function onSuccess,
    required Function(String p1) onError,
}) async {
    try {
        final contactMessageModel = messageModel.copyWith(
            userId: messageModel.senderUID,
        );
        final senderLastMessage = LastMassegaModel(
            senderUID: messageModel.senderUID,
            contactUID: contactUID,
            contactName: contactName,
            contactImage: contactImage,
            message: messageModel.message,
            messageType: messageModel.messageType,
            timeSent: messageModel.timeSent,
            isSeen: false,
        );
        final contactLastMessage = senderLastMessage.copyWith(
            contactImage: messageModel.senderImage,
            contactUID: messageModel.senderUID,
            contactName: messageModel.senderName,
        );
        await _firestore
            .collection(Constants.users)
            .doc(messageModel.senderUID)

```

					КН.КР.Б-122-24-ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .collection(Constants.chats)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageModel.messageId)
        .set(messageModel.toMap());
    await _firestore
        .collection(Constants.users)
        .doc(contactUID)
        .collection(Constants.chats)
        .doc(messageModel.senderUID)
        .collection(Constants.messages)
        .doc(messageModel.messageId)
        .set(contactMessageModel.toMap());
    await _firestore
        .collection(Constants.users)
        .doc(messageModel.senderUID)
        .collection(Constants.chats)
        .doc(contactUID)
        .set(senderLastMessage.toMap());
    await _firestore
        .collection(Constants.users)
        .doc(contactUID)
        .collection(Constants.chats)
        .doc(messageModel.senderUID)
        .set(contactLastMessage.toMap());
    setLoading(false);
    onSuccess();
} on FirebaseException catch (e) {
    setLoading(false);
    onError(e.message ?? e.toString());
} catch (e) {
    setLoading(false);
    onError(e.toString());
}
}

Future<void> sendReactionMessage({
    required String senderUID,
    required String contactUID,
    required String messageId,
    required String reaction,
    required bool groupId,
}) async {
    setLoading(true);
    String reactionToAdd = '$senderUID=$reaction';
    try {
        if (groupId) {
            final messageData = await _firestore
                .collection(Constants.groups)
                .doc(contactUID)
                .collection(Constants.messages)

```

```

        .doc(messageId)
        .get();
final message = MessageModel.fromMap(messageData.data()!);
if (message.reactions.isEmpty) {
    await _firestore
        .collection(Constants.groups)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({
            Constants.reactions: FieldValue.arrayUnion([reactionToAdd])
        });
} else {
    final uids = message.reactions.map((e) => e.split('=')[0]).toList();
    if (uids.contains(senderUID)) {
        final index = uids.indexOf(senderUID);
        message.reactions[index] = reactionToAdd;
    } else {
        message.reactions.add(reactionToAdd);
    }
    await _firestore
        .collection(Constants.groups)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({Constants.reactions: message.reactions});
}
} else {
    final messageData = await _firestore
        .collection(Constants.users)
        .doc(senderUID)
        .collection(Constants.chats)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageId)
        .get();
final message = MessageModel.fromMap(messageData.data()!);
if (message.reactions.isEmpty) {
    await _firestore
        .collection(Constants.users)
        .doc(senderUID)
        .collection(Constants.chats)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({
            Constants.reactions: FieldValue.arrayUnion([reactionToAdd])
        });
    await _firestore
        .collection(Constants.users)

```

					КН.КР.Б-122-24-ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .doc(contactUID)
        .collection(Constants.chats)
        .doc(senderUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({
            Constants.reactions: FieldValue.arrayUnion([reactionToAdd])
        });
    } else {
        final uids = message.reactions.map((e) => e.split('=')[0]).toList();
        if (uids.contains(senderUID)) {
            final index = uids.indexOf(senderUID);
            message.reactions[index] = reactionToAdd;
        } else {
            message.reactions.add(reactionToAdd);
        }
    }
    await _firestore
        .collection(Constants.users)
        .doc(senderUID)
        .collection(Constants.chats)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({Constants.reactions: message.reactions});
    await _firestore
        .collection(Constants.users)
        .doc(contactUID)
        .collection(Constants.chats)
        .doc(senderUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({Constants.reactions: message.reactions});
    }
    }
    setLoading(false);
} catch (e) {
    print(e.toString());
}
}

