БАКАЛАВРСЬКА РОБОТА

БР. ІП - 43.00.00.000 ПЗ

Група ІП-21-3

Генсіцька Христина

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Генсіцька Христина Русланівна

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Реалізація проекту соціального програмного забезпечення

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього рівня Генсіцька Х.Р.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Крихівський Михайло Васильович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Генсіцькій Христині Русланівні

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) **“ Реалізація проекту соціального програмного забезпечення ”**

керівник проекту (роботи) Крихівський М.В., доцент

затверджені наказом закладу вищої освіти від **“** 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 09 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз та дослідження предметної області побудови соціального програмного забезпечення
2. Архітектура та функціональні особливості програмного забезпечення для соціальних мереж
3. Проектування карти потоку інтерфейсу користувача
4. Розробка діаграм варіантів використання
5. Програмна реалізація проекту соціального програмного забезпечення обміну контентом

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Огляд процесу обміну конвентом в пропонованій системі (рис. 1.1)
2. Вигляд платформи MoodChat (рис. 1.2)
3. Огляд етапів роботи DISSEMINATE (рис. 1.3)
4. Скріншоти додатку DISSEMINATE (рис. 1.4)
5. Схема взаємодії клієнта додатку з хмарними сервісами (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз та дослідження предметної області побудови соціального програмного забезпечення	04.05.2025	виконано
2	Архітектура та функціональні особливості програмного забезпечення для соціальних мереж	15.05.2025	виконано
3	Проектування карти потоку інтерфейсу користувача	21.05.2025	виконано
4	Розробка діаграм варіантів використання	28.05.2025	виконано
5	Програмна реалізація проекту соціального програмного забезпечення обміну контентом	03.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	09.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 77 сторінок, 30 рисунків, список використаних джерел із 30 найменуваннями, 1 додаток.

Метою роботи є розробка і реалізація функціонального прототипу соціального програмного забезпечення для обміну контентом, що базується на сучасних архітектурних підходах, забезпечує масштабованість, безпечність і зручність у користуванні

Об'єктом дослідження є процеси розробки та функціонування соціального програмного забезпечення для взаємодії користувачів.

Предметом дослідження є методи, інструменти та архітектурні підходи до реалізації соціального програмного забезпечення для обміну контентом.

В першому розділі у ході аналізу предметної області було обґрунтовано потребу у створенні нового програмного рішення та вивчено сучасні архітектурні підходи до соціальних платформ

В другому розділі було розроблено архітектуру системи, визначено її функціональні вимоги, змодельовано взаємодію користувачів і спроєктовано інтерфейс.

В третьому розділі реалізовано прототип соціального програмного забезпечення, виконано тестування функцій та розроблено документацію до API й вимог системи.

Висновок: розроблений прототип може бути використаний як основа для створення повноцінної соціальної платформи з розширеними функціями. Запропоновані архітектурні та інтерфейсні рішення можуть бути адаптовані для застосування в інших системах соціального призначення

КЛЮЧОВІ СЛОВА: СОЦІАЛЬНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МІКРОСЕРВІСНА АРХІТЕКТУРА, ОБМІН КОНТЕНТОМ, ІНТЕРФЕЙС КОРИСТУВАЧА, ТЕСТУВАННЯ, ХМАРНІ СЕРВІСИ, UX/UI, API, UML.

ANNOTATION

This bachelor's thesis comprises 77 pages, 30 figures, a list of 30 references, and 1 appendix.

The aim of the work is to develop and implement a functional prototype of social software for content sharing, based on modern architectural approaches, ensuring scalability, security, and user-friendliness.

The object of the research is the processes of development and functioning of social software for user interaction.

The subject of the research is the methods, tools, and architectural approaches for implementing social software for content sharing.

In the first chapter, an analysis of the subject area justified the need for creating a new software solution and explored modern architectural approaches to social platforms.

In the second chapter, the system's architecture was developed, its functional requirements were defined, user interaction was modeled, and the interface was designed.

In the third chapter, a prototype of the social software was implemented, functional testing was performed, and documentation for the API and system requirements was developed.

Conclusion: The developed prototype can be used as a basis for creating a full-fledged social platform with extended functionalities. The proposed architectural and interface solutions can be adapted for application in other social systems.

KEYWORDS: SOCIAL SOFTWARE, MICROSERVICES ARCHITECTURE, CONTENT SHARING, USER INTERFACE, TESTING, CLOUD SERVICES, UX/UI, API, UML.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AWS – Amazon Web Services – Веб-сервіси Amazon

S3 – Simple Storage Service – Простий сервіс зберігання

SDK – Software Development Kit – Комплект для розробки програмного забезпечення

FAB – Floating Action Button – Плаваюча кнопка дії

GPS – Global Positioning System – Глобальна система позиціонування

iOS – iPhone Operating System – Операційна система iPhone

UML – Unified Modeling Language – Уніфікована мова моделювання

QoE – Quality of Experience – Якість досвіду

ВСТУП

Сучасний цифровий простір характеризується стрімким розвитком соціальних платформ, що забезпечують обмін інформацією, взаємодію між користувачами, а також формування спільнот за інтересами. Зростання попиту на швидкий, безпечний і функціональний обмін контентом, а також вимоги до децентралізації даних і конфіденційності користувачів зумовлюють потребу у створенні нових рішень у сфері соціального програмного забезпечення.

Дана дипломна робота присвячена реалізації проєкту соціального програмного забезпечення, яке поєднує традиційні можливості соціальних мереж із сучасними підходами до архітектури, моделювання взаємодій і захисту даних. У межах роботи проведено аналіз предметної області, визначено функціональні та нефункціональні вимоги, спроєктовано архітектуру системи та реалізовано її програмну частину. Проєкт орієнтований на забезпечення ефективного обміну контентом між користувачами з урахуванням вимог масштабованості, зручності та безпеки.

Актуальність роботи

У контексті глобального поширення цифрових сервісів та активного користування соціальними мережами спостерігається зростання вимог до гнучкості, безпеки та персоналізації платформ соціальної взаємодії. Традиційні рішення часто мають централізований характер, що створює потенційні ризики втрати конфіденційності, нестабільності доступу до даних та обмеженої масштабованості. Водночас, зростає інтерес до архітектур на основі мікросервісів, хмарних обчислень та децентралізованих систем. Актуальність даної роботи полягає у реалізації проєкту, що відповідає сучасним викликам цифрової взаємодії, забезпечує зручний інтерфейс користувача, ефективний обмін контентом, а також високий рівень захисту даних та масштабованості.

Метою роботи є розробка і реалізація функціонального прототипу соціального програмного забезпечення для обміну контентом, що базується на сучасних архітектурних підходах, забезпечує масштабованість, безпечність і зручність у користуванні.

Завдання дослідження

1. Провести аналіз функціональних особливостей соціального програмного забезпечення.

2. Дослідити існуючі архітектури та парадигми взаємодії користувачів у соціальних платформах.

3. Розробити функціональну архітектуру системи на основі мікросервісного підходу.

4. Створити прототип користувацького інтерфейсу з урахуванням принципів UX/UI-дизайну.

5. Реалізувати основні програмні компоненти системи та протестувати їх.

6. Визначити специфікації API, вимоги до продуктивності та безпеки.

7. Провести тестування ключових функцій із залученням типових сценаріїв взаємодії.

Об’єктом дослідження є процеси розробки та функціонування соціального програмного забезпечення для взаємодії користувачів.

Предметом дослідження є методи, інструменти та архітектурні підходи до реалізації соціального програмного забезпечення для обміну контентом.

Методи дослідження

У роботі використано системний аналіз, методи об’єктно-орієнтованого моделювання (UML), прототипування, структурне проектування архітектури, методи UX-дизайну, а також методи функціонального тестування.

Наукова новизна роботи полягає у поєднанні мікросервісної архітектури, інтерфейсного моделювання та децентралізованого обміну

контентом для створення соціального програмного забезпечення, що орієнтоване на підвищення конфіденційності, масштабованості та гнучкості системи.

Практичне застосування

Розроблений прототип може бути використаний як основа для створення повноцінної соціальної платформи з розширеними функціями. Запропоновані архітектурні та інтерфейсні рішення можуть бути адаптовані для застосування в інших системах соціального призначення.

Бакалаврська робота містить 77 сторінок, 30 рисунків, 3 розділи список використаних джерел із 30 найменуванням, 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ПОБУДОВИ СОЦІАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Аналіз функціональних можливостей розроблюваного програмного забезпечення для обміну конвентом

Представлене розроблюване програмне забезпечення призначене для функціонування як глобальна платформа, що забезпечує безперебійну доставку цифрового контенту. Основний принцип його дії полягає у механізмі "скидання" контенту в географічно визначених локаціях для цільової аудиторії користувачів. Контент, що підлягає розповсюдженню, включає текстові повідомлення, короткоформатні відеофайли та статичні зображення. Після "скидання" користувачі отримують можливість розповсюджувати цей контент серед своїх соціальних контактів.

Функціональні можливості пропонованого ПЗ спроєктовані для забезпечення інтерактивного користувацького досвіду, що сприяє активній соціалізації та розширенню соціальних зв'язків між користувачами. Для забезпечення масштабованості, надійності та керованості цього інноваційного соціального додатку використовується хмарна інфраструктура. Такий підхід дозволяє ефективно розподіляти обчислювальні ресурси та забезпечувати безперебійну роботу системи в умовах зростаючого навантаження.

Дизайн інтерфейсу кінцевого користувача та архітектура внутрішньої системи РПЗ розроблені з урахуванням ключових вимог користувачів. До таких вимог відносяться:

1. Низька затримка.

Мінімальний час відгуку системи на дії користувача, що забезпечує плавне та безперебійне використання.

2. Чіткий графічний дизайн.

Інтуїтивно зрозумілий та естетично привабливий інтерфейс, що спрощує навігацію та взаємодію.

2. Надійний сервіс.

Стабільна та безперебійна робота системи, що мінімізує збої та втрату даних.

Завдяки своїм інноваційним функціональним можливостям та продуманій архітектурі, пропоноване ПЗ покликане запропонувати користувачам соціальних мереж якісно новий підхід до організації їхніх соціальних взаємодій, сприяючи більш значущій та ефективній комунікації з їхньою аудиторією.

1.2. Дослідження парадигми взаємодії у розроблюваних соціальних мережах

Сучасний ландшафт соціальних медіа характеризується безперервним розвитком і випуском нових програмних продуктів. Однак значна частка цих рішень демонструє тенденцію до віддалення від реальної фізичної взаємодії, що переважно зумовлено зростаючою спокусою технологічно опосередкованої соціалізації. Поширені платформи, такі як Facebook, Instagram або Twitter, пропонують користувачам надзвичайно зручний доступ до функцій обміну повідомленнями та контентом за допомогою єдиного натискання клавіші. Незважаючи на очевидні переваги у зручності, ці додатки істотно обмежують фізичну взаємодію між користувачами.