Stream<List<LastMassegaModel>> getChatListStream(String userId) {
    return _firestore
        .collection(Constants.users)
        .doc(userId)
        .collection(Constants.chats)
        .orderBy(Constants.timeSent, descending: true)
        .snapshots()
        .map((snapshot) {
            return snapshot.docs.map((doc) {
                return LastMassegaModel.fromMap(doc.data());
            }).toList();
        });
}

```

					КН.КР.Б-122-24-ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    });
}
Stream<List<MessageModel>> getMessagesStream({
    required String userId,
    required String contactUID,
    required String isGroup,
}) {
    if (isGroup.isNotEmpty) {
        return _firestore
            .collection(Constants.groups)
            .doc(contactUID)
            .collection(Constants.messages)
            .snapshots()
            .map((snapshot) {
                return snapshot.docs.map((doc) {
                    return MessageModel.fromMap(doc.data());
                }).toList();
            });
    } else {
        return _firestore
            .collection(Constants.users)
            .doc(userId)
            .collection(Constants.chats)
            .doc(contactUID)
            .collection(Constants.messages)
            .snapshots()
            .map((snapshot) {
                return snapshot.docs.map((doc) {
                    return MessageModel.fromMap(doc.data());
                }).toList();
            });
    }
}
Stream<int> getUnreadMessagesStream({
    required String userId,
    required String contactUID,
    required bool isGroup,
}) {
    if (isGroup) {
        return _firestore
            .collection(Constants.groups)
            .doc(contactUID)
            .collection(Constants.messages)
            .snapshots()
            .asynMap((event) {
                int count = 0;
                for (var doc in event.docs) {
                    final message = MessageModel.fromMap(doc.data());
                    if (!message.isSeenBy.contains(userId)) {
                        count++;
                    }
                }
                return count;
            });
    }
}

```

```

        }
    }
    return count;
});
} else {
    return _firestore
        .collection(Constants.users)
        .doc(userId)
        .collection(Constants.chats)
        .doc(contactUID)
        .collection(Constants.messages)
        .where(Constants.isSeen, isEqualTo: false)
        .where(Constants.senderUID, isNotEqualTo: userId)
        .snapshots()
        .map((event) => event.docs.length);
}
}
Future<void> setMessageStatus({
    required String currentUserId,
    required String contactUID,
    required String messageId,
    required List<String> isSeenByList,
    required bool isGroup,
}) async {
    if (isGroup) {
        if (isSeenByList.contains(currentUserId)) {
            return;
        } else {
            await _firestore
                .collection(Constants.groups)
                .doc(contactUID)
                .collection(Constants.messages)
                .doc(messageId)
                .update({
                    Constants.isSeenBy: FieldValue.arrayUnion([currentUserId])
                });
        }
    } else {
        await _firestore
            .collection(Constants.users)
            .doc(currentUserId)
            .collection(Constants.chats)
            .doc(contactUID)
            .collection(Constants.messages)
            .doc(messageId)
            .update({Constants.isSeen: true});
        await _firestore
            .collection(Constants.users)
            .doc(contactUID)
            .collection(Constants.chats)

```

					КН.КР.Б-122-24-ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .doc(currentUserId)
        .collection(Constants.messages)
        .doc(messageId)
        .update({Constants.isSeen: true});
    await _firestore
        .collection(Constants.users)
        .doc(currentUserId)
        .collection(Constants.chats)
        .doc(contactUID)
        .update({Constants.isSeen: true});
    await _firestore
        .collection(Constants.users)
        .doc(contactUID)
        .collection(Constants.chats)
        .doc(currentUserId)
        .update({Constants.isSeen: true});
    }
}

Future<void> deleteMessage({
    required String currentUserId,
    required String contactUID,
    required String messageId,
    required bool isGroup,
    required String messageType,
    required bool deleteForAll,
}) async {
    setLoading(true);
    if (isGroup) {
        await _firestore
            .collection(Constants.groups)
            .doc(contactUID)
            .collection(Constants.messages)
            .doc(messageId)
            .update({
                Constants.deletedBy: FieldValue.arrayUnion([currentUserId])
            });
    }
    if (deleteForAll) {
        final groupData =
            await _firestore.collection(Constants.groups).doc(contactUID).get();
        final List<String> groupMembers =
            List<String>.from(groupData.data()![Constants.membersUID]);
        await _firestore
            .collection(Constants.groups)
            .doc(contactUID)
            .collection(Constants.messages)
            .doc(messageId)
            .update({Constants.deletedBy: FieldValue.arrayUnion(groupMembers)});
    }
    if (messageType != MessageEnum.text.name) {
        await deleteFileFromStorage(
            currentUserId: currentUserId,

```

					КН.КР.Б-122-24-ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        contactUID: contactUID,
        messageId: messageId,
        messageType: messageType,
    );
    }
}
setLoading(false);
} else {
    await _firestore
        .collection(Constants.users)
        .doc(currentUserId)
        .collection(Constants.chats)
        .doc(contactUID)
        .collection(Constants.messages)
        .doc(messageId)
        .update({
            Constants.deletedBy: FieldValue.arrayUnion([currentUserId])
        });
    if (deleteForAll) {
        await _firestore
            .collection(Constants.users)
            .doc(contactUID)
            .collection(Constants.chats)
            .doc(currentUserId)
            .collection(Constants.messages)
            .doc(messageId)
            .update({
                Constants.deletedBy: FieldValue.arrayUnion([contactUID])
            });
    }
    if (messageType != MessageEnum.text.name) {
        await deleteFileFromStorage(
            currentUserId: currentUserId,
            contactUID: contactUID,
            messageId: messageId,
            messageType: messageType,
        );
    }
    setLoading(false);
}
}

```

```

Future<void> deleteFileFromStorage({
    required String currentUserId,
    required String contactUID,
    required String messageId,
    required String messageType,
}) async {
    final firebaseStorage = FirebaseStorage.instance;
    final ref = firebaseStorage.ref(

```

					КН.КР.Б-122-24-ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        '${Constants.chatFiles}/$messageType/$currentUserId/$contactUID/$messageId');
    try {
        await ref.getDownloadURL();
    } catch (error) {
        if (error == 'storage/object-not-found') {
            print('Object does not exist, skipping deletion');
            return;
        } else {
            throw error;
        }
    }

    await ref.delete();
    print('Object deleted');
}

Stream<QuerySnapshot> getLastMessage(
    {required String userId, required String groupId}) {
    return groupId.isNotEmpty
        ? _firestore
            .collection(Constants.groups)
            .where(Constants.membersUID, arrayContains: userId)
            .snapshots()
        : _firestore
            .collection(Constants.users)
            .doc(userId)
            .collection(Constants.chats)
            .snapshots();
}
}

```

Лістинг файлу “group_provide.dart”

```

import 'dart:io';

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gayday_chat/constants.dart';
import 'package:flutter_gayday_chat/enum/enum.dart';
import 'package:flutter_gayday_chat/models/group_model.dart';
import 'package:flutter_gayday_chat/models/message_model.dart';
import 'package:flutter_gayday_chat/models/user_model.dart';
import 'package:flutter_gayday_chat/utilities/global_methods.dart';
import 'package:uuid/uuid.dart';

class GroupProvider extends ChangeNotifier {
    bool _isLoading = false;
    GroupModel _groupModel = GroupModel(
        creatorUID: '',
        groupName: '',

```

					КН.КР.Б-122-24-ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    groupDescription: '',
    groupImage: '',
    groupId: '',
    lastMessage: '',
    senderUID: '',
    messageType: MessageEnum.text,
    messageId: '',
    timeSent: DateTime.now(),
    createdAt: DateTime.now(),
    isPrivate: true,
    editSettings: true,
    approveMembers: false,
    requestToJoin: false,
    lockMessages: false,
    membersUID: [],
    adminsUID: [],
    awaitingApprovalUID: [],
);
final List<UserModel> _groupMembers = [];
final List<UserModel> _groupAdmins = [];

bool get isLoading => _isLoading;
GroupModel get groupModel => _groupModel;
List<UserModel> get groupMembers => _groupMembers;
List<UserModel> get groupAdmins => _groupAdmins;
final FirebaseFirestore _firestore = FirebaseFirestore.instance;

void setIsLoading({required bool value}) {
    _isLoading = value;
    notifyListeners();
}

void setEditGroupSettings({required bool value}) {
    _groupModel.editSettings = value;
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

void setApproveNewMembers({required bool value}) {
    _groupModel.approveMembers = value;
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

void setRequestToJoinGroup({required bool value}) {
    _groupModel.requestToJoin = value;
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;