Пропоноване програмне забезпечення має на меті подолати розрив між віртуальним та фізичним світами. Це досягається шляхом реалізації механізму, що дозволяє користувачам відправляти "віртуальні посилки" (Drops) у реальному просторі для подальшого отримання іншими користувачами. Така інноваційна парадигма взаємодії сприяє більш значущим та глибинним зв'язкам між користувачами та їхніми контактами,

оскільки взаємодія не обмежується суто цифровим середовищем. Додаток забезпечує доставку повідомлень користувачам у конкретному контексті часу та місця, що підвищує їхню релевантність та цінність.

1.2.1. Значимість проекту

Розробка позиціонується як платформа, що надасть користувачам соціальних мереж можливість якіснішої та змістовнішої взаємодії з їхньою аудиторією. Ключовим аспектом є можливість отримувати повідомлення саме тоді і там, де вони є найбільш необхідними та доречними.

Незважаючи на те, що система класифікується як платформа соціальних мереж, вона не накладає жорстких обмежень на креативність користувачів при створенні "Дропів". "Дропи" можуть слугувати ефективним інструментом для підвищення значущості щоденних повідомлень у соціальних мережах як для самого користувача, так і для його аудиторії.

1.2.2. Мета дослідження та розробки

Розробка орієнтована на користувачів, які прагнуть до розширеної функціональності та глибшої взаємодії у своїх соціальних мережах. Програмний продукт дозволить користувачам "скидати" повідомлення у форматі тексту, зображень або короткоформатних відео для визначених одержувачів у конкретних географічних локаціях. Залежно від обраного місця, користувачі можуть створювати "Дропи", що забезпечують більш значущу контекстну взаємодію.

Наприклад, розглянемо сценарій, коли користувач заходить до продуктового магазину. Перевіряючи свій мобільний пристрій, він виявляє розблоковане повідомлення. Це повідомлення містить список продуктів, який був надісланий його другою половинкою раніше, але через напружений робочий день користувач його забув. Завдяки пропонованій системі, його партнер відправив "Дроп" зі списком покупок безпосередньо у магазин,

передбачаючи можливу забудькуватість. Тепер користувач може надіслати підтверджуюче повідомлення у формі коментаря, інформуючи про успішне виконання завдання. У цьому випадку повідомлення в соціальній мережі відіграло ключову роль у успішному здійсненні покупки.

Основною метою даного проекту є створення функціонального та масштабованого прототипу програмного забезпечення, що реалізує всі необхідні функції, визначені замовником. Цей документ деталізує функціональні можливості програми, її обмеження та вимоги, надані замовником.

1.2.3. Обсяг проекту

Користувацький інтерфейс (фронтенд) включатиме наступні стандартизовані сторінки: Головна, Друзі, Обліковий запис, Перегляд розташування Drop, Створення Drop, Вхід, Створення облікового запису, Сповіщення та Сплеш-сторінки. Всі сторінки будуть повністю функціональними, інтуїтивно зрозумілими для навігації та забезпечуватимуть єдиний естетичний вигляд.

Для обробки всієї серверної функціональності (бекенд), включаючи управління даними користувачів та клієнтськими підключеннями, пропоноване рішення використовуватиме технологію хмарних сервісів. Головним завданням цього проекту є розробка працюючого та масштабованого прототипу, що включає всі заявлені функції програми.

Оскільки дана розрбка є платформою соціальних мереж, її дизайн передбачатиме механізми для розширення мережі контактів користувачів. Система покращить соціальні взаємодії шляхом з'єднання людей через взаємні "Дропи" друзів. Додатково, це дозволить користувачам ділитися своїми ідеями та іменами через секцію коментарів до "Дропу", сприяючи активнішому обміну думками та взаємодії.

1.3. Архітектура та функціональні особливості програмного забезпечення для соціальних мереж

Розроблюване програмне забезпечення є автономним програмним додатком, призначеним для надання послуг соціальних мереж. Його функціональність полягає у можливості надсилання та отримання цифрових об'єктів, іменованих "Дропами" (Drops), які можуть містити текстові повідомлення, зображення або короткоформатні відеозаписи. Ці "Дропи" створюються користувачами та адресуються конкретним одержувачам з вказанням певного географічного місцеположення для їх отримання.

1.3.1. Функціональні можливості та взаємодія

Проектування додатку передбачає можливість надсилання "Дропів" декільком користувачам одночасно, що сприяє консолідації користувачів у визначеному місці отримання "Дропу". Після отримання "Дропу" користувачі мають можливість надати зворотний зв'язок шляхом залишення коментарів. Ці коментарі є доступними для перегляду всіма користувачами, які отримали відповідний "Дроп", що створює простір для колективної взаємодії.

Для забезпечення бажаної функціональності система реалізує архітектуру, що складається з одного основного компонента – клієнтського додатку, який взаємодіє з кількома хмарними сервісами. Клієнтський додаток відповідає за відображення всіх елементів користувацького інтерфейсу та забезпечення зв'язку з хмарними сервісами за допомогою їхніх SDK (Software Development Kits). Хмарні сервіси, у свою чергу, здійснюють управління даними, обробку клієнтських підключень та виконання користувацьких запитів за допомогою відповідних інструментів та сервісів розробки.

1.3.2. Хмарні сервіси та інтеграція

Для розробки застосовуватимуться чотири ключові хмарні сервіси:

- React Native IDE: Використовується для клієнтської розробки.

- MongoDB's Realm: Застосовується для управління базами даних NoSQL та серверною архітектурою.

- AWS (Amazon Web Services): Використовується для зберігання медіафайлів (зображень та відео) за допомогою сервісу S3 Simple Storage.

- Google Maps API: Інтегрований для забезпечення картографічних послуг.

Зв'язок між цими сервісами буде здійснюватися через вбудовані SDK у клієнтському додатку React Native, що забезпечить обробку різноманітних функціональних можливостей. Клієнтський додаток, у свою чергу, буде маніпулювати даними між сервісами для підтримки узгодженої системи між функціями SDK.

1.3.3. Управління пам'яттю та сумісність

Додаток буде використовувати фізичні ресурси пам'яті, надані хмарними сервісами MongoDB та AWS S3. Обсяг використаної пам'яті відповідатиме моделі зберігання хмарних сервісів, де доступний простір для системи буде масштабуватися відповідно до використаного обсягу.

Для завантаження програми на фізичні пристрої додаток вимагатиме близько 90 МБ вільного місця. Додаток буде сумісний з будь-якими смарт-пристроями, що працюють під управлінням операційної системи Android.

1.3.4. Принципи функціонування

Додаток реалізує бекенд-орієнтовану архітектуру, де вся базова інфраструктура обробляється хмарними сервісами. MongoDB's Realm забезпечує серверні обчислення за допомогою функцій, зберігання даних у MongoDB's Atlas (рішення NoSQL) та аутентифікацію користувачів. AWS S3 відповідає за зберігання зображень та відео, які можуть бути легко отримані за допомогою URL-посилань. Використання цих хмарних сервісів дозволяє

досягти економічної ефективності, еластичної масштабованості, швидкої ітерації, низьких адміністративних витрат та високої продуктивності розробки.

1.3.5. Аспекти безпеки

З огляду на потенційну можливість зустрічей користувачів у фізичних локаціях, питання безпеки вимагають ретельного розгляду для запобігання ризикованим ситуаціям. Хоча користувачі несуть відповідальність за додавання довірених друзів, додаток може інтегрувати механізм аутентифікації особи перед прийняттям будь-яких "Дропів" від користувачів, які не викликають повної довіри. Потенційним рішенням може бути аутентифікація за допомогою текстових повідомлень або електронної пошти. Після успішної взаємної аутентифікації користувачі зможуть безпечно обмінюватися "Дропами".

1.4. Дослідження децентралізованих підходів до розповсюдження контенту. Концепція та архітектура системи

Сучасні широко поширені мобільні додатки значною мірою залежать від централізованої хмарної інфраструктури для процесів зберігання, обчислень та управління в контексті розповсюдження мобільного контенту. Персональні хмарні сховища, такі як Dropbox та Google Drive, набули статусу майже товарної послуги. Однак усі ці популярні сервіси характеризуються централізованою природою, де весь контент розміщується у постачальника послуг, що вимагає від усіх користувачів постійного підключення до мережі Інтернет для доступу до сервісу. Сприйняті переваги цієї моделі комунікації включають централізоване управління, спрощення функціональності пристроїв та можливість підключення до будь-якого іншого пристрою незалежно від його фізичного місцезнаходження. Проте, така архітектура

робить сервіс недоступним для офлайн-користувачів та неефективним з точки зору використання мережевих ресурсів, оскільки сервіс вимагає комунікації через хмарний сервер навіть у випадках, коли кінцеві пристрої користувачів фізично розташовані поруч.

Незважаючи на значні наукові зусилля у галузі опортуністичної комунікації та ad-hoc мереж, централізовані сервіси зазвичай надаються переважно завдяки вигодам від збору даних користувачів, які переважають витрати, пов'язані зі зберіганням та розповсюдженням даних. У даній статті ми прагнемо продемонструвати можливість переміщення функціональності хмарних сервісів зберігання на периферію мобільної мережі, зокрема на смарт-мобільні пристрої. Це дозволить користувачам використовувати переваги спільного розташування при обміні контентом, створеним користувачами. Обмін контентом на основі близькості через короткодіючі бездротові мережі, такі як Wi-Fi та Bluetooth, не тільки дозволяє розвантажити комунікації з перевантажених стільникових мереж, але й, що найважливіше з точки зору користувача, зменшує витрати на пропускну здатність та затримку. Локальне розповсюдження контенту часто покращує якість користувацького досвіду, особливо коли обсяг контенту великий, а щільність користувачів висока.

Ми пропонуємо розподілений мобільний хмарний сервіс зберігання, який використовує географічне спільне розташування користувачів для локального обміну контентом, одночасно надаючи зручність традиційних хмарних сервісів зберігання, коли користувачі не знаходяться поблизу один одного. Загалом, емпіричні дослідження [1] демонструють, що користувачі, які знаходяться поблизу, ймовірно, зацікавлені в подібному контенті. Наприклад, групи людей на великомасштабних заходах, таких як фестивалі або концерти, швидше за все зацікавлені в обміні своїм досвідом події. Фотографії або відео, зняті з оптимального ракурсу, можуть бути легко поширені серед зацікавлених осіб. З огляду на повсюдність мобільних

пристроїв та їхні зростаючі обчислювальні можливості, додаток використовує ці локальні концентрації інтересів користувачів, що розширює охоплення хмарного сховища для користувачів, які знаходяться офлайн. Обмін у близькому оточенні розширює соціальну мережу користувачів до фізичної близькості, а потім інтегрує фізичну близькість назад у соціальну мережу. Ми вважаємо, що це призведе до значного покращення ефективності розповсюдження мобільного контенту. Ми демонструємо життєздатність додатку через прототипну реалізацію на смартфонах Android, що включає функцію соціального виявлення на основі місцезнаходження та процеси локального "скидання" та отримання контенту.

1.4.1. Огляд системи

Додаток має на меті надати можливість безперебійного обміну контентом з користувачами, що знаходяться поблизу, незалежно від наявності традиційної мережевої інфраструктури. Система досягає цього, використовуючи розширені можливості сучасних смарт-мобільних пристроїв через мобільний додаток, що дозволяє користувачам створювати локальну мережу та співпрацювати з іншими користувачами у безпосередній близькості. Користувачі також матимуть доступ до розділу хмарного сховища, який вони можуть використовувати для співпраці у випадках, коли вони не знаходяться поблизу. Будь-який користувач може ініціювати обмін контентом, активуючи відповідну функцію на пристрої, яка ініціює функцію Wi-Fi Hotspot на пристрої для формування Drop-LAN. Огляд процесу обміну контентом у проілюстровано на рисунку 1.1., який складається з двох фаз: (А Соціальне виявлення та (Б Скидання та отримання контенту, описаних нижче.

А. Соціальне виявлення

Під час інсталяції кожен користувач запрошується до реєстрації на сервері управління контентом (CMS) для отримання доступу до розділу

хмарного сховища (DropCloud). Цей розділ може бути використаний для розміщення контенту та/або як сервіс резервного копіювання даних.

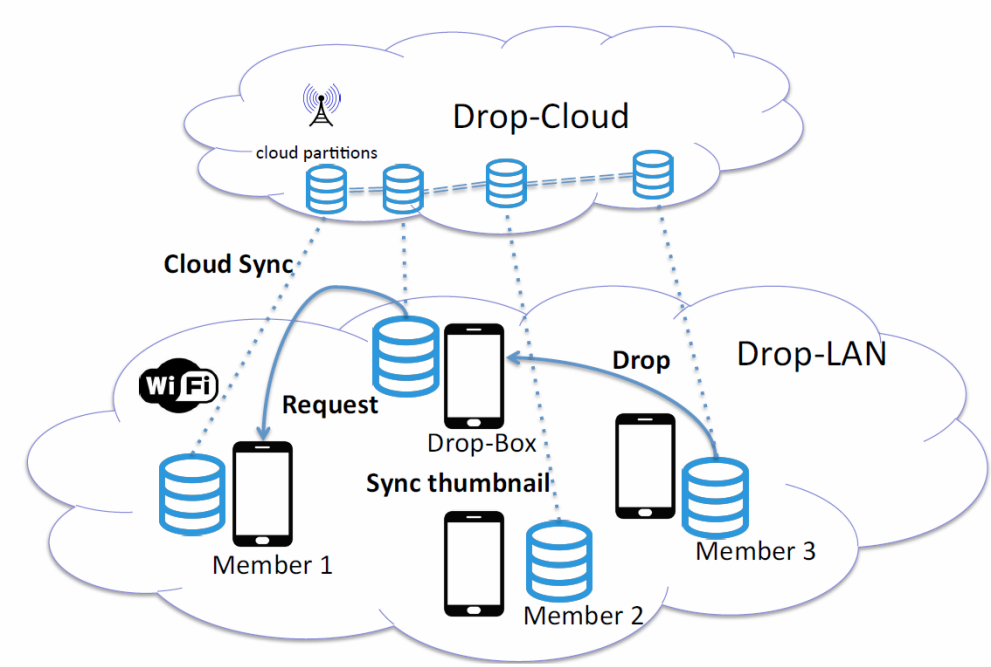

Рисунок 1.1 - Огляд процесу обміну конвентом в пропонованій системі

Додаток періодично передає дані про місцезнаходження користувачів на сервер, щоб сприяти виявленню інших користувачів у порядку географічної близькості до поточного місцезнаходження користувача або на карті. Користувачі можуть запрошувати інших користувачів, що знаходяться поблизу, до формування мережі для ініціації локального обміну контентом. Додаток також інформує користувача про необхідність наблизитися до Drop-Box, якщо користувач знаходиться поза межами досяжності мережі. У разі відсутності підключення до Інтернету, користувачі, що знаходяться поблизу, можуть бути виявлені лише за наступних умов:

1) у безпосередній близькості існує активна мережі

2) шляхом активації мережі на пристрої.

Крім того, список користувачів, що знаходяться поблизу, обмежується користувачами в межах діапазону зв'язку мережі (LAN). Більше того, додаток дозволяє своїм користувачам запам'ятовувати раніше підключених

користувачів та додавати їх як друзів, формуючи мережу обміну контентом на основі місцезнаходження.

Б. Скидання та отримання контенту

Кожен учасник мережі може "скинути" будь-який контент у підключений Drop-Box. Більш конкретно, пристрій, що хостить Drop-Box, отримує метаінформацію будь-якого "скинутого" контенту його учасником, включаючи мініатюру для контенту. Drop-Box робить цю інформацію доступною для всіх підключених учасників. Учасники мережі також можуть обмінюватися контентом безпосередньо, використовуючи Drop-Box для мінімальної затримки, і цей контент буде передано до Drop-Box пізніше. Фактичний контент буде завантажено до Drop-Box, якщо учасник "скине" певний контент або контент буде передано після отримання. Локальні передачі даних покращують якість користувацького досвіду, заощаджуючи мобільну пропускну здатність для користувача.

Всі додатки періодично оновлюють CMS метаінформацією про підключені пристрої, коли вони підключені до Інтернету. Це дозволяє CMS створити віртуальну мережу, налаштувавши відповідні права доступу між розділами Drop-Cloud. Користувачі також можуть співпрацювати над одним і тим же контентом, де CMS буде виконувати резервне копіювання кількох версій контенту. Коли користувачі підключені до свого Drop-Cloud через мережу з низькою вартістю, наприклад, WLAN, додаток завантажує контент, який зберігається на локальному Drop-Box, щоб зробити контент доступним онлайн для учасників віртуальної мережі.

1.5. Огляд схожих програмних рішень

MoodChat — це інноваційне програмне забезпечення, розроблене для фасилітації ефективної комунікації та взаємодії між користувачами, особливо зосереджене на обміні емоційними станами та покращенні ментального

благополуччя. Основна концепція MoodChat полягає у створенні безпечного та підтримуючого цифрового середовища, де користувачі можуть виражати свої настрої, отримувати підтримку від однолітків та експертів, а також відстежувати зміни у своєму емоційному стані з плином часу.

Рисунок 1.2 – Вигляд платформи MoodChat

Основні функціональні можливості:

- Користувачі мають можливість щоденно або за вимогою фіксувати свій поточний настрій за допомогою інтуїтивно зрозумілого інтерфейсу. Це може включати вибір з набору емоцій (наприклад, радість, смуток, тривога, спокій) або використання шкали інтенсивності. Система збирає ці дані для формування персоналізованих графіків та статистики, що дозволяє користувачам візуалізувати динаміку своїх емоційних станів та виявляти певні закономірності чи тригери.

MoodChat надає простір для обміну емоційними станами з обраними друзями, групами підтримки або спільнотами. Це може відбуватися через приватні та групові чати.

Платформа інтегрує бібліотеку ресурсів, що стосуються ментального благополуччя. Це можуть бути:

- Статті та блоги про техніки управління стресом, методи розслаблення, когнітивно-поведінкову терапію (КПТ) та інші аспекти психології.

- Аудіо- та відео матеріали, медитації, дихальні вправи, подкасти від психологів.

- Інтерактивні сесії для покращення настрою або розвитку навичок емоційної регуляції.

На основі даних про настрій користувача та його взаємодії, MoodChat може надавати персоналізовані рекомендації щодо контенту, вправ або спільнот, які можуть бути корисними для покращення його емоційного стану. Це може включати пропозиції щодо медитацій, порад з управління стресом або рекомендацій приєднатися до певної групи підтримки.

MoodChat розроблений з використанням сучасних технологій, що забезпечують його масштабованість, надійність та безпеку. Зазвичай такі системи базуються на клієнт-серверній архітектурі, де мобільні додатки (клієнти) взаємодіють з бекенд-сервісами, розміщеними в хмарній інфраструктурі. Це дозволяє ефективно обробляти великі обсяги даних, забезпечувати високу доступність та гнучко масштабуватися відповідно до зростаючої кількості користувачів.

Для зберігання даних про користувачів, їхні настрої, повідомлення та контент використовується база даних, яка може бути як реляційною, так і NoSQL, в залежності від специфічних вимог до гнучкості та швидкості доступу до даних. Механізми шифрування даних та аутентифікації користувачів є критично важливими для забезпечення конфіденційності та безпеки персональної інформації, особливо з огляду на чутливість даних про ментальне здоров'я.

Протокол DISSEMINATE призначений для ефективного розповсюдження даних шляхом поділу кожного елемента даних на менші фрагменти (chunks). Ці фрагменти можуть бути доставлені не за порядком і

зібрані на приймальному кінці. Коли пристрої обмінюються такими фрагментами замість цілих елементів даних, втрата одного фрагмента є значно менш критичною, ніж втрата всього елемента даних. Більше того, даний пристрій може збирати фрагменти від кількох різних безпосередньо підключених пристроїв, замість того, щоб отримувати весь елемент даних від одного пристрою. Підхід до обміну фрагментами реалізований у вигляді протоколу типу "публікація-підписка" (publish-subscribe), що використовує комбінацію оголошень, підписок та публікацій. Загальний огляд етапів роботи DISSEMINATE представлено на рисунку 1.3.

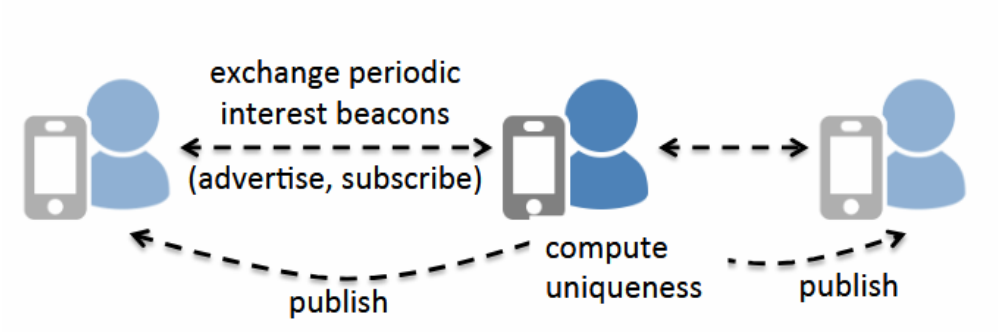

Рисунок 1.3 - Огляд етапів роботи DISSEMINATE

Тут пропонується оголошення та підписку в один крок. Зокрема, пристрої періодично надсилають маяки (beacons) з дуже легковаговим представленням:

1) елементів даних, що їх цікавлять,

2) фрагментів цих елементів даних, які вони вже зібрали.