```

					КН.КР.Б-122-24-ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        updateSettings();
    }

    void setLockMessages({required bool value}) {
        _groupModel.lockMessages = value;
        notifyListeners();
        if (_groupModel.groupId.isEmpty) return;
        updateSettings();
    }

    Future<void> updateSettings() async {
        try {
            await _firestore
                .collection(Constants.groups)
                .doc(_groupModel.groupId)
                .update(_groupModel.toMap());
        } catch (error) {
            print(error.toString());
        }
    }

    void addGroupMember({required UserModel groupMember}) {
        _groupMembers.add(groupMember);
        _groupModel.membersUID.add(groupMember.uid);
        notifyListeners();
        if (_groupModel.groupId.isEmpty) return;
        updateSettings();
    }

    void addGroupAdmin({required UserModel groupAdmin}) {
        _groupAdmins.add(groupAdmin);
        _groupModel.adminsUID.add(groupAdmin.uid);
        notifyListeners();
        if (_groupModel.groupId.isEmpty) return;
        updateSettings();
    }

    Future<void> setGroupModel({required GroupModel groupModel}) async {
        _groupModel = groupModel;
        notifyListeners();
    }

    Future<void> removeGroupMembers({required UserModel groupMember}) async {
        _groupMembers.remove(groupMember);
        _groupAdmins.remove(groupMember);
        _groupModel.membersUID.remove(groupMember.uid);
        notifyListeners();
        if (_groupModel.groupId.isEmpty) return;
        updateSettings();
    }
}

```

					КН.КР.Б-122-24-ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

void removeGroupAdmins({required UserModel groupAdmin}) {
    _groupAdmins.remove(groupAdmin);
    _groupModel.adminsUID.remove(groupAdmin.uid);
    notifyListeners();
    if (_groupModel.groupId.isEmpty) return;
    updateSettings();
}

Future<List<UserModel>> getGroupMembersData({
    required bool isAdmin,
}) async {
    try {
        List<UserModel> groupMembersData = [];
        List<String> groupMembersUID =
            isAdmin ? _groupModel.adminsUID : _groupModel.membersUID;
        for (var uid in groupMembersUID) {
            var user = await _firestore.collection(Constants.users).doc(uid).get();
            groupMembersData.add(UserModel.fromMap(user.data()!));
        }
        return groupMembersData;
    } catch (error) {
        return [];
    }
}

Future<void> updateMembers() async {
    _groupMembers.clear();
    _groupMembers.addAll(await getGroupMembersData(isAdmin: false));
    notifyListeners();
}

Future<void> updateAdmins() async {
    _groupAdmins.clear();
    _groupAdmins.addAll(await getGroupMembersData(isAdmin: true));
    notifyListeners();
}

Future<void> clearGroupMembers() async {
    _groupMembers.clear();
    _groupAdmins.clear();
    _groupModel = GroupModel(
        creatorUID: '',
        groupName: '',
        groupDescription: '',
        groupImage: '',
        groupId: '',
        lastMessage: '',
        senderUID: '',
        messageType: MessageEnum.text,
    );
}