Остання частина є оголошенням: ці фрагменти представляють контент, який пристрій може розповсюджувати. Обидві частини разом утворюють підписку: пристрій, що отримує маяк, може визначити, які фрагменти необхідно доставити пристрою-відправнику. Ми розробили схему для визначення того, які дані публікувати, ґрунтуючись на метриці, яку ми називаємо унікальністю: інтуїтивно, наш механізм публікації намагається збалансувати два аспекти:

1) надсилання фрагментів, які не є широко доступними в безпосередній близькості,

2) надсилання фрагментів, які, ймовірно, будуть отримані передбачуваним підписником (підписниками).

Для першого аспекту вимірюють ступінь надмірності конкретного фрагмента і обираємо менш надлишкові фрагменти. Для другого, ми віддаємо перевагу фрагментам, які потрібні сусідам, чиї з'єднання мають високу якість зв'язку, виміряну за допомогою RSSI (Received Signal Strength Indicator). У DISSEMINATE кожен пристрій працює незалежно, приймаючи найкраще очевидне рішення на основі інформації, отриманої від маяків сусідніх пристроїв.

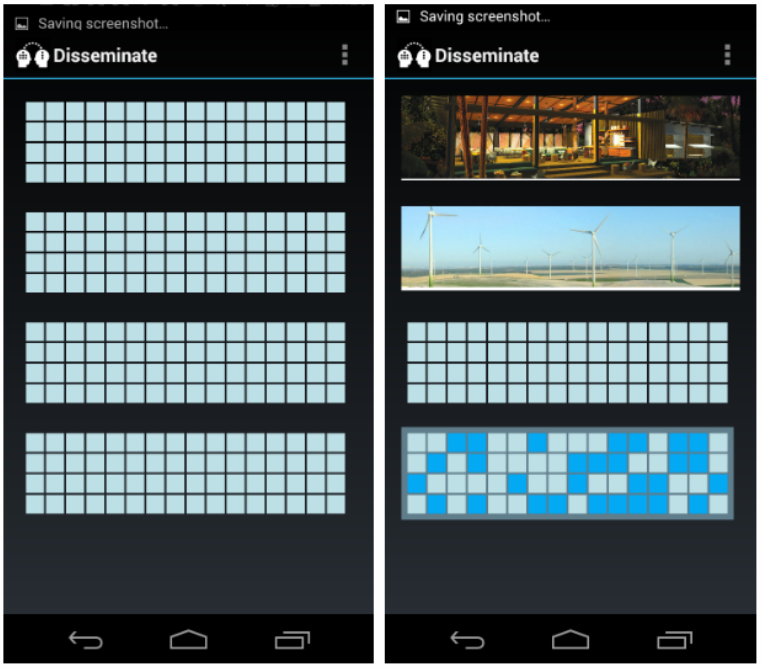

Рисунок 1.4 – Скріншоти додатку DISSEMINATE

На рисунку 1.4 представлені два скріншоти додатку DISSEMINATE. Ліворуч показано стан, коли користувач ще не ініціював жодного завантаження: пристрій підписаний на отримання всіх чотирьох елементів даних, але ще не отримав жодного фрагмента. Праворуч видно, що два завантаження завершені, а одне перебуває в процесі.

Коли додаток запускається, користувач побачить екран, зображений ліворуч на рис. 1.4, на кожному пристрої, де додаток увімкнено.

Якщо користувач одного з пристроїв торкнеться однієї з сіток "цеглинок", розпочнеться "завантаження" цього файлу з (симульованої) інфраструктури. Учасники зможуть регулювати швидкість завантаження, імітуючи як високо-, так і низькоякісне стільникове з'єднання. Оскільки всі активні пристрої підписані на всі елементи даних, фрагменти елемента даних під час "завантаження" негайно передаватимуться на всі підключені пристрої. Учасники спостерігатимуть, як сітки, що представляють фрагменти для кожного елемента даних, заповнюватимуться в міру їхнього надходження. Коли весь елемент даних буде отримано, буде відображено медіафайл.

За допомогою додатку DISSEMINATE учасники зможуть експериментувати з різними налаштуваннями, щоб спостерігати, коли спільне розповсюдження є ефективним, а коли — менш ефективним. Наприклад, учасники зможуть перевірити, що відбувається, коли кілька пристроїв завантажують один і той самий файл з інфраструктури на дуже низьких швидкостях стільникового зв'язку, а потім діляться фрагментами в процесі завантаження (тобто, кооперативне завантаження принесе користь усім користувачам).

РОЗДІЛ 2. РЕАЛІЗАЦІЯ АРХІТЕКТРИ ТА АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ СОЦІАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОБМІНУ КОНТЕНТОМ

2.1. Функціональна архітектура взаємодії мікросервісів

Система розроблена відповідно до парадигми мікросервісної архітектури. Це означає, що її функціональність розподілена між незалежними хмарними сервісами, які взаємодіють між собою, але можуть функціонувати автономно. Принципова схема цієї архітектури представлена на рисунку 2.1. Така декомпозиція забезпечує високу модульність, масштабованість та відмовостійкість системи, дозволяючи незалежне розгортання, оновлення та управління кожним сервісом.

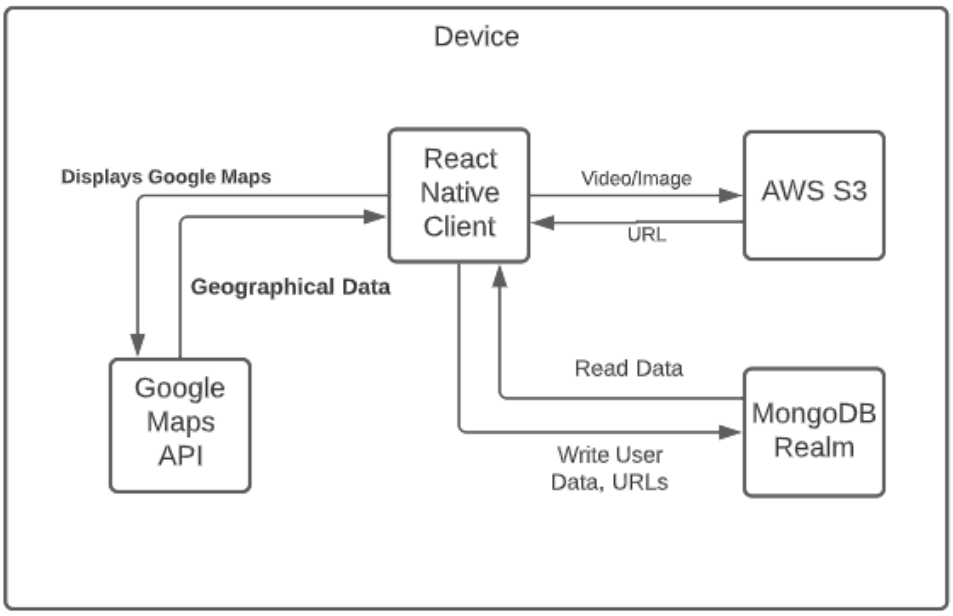

Рисунок 2.1 - Схема взаємодії клієнта додатку з хмарними сервісами

Схема відображає взаємодію між клієнтським додатком, що працює на мобільному пристрої, та різними хмарними компонентами.

Розглянемо основні компоненти схеми.

1. Device. Представляє кінцевий мобільний пристрій (смартфон або планшет), на якому працює клієнтський додаток.

2. React Native Client.

- Це основний клієнтський компонент, розроблений за допомогою фреймворку React Native.

- Він відповідає за користувацький інтерфейс та всю логіку взаємодії на стороні пристрою.

- Виступає центральним вузлом, що координує обмін даними з усіма зовнішніми хмарними сервісами.

3. Google Maps API.

Цей сервіс відповідає за надання картографічних функцій та географічних даних для додатку.

Клієнт React Native відправляє запити до Google Maps API для отримання та відображення карт на екрані пристрою. Google Maps API надає клієнту React Native географічні дані, такі як координати місцезнаходження, інформація про об'єкти на карті тощо. Це дозволяє реалізувати функцію "скидання" контенту в певних місцях.

4 AWS S3.

- Сервіс хмарного зберігання даних, що надається Amazon.

- Використовується для зберігання медіаконтенту (відео та зображень), які "скидаються" користувачами.

Клієнт React Native завантажує відео- та зображення-файли безпосередньо до AWS S3. AWS S3 повертає клієнту React Native URL-посилання на завантажений контент. Ці посилання потім можуть бути збережені в базі даних (MongoDB Realm) та використовуватися для доступу до медіафайлів іншими користувачами.

5. MongoDB Realm - хмарна платформа, яка надає NoSQL базу даних (MongoDB Atlas), а також функції для бекенд-логіки, аутентифікації користувачів та синхронізації даних.

Клієнт React Native записує до MongoDB Realm різноманітні дані, включаючи інформацію про користувачів, метадані "Дропів" та URL-посилання на медіафайли, що зберігаються в AWS S3. Клієнт React Native зчитує необхідні дані (наприклад, інформацію про "Дропи", що знаходяться поблизу, профілі друзів) з MongoDB Realm для відображення в інтерфейсі користувача.

Опишемо загальний принцип роботи.

Користувач взаємодіє з додатком через React Native Client. Коли користувач хоче "скинути" контент (наприклад, відео або зображення) у певне місце, клієнт:

1. Використовує Google Maps API для визначення та відображення географічного місцезнаходження.

2. Завантажує медіафайл до AWS S3 і отримує у відповідь URL.

3. Зберігає дані про "Дроп" (включаючи URL-посилання на медіафайл, місцезнаходження, інформацію про одержувачів) у MongoDB Realm.

Коли інший користувач наближається до місця "Дропу" або хоче переглянути контент, клієнт:

1. Зчитує відповідні дані з MongoDB Realm.

2. Використовує отримані URL-посилання для завантаження медіафайлів з AWS S3 та їх відображення.

3. Взаємодіє з Google Maps API для відображення "Дропів" на карті.

Ця архітектура забезпечує гнучкість, масштабованість та ефективну обробку даних завдяки розподілу функціональності між спеціалізованими хмарними сервісами.

2.2. Проектування карти потоку інтерфейсу користувача

Користувацький інтерфейс (UI) розроблений для забезпечення інтуїтивної взаємодії та доступу до широкого спектра функцій.

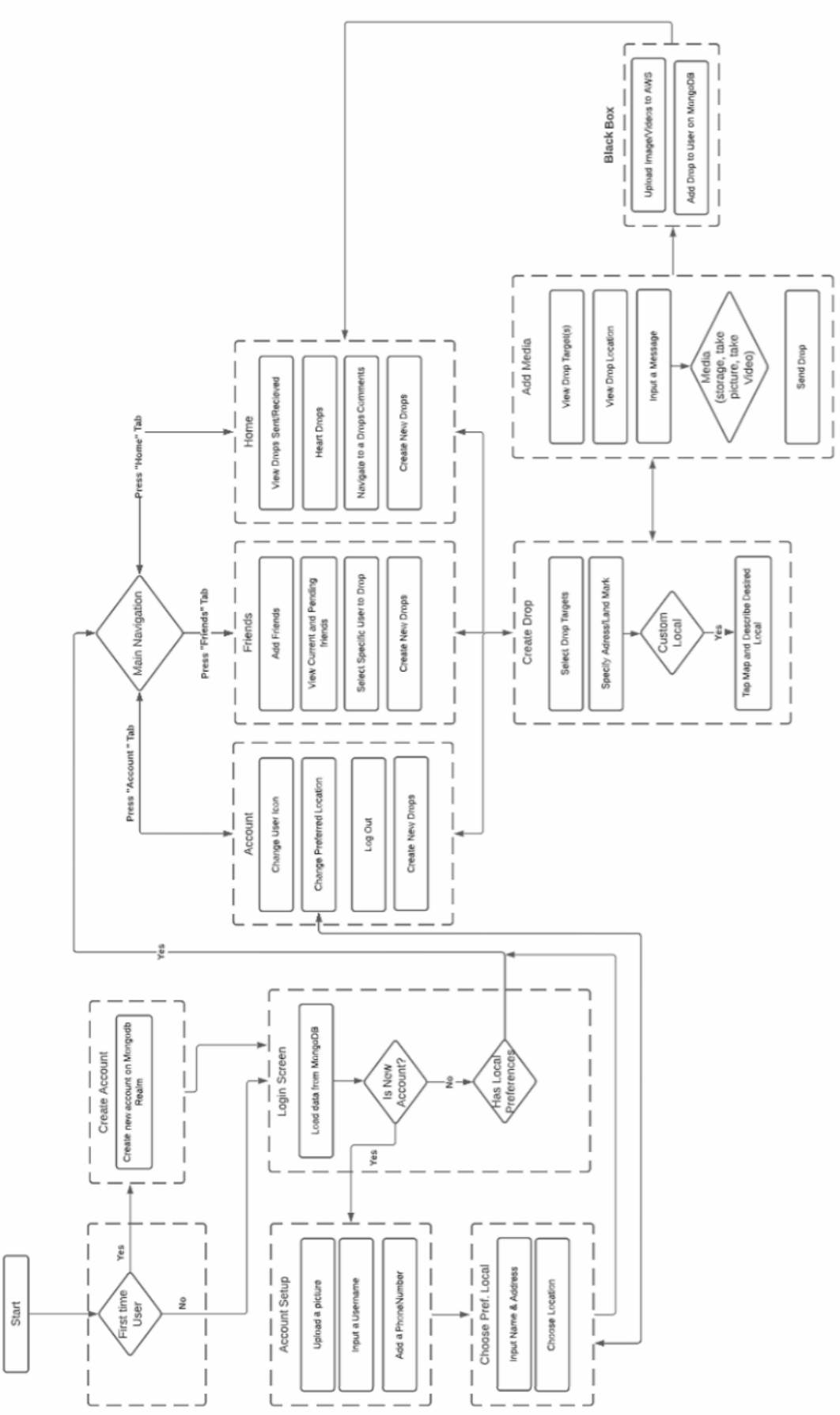

Рисунок 2.2 - Карта потоку інтерфейсу

Він складається з множини сторінок, між якими користувач може здійснювати навігацію. Загальний огляд карти потоку інтерфейсу відображено на рисунку 2.2.

Розглянемо етапи та функціональні блоки подані на рисунку.

1. Start

- Початкова точка входу до додатку.
- First Time User?: Система перевіряє, чи є користувач новим.
  - Yes: Перенаправляє до Create Account.
  - No: Перенаправляє до Login Screen.

2. Create Account

Create new account on MongoDB Realm: Реєстрація нового користувача в системі. Після успішної реєстрації користувач перенаправляється на Account Setup.

3. Login Screen

- Load data from MongoDB: Завантаження даних існуючого користувача.
  - Is New Account?:
    - Yes: Перенаправляє до Account Setup. Це забезпечує, що навіть якщо користувач вже зареєстрований, але ще не завершив налаштування, він буде направлений туди.
    - No: Перенаправляє до Main Navigation (Головна навігація).
- Tune Local Preferences: Можливість налаштування локальних переваг для вже існуючого користувача.

4. Account Setup:

- Цей блок охоплює початкові налаштування облікового запису для нових користувачів.
- Upload a picture: Завантаження зображення профілю.
- Input a username: Введення унікального імені користувача.
- Add A Phone/Number: Додавання контактного номера телефону.

- Choose Pref. Local: Вибір бажаного місцезнаходження.

- Input Name & Address: Введення особистих даних.

- Choose Location: Вибір географічного місцезнаходження.

- Після завершення налаштування користувач переходить до Main Navigation.

5. Main Navigation

- Центральний хаб, з якого користувач може перейти до основних розділів додатку.

- Press "Account" Tab: Перехід до розділу Account.

- Press "Friends" Tab : Перехід до розділу Friends.

- Press "Home" Tab: Перехід до розділу Home.

6. Account

- Change Main Icon: Можливість змінити іконку профілю.

- Change Preferred Location: Оновлення бажаного місцезнаходження.

- Log Out: Вихід з облікового запису.

- Create New Drops: Прямий перехід до створення "Дропів".

7. Friends

- Add Friends: Функціонал для додавання нових друзів.

- View Current and Pending Friends: Перегляд списку друзів та запитів на дружбу.

- Select Specific User To Drop: Можливість вибрати одержувача для "Дропу".

- Create New Drops: Прямий перехід до створення "Дропів".

8. Home

- View Drops (Sent/Received): Перегляд історії "Дропів".

- Hear Drops: Функція, що стосується аудіо-контенту або сповіщень про "Дропи".

- Maps to the Drops Comments: Перехід до розділу коментарів певного "Дропу".

- Create New Drops: Прямий перехід до створення "Дропів".

9. Create Drop

- Це ключовий функціональний блок для створення нового "Дропу".
- Select Drop Targets: Вибір одержувачів "Дропу".
- Specify Admin/Land Mark: Вказання певного місця для "Дропу".
- Custom Local:
    - Yes: Tag Map and Describe Desired Local.
    - No: Просто вказати точку.
- Після визначення місцезнаходження відбувається перехід до Add Media.

10. Add Media (Додати медіа):

- View Drop Target(s): Перегляд вибраних одержувачів.
- View Drop location: Перегляд вибраного місцезнаходження.
- Input a Message: Введення текстового повідомлення.
- Media (pre-recorded, take picture, take Video(s)): Можливість додати медіафайл (з галереї або зробити новий).
- Send Drop: Кінцевий етап надсилання "Дропу". Після надсилання відбувається взаємодія з "Чорним ящиком".

11. Black Box

- Цей блок представляє внутрішні системні операції, які не є частиною прямої взаємодії користувача з UI, але є критично важливими для функціонування "Дропів".
- Upload Image/Video via S3/AWS: Завантаження медіафайлів у хмарне сховище.
- Add Drop to User on MongoDB: Збереження інформації про "Дроп" у базі даних.

Деякі переходи автоматично визначаються системою на основі даних користувача (наприклад, для нових облікових записів). Переважна більшість переходів між сторінками є двосторонніми, що забезпечує користувачеві

гнучкість у переміщенні по додатку. Проте, існують специфічні сценарії, коли клієнтська частина програмного забезпечення автоматично визначає наступну сторінку для відображення, ґрунтуючись на аналізі даних користувача. Наприклад, у випадку, якщо системні дані підтверджують, що користувач здійснив вхід до системи з новоствореного облікового запису, він буде автоматично перенаправлений до сторінки налаштування облікового запису. Цей підхід оптимізує початковий досвід користувача та забезпечує його цілеспрямовану взаємодію з необхідними функціями.

2.3. Аналіз цільової аудиторії та системних обмежень програмного забезпечення

2.3.1. Цільова аудиторія користувачів

Додаток орієнтований на широкий спектр користувачів, які прагнуть до комунікації з друзями, родиною, шанувальниками, підписниками або однодумцями. Основною метою є надання інноваційної перспективи у взаємодії в соціальних мережах, що буде доступна будь-якій особі, що володіє розумним мобільним пристроєм. Цільова аудиторія включає осіб, які бажають підвищити рівень залученості до контенту, яким вони обмінюються. Аналогічно, додаток призначений для користувачів, які прагнуть бути проактивними в отриманні нового контенту від осіб, яких вони високо цінують, таких як близькі друзі, члени родини або публічні особи. На поточний момент вікові обмеження для користувачів відсутні.

2.3.2. Обмеження системи

У функціональності передбачено кілька ключових обмежень:

1. Термін зберігання контенту: "Дропи" (повідомлення) будуть автоматично видалятися з системи після 30-денного періоду з моменту їх створення, після чого вони стануть недоступними для користувачів. Це

обмеження може бути зумовлене оптимізацією ресурсів зберігання та забезпеченням актуальності контенту.

2. Модель монетизації: Додаток буде надаватися для безкоштовного використання, проте його функціонування включатиме інтеграцію рекламних оголошень Google. Ця модель дозволяє підтримувати розробку та експлуатацію платформи без прямих фінансових витрат для користувачів.

3. Технічні обмеження проекту: В рамках поточного проекту до ключових обмежень належить необхідність забезпечення:

- Надійної масштабованості (reliable scalability): Здатність системи ефективно обробляти зростаючу кількість користувачів та обсягів даних без суттєвого зниження продуктивності.

- Ефективності за вартістю (cost-effectiveness): Оптимізація витрат на розробку, розгортання та підтримку інфраструктури, зокрема хмарних сервісів.

- Чудового користувацького досвіду (excellent user experience): Забезпечення інтуїтивно зрозумілого, швидкого та приємного інтерфейсу для всіх користувачів.

2.4. Моделювання архітектури та функціональних вимог системи

2.4.1. Діаграми класів та варіантів використання

Діаграма класів (представлена на рис. 2.3) надає візуальне представлення взаємозв'язків між основними сутностями системи: Користувачами, Друзями, Повідомленнями та Сповіщеннями. Загалом, об'єкт класу "Користувач" може мати від нуля до множини зв'язків з об'єктами класів "Друг", "Сповіщення" або "Повідомлення Drop", що залежить від їхньої активності в системі. У свою чергу, об'єкти класу "Повідомлення Drop" можуть агрегувати медіа-контент і підтримувати зворотний зв'язок у формі "Сердечок" (Likes) або "Коментарів".