```

```

        messageId: '',
        timeSent: DateTime.now(),
        createdAt: DateTime.now(),
        isPrivate: true,
        editSettings: true,
        approveMembers: false,
        requestToJoin: false,
        lockMessages: false,
        membersUID: [],
        adminsUID: [],
        awaitingApprovalUID: [],
    );
    notifyListeners();
}
List<String> getGroupMembersUid() {
    return _groupMembers.map((e) => e.uid).toList();
}
List<String> getGroupAdminsUid() {
    return _groupAdmins.map((e) => e.uid).toList();
}
Stream<DocumentSnapshot> groupStream({required String groupId}) {
    return _firestore.collection(Constants.groups).doc(groupId).snapshots();
}
streamGroupMember({required List<String> memberUid}) {
    return Stream.fromFuture(Future.wait<DocumentSnapshot>(
        memberUid.map<Future<DocumentSnapshot>>((uid) async {
            return await _firestore.collection(Constants.users).doc(uid).get();
        })));
}
Future<void> createGroup({
    required GroupModel newGroup,
    required File? fileImage,
    required Function onSuccess,
    required Function(String) onFall,
}) async {
    setIsLoading(value: true);
    try {
        var groupId = const Uuid().v4();
        newGroup.groupId = groupId;
        if (fileImage != null) {
            final String imageUrl = await storeFileToStorage(
                file: fileImage, reference: '${Constants.groupImages}/${groupId}');
            newGroup.groupImage = imageUrl;
        }
        newGroup.adminsUID = [newGroup.creatorUID, ...getGroupAdminsUid()];
        newGroup.membersUID = [newGroup.creatorUID, ...getGroupMembersUid()];
        setGroupModel(groupModel: newGroup);
        await _firestore
            .collection(Constants.groups)
            .doc(groupId)

```

					КН.КР.Б-122-24-ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        .set(groupModel.toMap());
        setIsLoading(value: false);
        onSuccess();
    } catch (e) {
        setIsLoading(value: false);
        onFall(e.toString());
    }
}

Stream<List<GroupModel>> getPrivateGroups({required String userId}) {
    return _firestore
        .collection(Constants.groups)
        .where(Constants.membersUID, arrayContains: userId)
        .where(Constants.isPrivate, isEqualTo: true)
        .snapshots()
        .asycMap((event) {
            List<GroupModel> groups = [];
            for (var group in event.docs) {
                groups.add(GroupModel.fromMap(group.data()));
            }

            return groups;
        });
}

Stream<List<GroupModel>> getPublicGroups({required String userId}) {
    return _firestore
        .collection(Constants.groups)
        .where(Constants.isPrivate, isEqualTo: false)
        .snapshots()
        .asycMap((event) {
            List<GroupModel> groups = [];
            for (var group in event.docs) {
                groups.add(GroupModel.fromMap(group.data()));
            }

            return groups;
        });
}

void changeType() {
    _groupModel.isPrivate = !_groupModel.isPrivate;
    notifyListeners();
    updateSettings();
}

Future<void> sendRequestToJoin({
    required String groupId,
    required String uid,
    required String groupName,
    required String groupImage,
}) async {
    await _firestore.collection(Constants.groups).doc(groupId).update({

```

```

        Constants.awaitingApprovalUID: FieldValue.arrayUnion([uid]),
    });
}

Future<void> acceptRequest({
    required String groupId,
    required String friendUID,
}) async {
    await _firestore.collection(Constants.groups).doc(groupId).update({
        Constants.membersUID: FieldValue.arrayUnion([friendUID]),
        Constants.awaitingApprovalUID: FieldValue.arrayRemove([friendUID]),
    });

    _groupModel.awaitingApprovalUID.remove(friendUID);
    _groupModel.membersUID.add(friendUID);
    notifyListeners();
}

bool isAdminSender({required MessageModel message, required String uid}) {
    if (message.senderUID == uid) {
        return true;
    } else if (_groupModel.adminsUID.contains(uid)) {
        return true;
    } else {
        return false;
    }
}

Future<void> exitGroup({
    required String uid,
}) async {
    bool isAdmin = _groupModel.adminsUID.contains(uid);

    await _firestore
        .collection(Constants.groups)
        .doc(_groupModel.groupId)
        .update({
            Constants.membersUID: FieldValue.arrayRemove([uid]),
            Constants.adminsUID:
                isAdmin ? FieldValue.arrayRemove([uid]) : _groupModel.adminsUID,
        });
    _groupMembers.removeWhere((element) => element.uid == uid);
    _groupModel.membersUID.remove(uid);
    if (isAdmin) {
        _groupAdmins.removeWhere((element) => element.uid == uid);
        _groupModel.adminsUID.remove(uid);
    }
    notifyListeners();
}

```