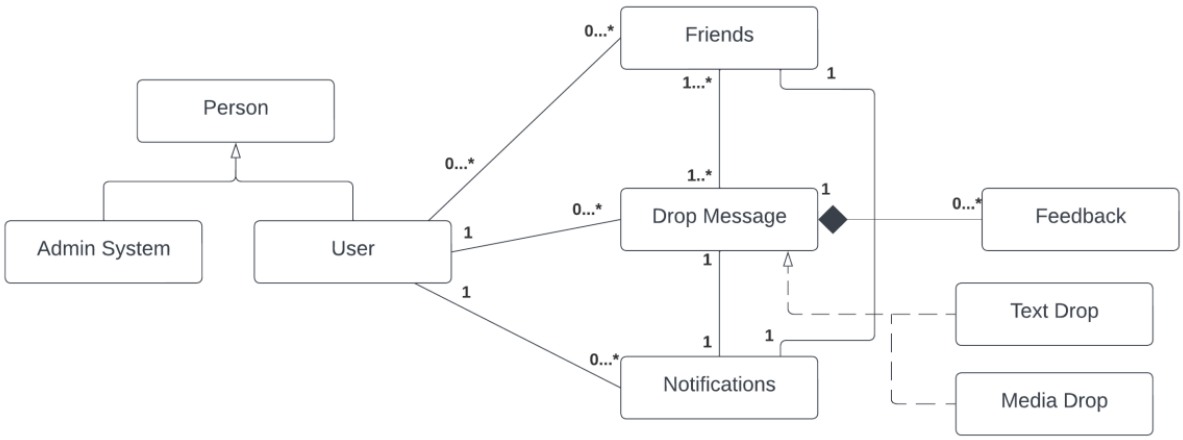

Рисунок 2.3 – Діаграма класів

Проведемо опис основних класів та їхніх відносин.

1. Person

- Це абстрактний базовий клас (суперклас), що є узагальненням для всіх видів "осіб" у системі. Він містить загальні атрибути, такі як ім'я, ID тощо.

2. Admin System

- Це підклас (субклас) класу `Person`. Це означає, що `Admin System` є різновидом `Person` і успадковує її властивості. Клас `Admin System` представляє сутність, яка має адміністративні привілеї в системі.

3. User

- Це також підклас класу `Person`. `User` представляє звичайного користувача системи, який взаємодіє з її основними функціями.

Взаємозв'язки класу "User" з іншими класами:

- User - Friends (Друзі):

- Між класами `User` та `Friends` існує асоціація.

- Кратність: `User` може мати `0..-` (від нуля до багатьох) об'єктів класу `Friends`. Це означає, що один користувач може мати багато друзів або не мати жодного. `Friends` (що тут є сукупністю друзів або зв'язком дружби) пов'язаний з `1` (один) `User`.

- User - Drop Message (Повідомлення Drop):

- Між `User` та `Drop Message` існує асоціація.

- Кратність: `User` може бути пов'язаний з `0..-` (від нуля до багатьох) `Drop Message`. Це означає, що користувач може створити багато "Дропів" або не створювати жодного. `Drop Message` пов'язаний з `1` (один) `User`, що вказує на те, що кожен "Дроп" має одного творця/відправника.

- User - Notifications (Сповіщення):

- Між `User` та `Notifications` існує асоціація.

- Кратність: `User` може мати `0..-` (від нуля до багатьох) `Notifications`. Це означає, що користувач може отримувати багато сповіщень або не отримувати жодного. `Notifications` пов'язаний з `1` (один) `User`, що вказує на те, що кожне сповіщення призначене одному користувачеві.

Взаємозв'язки класу "Drop Message" з іншими класами:

- Drop Message - Feedback (Зворотний зв'язок):

- Між `Drop Message` та `Feedback` існує асоціація.

- Кратність: `Drop Message` може мати `0..-` (від нуля до багатьох) `Feedback`. Це означає, що один "Дроп" може отримати багато відгуків або не отримати жодного.

- Drop Message - Friends (Друзі):

- Між `Drop Message` та `Friends` існує асоціація.

- Кратність: `Drop Message` може бути пов'язаний з `1..-` (від одного до багатьох) `Friends`. Це може вказувати на те, що "Дроп" може бути адресований одній або багатьом групам "Друзів", або що "Дроп" асоційований з певними зв'язками дружби.

- Drop Message - Notifications (Сповіщення):

- Між `Drop Message` та `Notifications` існує асоціація.

- Кратність: `Drop Message` пов'язаний з `1` (один) `Notifications`. Це може означати, що створення або отримання "Дропу" завжди генерує одне сповіщення.

Клас `Drop Message` є абстрактним класом або інтерфейсом.

- Text Drop - це підклас `Drop Message`, що представляє "Дроп", який містить лише текстовий контент.

- Media Drop - це підклас `Drop Message`, що представляє "Дроп", який містить медіа-контент (зображення, відео тощо).

Розроблена діаграма класів надає високорівневе уявлення про основні сутності та їхні взаємозв'язки в проектованій системі. Вона демонструє, як користувачі можуть взаємодіяти з друзями, створювати різні типи "Дропів", отримувати сповіщення та надавати зворотний зв'язок. Використання узагальнення (`Person` та `Drop Message`) допомагає структурувати модель, дозволяючи перевикористовувати загальні властивості та поведінку.

2.4.2. Розробка діаграм варіантів використання

Діаграма варіантів використання (представлена на рисунку 2.4) ілюструє взаємодії між системою та її зовнішнім середовищем, визначаючи ролі акторів та їхні функціональні можливості.

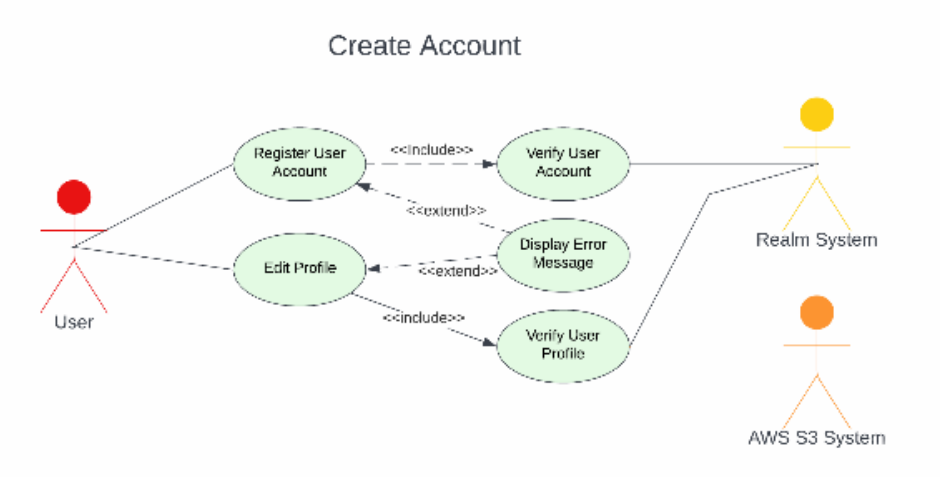

Рисунок 2.4 – Діаграма варіантів використання (Create Account)

1. Система Realm:

- Відповідає за управління обліковими записами користувачів та їхніми профілями.

- Реєстрація нового облікового запису: Система Realm отримує запит на реєстрацію облікового запису, здійснює валідацію наданих облікових даних та приймає або відхиляє запит на основі результатів перевірки.

- Редагування профілю користувача: Система Realm верифікує рівень дозволів користувача та виконує оновлення його профільної інформації.

- Перевірка користувача: Система Realm здійснює аутентифікацію користувачів та перевірку їхніх рівнів дозволів для надання відповідного доступу до даних.

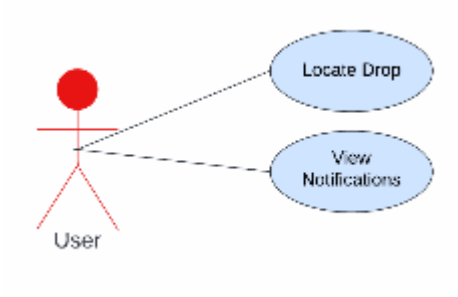

Рисунок 2.5 – Діаграма варіантів використання (user)

2. Користувач:

Є основним актором, який виконує всі перелічені нижче функції.

- Реєстрація: Для використання функціоналу користувачеві необхідна реєстрація з наданням валідованих облікових даних.

- Додавання/видалення друзів: Користувачі мають можливість надсилати запити на дружбу та видаляти існуючих друзів у системі.

- Створення Drop: Користувачі, що мають друзів, можуть створювати "Дропи", які містять медіа-контент або текст і призначені для отримання у зазначеному географічному місцезнаходженні.

- Локалізація заблокованих Drop: Користувачі, які є одержувачами "Дропу", мають можливість локалізувати його для подальшого розблокування.

- Перегляд сповіщень: Користувачі можуть переглядати сповіщення щодо запитів на дружбу або отримання "Дропів".

3. Система AWS S3:

Відповідає за перевірку та зберігання медіафайлів, пов'язаних з профілями користувачів та контентом "Дропів".

4. Google Maps API:

Відповідає за валідацію всіх географічних місцезнаходжень, які використовуються в системі.

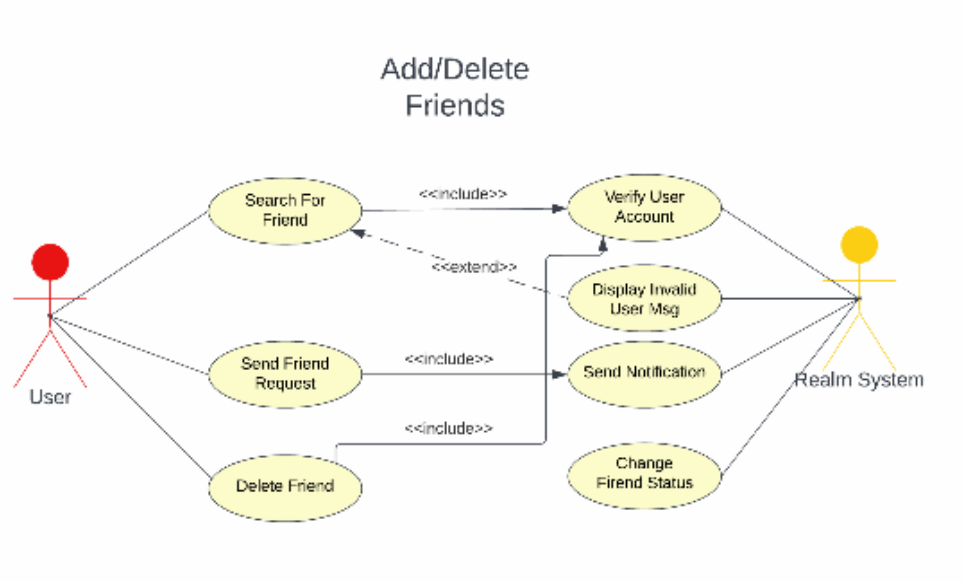

Рисунок 2.6 – Діаграма варіантів використання (Add/Delete Friends)

Рисунок 2.6 є діаграмою варіантів використання, що ілюструє функціональність системи, пов'язану з управлінням друзями, та взаємодію між основними акторами та системою.

Основні актори:

1. User - основний актор, який ініціює дії, пов'язані з додаванням або видаленням друзів.

2. Realm System (Система Realm) - системний актор, який представляє бекенд-систему (зокрема, базу даних MongoDB Realm та її сервіси), що відповідає за управління обліковими записами користувачів, їхніми даними та обробку запитів.

Діаграма фокусується на трьох основних варіантах використання, що ініціюються користувачем:

1. Search For Friend - цей варіант використання дозволяє користувачеві знаходити інших користувачів у системі.

Зв'язок <<include>> з Verify User Account означає, що Пошук Друга завжди включає в себе перевірку облікового запису користувача. Це необхідно для підтвердження існування користувача або його статусу.

Зв'язок <<extend>> з Display Invalid User Msg - вказує на те, що умовою для розширення (extension point) є ситуація, коли "Пошук Друга" призводить до недійсного результату (користувача не знайдено). У такому випадку система додатково виконує варіант використання "Відобразити Повідомлення про Недійсного Користувача".

2. Send Friend Request - цей варіант використання дозволяє користувачеві надсилати запит на дружбу іншому користувачеві.

Зв'язок <<include>> з Send Notification означає, що надсилання запиту на дружбу завжди супроводжується надсиланням сповіщення одержувачу запиту.

Зв'язок <<include>> з Verify User Account вказує на те, що перед надсиланням запиту на дружбу необхідно перевірити обліковий запис цільового користувача.

3. Delete Friend - варіант використання дозволяє користувачеві видалити існуючого друга зі свого списку.

Зв'язок <<include>> з Change Friend Status означає, що видалення друга завжди включає зміну статусу дружби між користувачами в системі.

Варіанти Використання, що виконуються "Системою Realm":

1. Verify User Account

Виконується "Системою Realm". Цей варіант використання перевіряє наявність користувача, його активність, тощо.

Ініціюється варіантами використання "Пошук Друга" та "Надіслати Запит на Дружбу".

2. Display Invalid User Msg:

Виконується "Системою Realm" у відповідь на певну умову.

Розширює варіант використання "Пошук Друга".

3. Send Notification

Виконується "Системою Realm".

Ініціюється варіантом використання "Надіслати Запит на Дружбу".

4. Change Friend Status (Змінити Статус Друга):

Виконується "Системою Realm".

Ініціюється варіантом використання "Видалити Друга".

Загалом, діаграма відображає основну функціональність управління друзями, розподіл відповідальності між користувачем та системним бекендом (Realm System), а також залежності між різними операціями в рамках процесу додавання та видалення друзів.

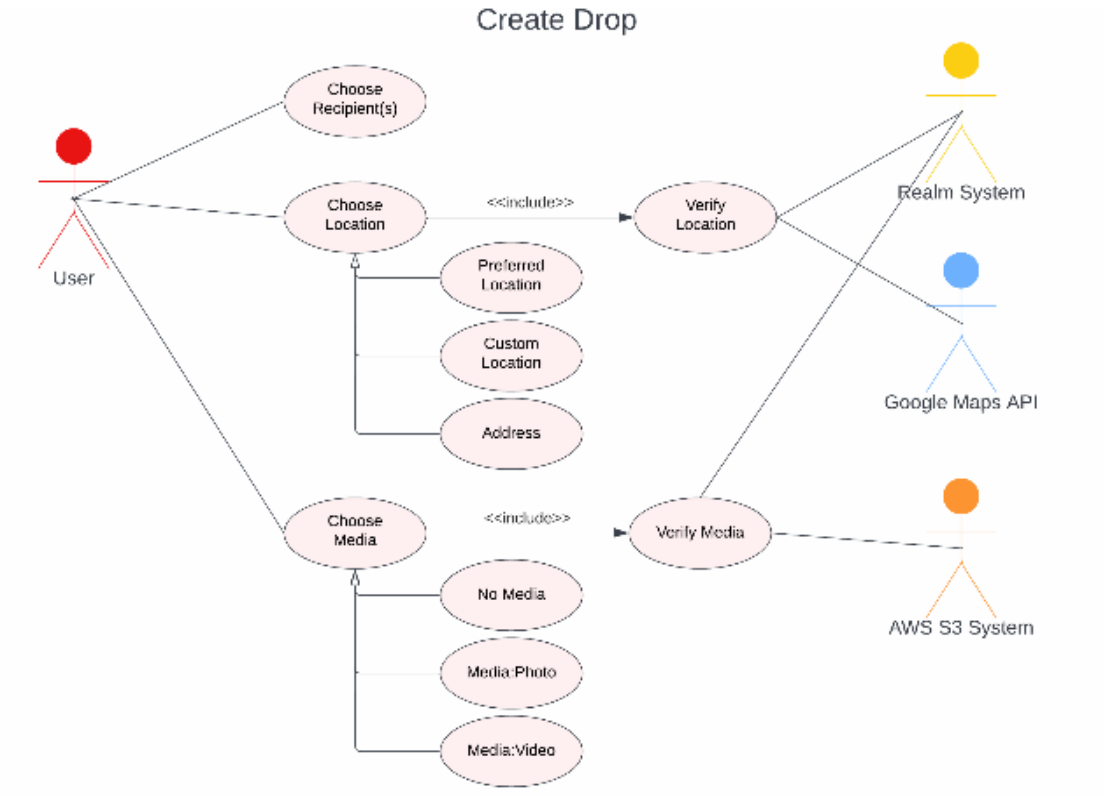

Рисунок 2.7 – Діаграма варіантів використання (Create Drop)

Рисунок 2.7 деталізує процес створення "Дропу" (Drop) у системі, показуючи взаємодію між користувачем, системою Realm, Google Maps API та AWS S3.

Основні Актори:

1. User

Основний актор, який ініціює та виконує дії для створення "Дропу".

2. Realm System

Системний актор, відповідальний за верифікацію даних, управління обліковими записами та збереження інформації про "Дропи".

Google Maps API

Системний актор, відповідальний за надання та верифікацію географічних даних.

AWS S3 System

Системний актор, відповідальний за зберігання медіафайлів.

Діаграма фокусується на головному варіанті використання Create Drop, який ініціюється користувачем. Цей варіант використання складається з декількох під-варіантів та розширень:

1. Choose Recipient(s) - дозволяє користувачеві вказати одного або кількох одержувачів для створюваного "Дропу".

2. Choose Location - цей варіант використання дозволяє користувачеві визначити географічне місце для "Дропу". Він має розширення, що показують різні способи вибору місцезнаходження:

- Preferred Location - вибір зі збережених або часто використовуваних локацій.

- Custom Location - вказання довільного місця.

- Address - введення конкретної адреси.

Зв'язок <<include>> з Verify Location означає, що процес вибору місцезнаходження завжди включає перевірку цього місцезнаходження.

Система Realm перевіряє місцезнаходження (через внутрішню логіку або взаємодію з Google Maps API). Google Maps API безпосередньо залучений до верифікації місцезнаходження, надаючи географічні дані та валідуючи введені координати або адреси.

3. Choose Media - цей варіант використання дозволяє користувачеві прикріпити медіаконтент до "Дропу". Він має розширення, що відображають різні типи медіа:

- No Media: Створення "Дропу" без медіа (наприклад, тільки текстове повідомлення).

- Media Photo: Додавання фотографії.

- Media/Video: Додавання відеофайлу.

Зв'язок <<include>> з Verify Media означає, що процес вибору медіа завжди включає його перевірку (наприклад, на сумісність формату, розмір, можливо, на вміст).

Система AWS S3 залучена до цього кроку для завантаження та зберігання медіа, а також для початкової перевірки перед завантаженням.

Отже, діаграма "Створити Drop" чітко розмежовує відповідальність між клієнтським додатком, бекенд-системою (Realm), картографічним сервісом (Google Maps API) та сервісом хмарного зберігання (AWS S3). Вона деталізує кроки, які користувач повинен виконати для створення "Дропу", включаючи вибір одержувачів, визначення місцезнаходження та додавання медіаконтенту, а також показує, які системні перевірки та взаємодії відбуваються на кожному етапі.

2.4.3. Специфічні вимоги до реалізації

Всі сторінки програмного забезпечення повинні забезпечувати високий рівень користувацького досвіду та надійну функціональність. Очікується, що якість користувацького досвіду буде підвищена за рахунок ретельного дизайну сторінок та інтеграції інтерактивних графічних компонентів, таких

як кнопки, елементи прокрутки або випадаючі меню. Функціональність системи повинна бути реалізована таким чином, щоб забезпечувати обробку всіх можливих випадків збоїв та демонструвати високу стійкість до відмов.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ СОЦІАЛЬНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОБМІНУ КОНТЕНТОМ

3.1. Архітектура інтерфейсу користувача

Інтерфейс користувача розроблюваної системи структурований як набір взаємопов'язаних сторінок, кожна з яких забезпечує доступ до специфічного функціоналу. Ця архітектура спроектована для забезпечення інтуїтивної навігації та ефективної взаємодії користувача з додатком.

Наведемо опис функціональних сторінок.

3.1.1. Сторінка входу та створення облікового запису

Призначена для аутентифікації користувачів, які вже мають обліковий запис. Після успішної авторизації користувач автоматично перенаправляється на головну сторінку. Надає можливість переходу на сторінку створення облікового запису для нових користувачів (рис 3.1).

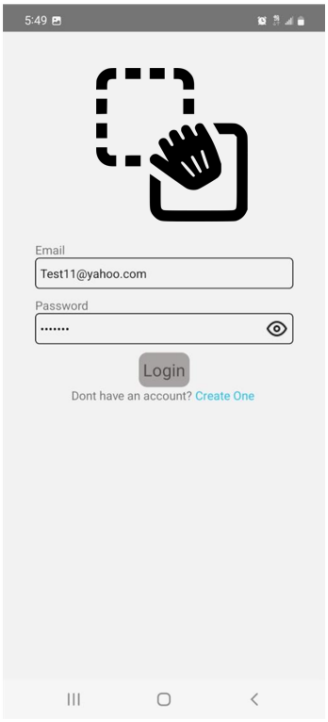

Рисунок 3.1 - Сторінка входу

Сторінка створення облікового запису (Account Creation Page) дозволяє новим користувачам зареєструватися, вводячи електронну пошту та пароль, які будуть асоційовані з їхнім обліковим записом. Після успішної реєстрації користувач повертається на сторінку входу (рис. 3.2).

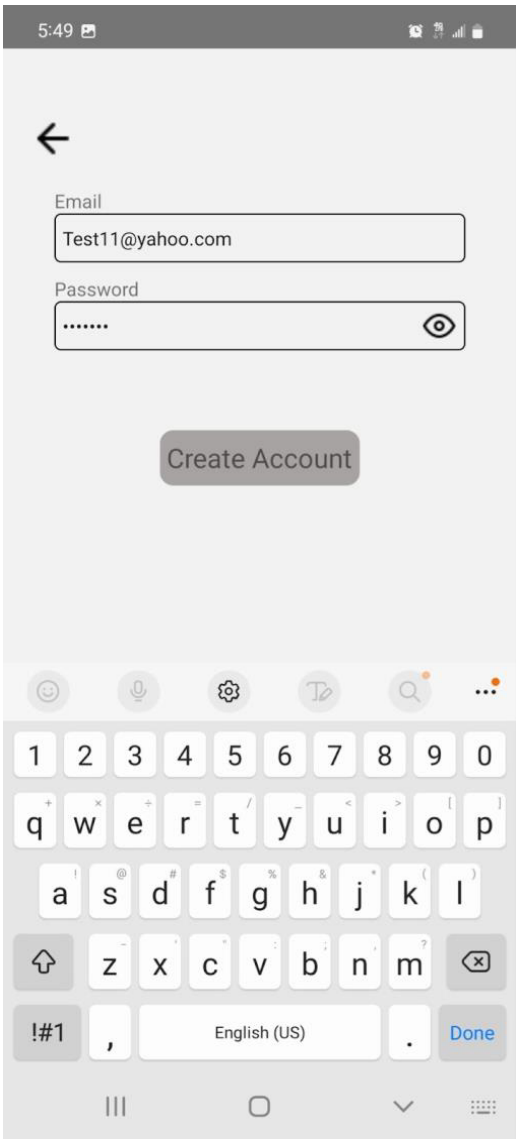

Рисунок 3.2 - Сторінка створення облікового запису

Після успішного створення екземпляра в Realm, клієнтський додаток перенаправляє користувача назад на сторінку входу, де він може авторизуватися у своєму новоствореному обліковому записі. Додаток завершує процес створення нового облікового запису користувача, скеровуючи його на сторінку вітання. На цій сторінці користувачі можуть

ввести ім'я для відображення, номер телефону та вибрати бажане місцезнаходження. Після того, як додаткова інформація буде успішно верифікована в екземплярі Realm, користувач автоматично перенаправляється на головну сторінку для подальшої взаємодії з додатком.

Користувачам буде відображатися вікно з попередженням, що описує помилки, пов'язані з їхніми обліковими даними, у разі порушення таких умов: порожні поля електронної пошти або пароля, неіснуючий обліковий запис або неправильно введені електронна пошта/пароль.

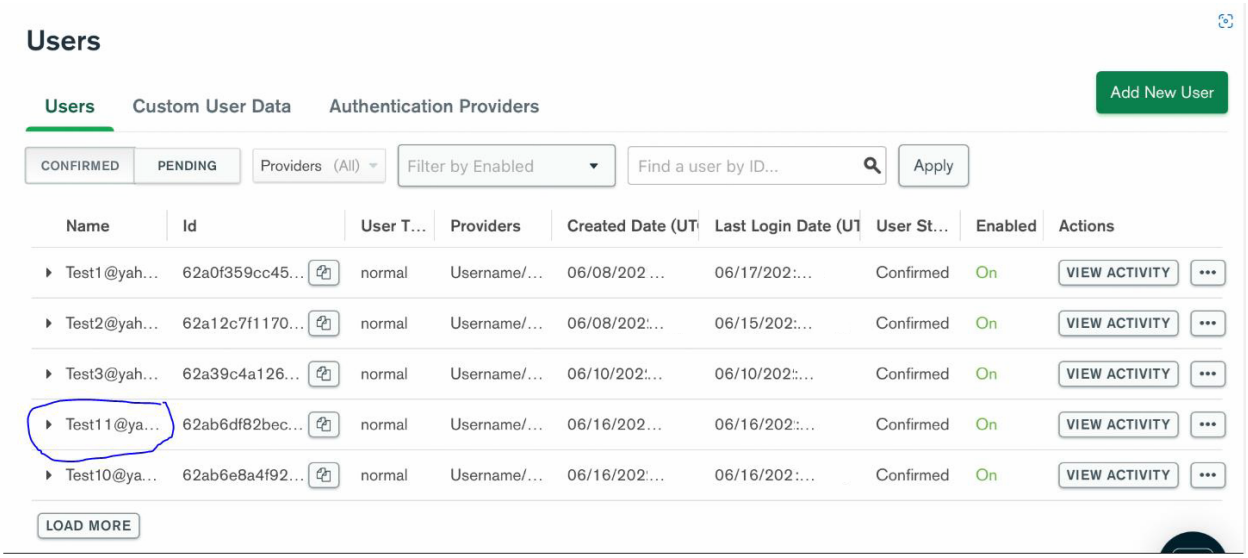

Рисунок 3.3 – Інформація про користувачів в Mongo DB

3.1.2. Сторінка вітання / Налаштування облікового запису (Welcome / Account Setup Page)

Призначена для первинного налаштування облікового запису новими користувачами.

Включає функціонал для завантаження користувацької іконки, введення відображуваного імені та номера телефону.

Після успішного завершення налаштувань користувач перенаправляється на сторінку бажаного місця розташування (рисунок 3.4).

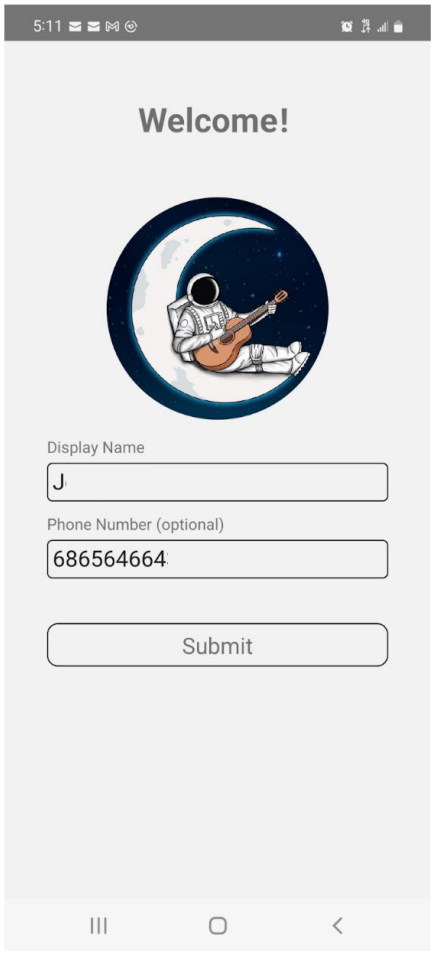

Рисунок 3.4 – Введення додаткової інформації про користувача

3.1.3. Сторінка бажаного місця розташування (Preferred Location Page)

Дана сторінка надає користувачам можливість вказати бажане географічне місце для доступу до їхніх "Дропів". Це місце може бути змінене в будь-який час через сторінку облікового запису (рис. 3.5).

Центральну частину екрану займає інтерактивна карта Google Maps. На карті видно супутникове зображення або гібридний вигляд території, що включає будівлі, дороги та елементи ландшафту. На карті розташована червона позначка-пін (marker), що вказує на точне місцезнаходження, яке користувач вибрав. Це дозволяє візуально підтвердити правильність вибору. Внизу карти видно логотип "Google", що підтверджує використання Google Maps API для реалізації цієї функціональності.

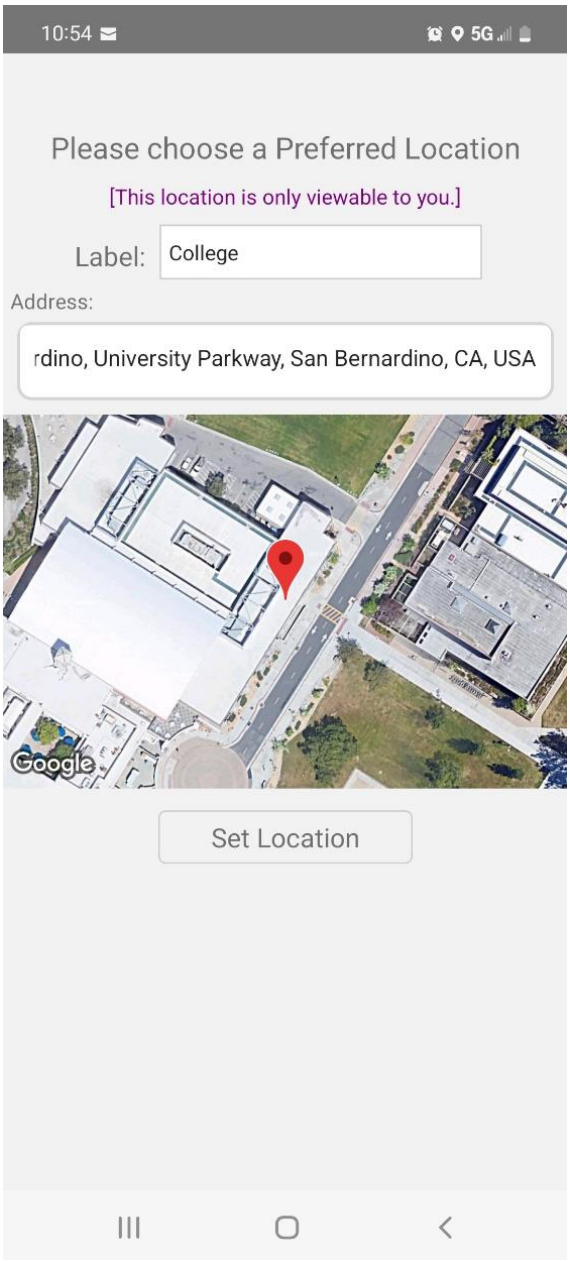

Рисунок 3.5 - Сторінка вибору бажаного місцезнаходження

3.1.4. Опис решти сторінок додатку

Головна сторінка (Home Page) - є центральним елементом навігації, що відображає наявні "Дропи" користувача. Забезпечує прямий доступ до сторінки облікового запису та сторінки друзів.

Включає плаваючу кнопку дії (Floating Action Button - FAB) для швидкого переходу до сторінки створення Drop.

Надає функціонал для переходу на сторінку коментарів для перегляду відгуків до "Дропів".

Сторінка облікового запису (Account Page) - дозволяє користувачам керувати своїм профілем: змінювати фотографію профілю, оновлювати бажане місце розташування та здійснювати вихід з системи. Також містить плаваючу кнопку дії для швидкого переходу на сторінку створення Drop (рис. 3.11).

Сторінка друзів (Friends Page) надає функціонал для управління соціальними зв'язками: перегляд поточних друзів, додавання нових друзів, видалення друзів та вибір конкретного друга для надсилання "Дропу". Включає плаваючу кнопку дії для швидкого переходу на Сторінку створення Drop (рис. 3.4, справа).

Сторінка створення Drop (Create Drop Page) дозволяє користувачам вибирати або змінювати цільових одержувачів для "Дропу" та вказувати місце розташування "Дропу", використовуючи один з трьох доступних варіантів вибору локації.

Сторінка додавання медіа (Add Media Page) забезпечує функціонал для введення текстового повідомлення або додавання медіаконтенту у вигляді зображень або відео до "Дропу" (рис. 3.10).

Сторінка розташування Drop (Drop Location Page) дозволяє користувачам перемикатися між їхнім поточним місцезнаходженням у реальному часі та місцезнаходженням "Дропу" на карті Google. При наближенні користувача до "Дропу" на відстань менше 30 метрів, система автоматично перенаправляє його на головну сторінку для доступу до "Дропу" (рис. 3.13, зліва).

Сторінка коментарів (Comments Page) надає користувачам можливість залишати коментарі до "Дропу" для його відправника або інших користувачів, які переглядають цей самий "Дроп" (рис. 3.14).

Сторінка сповіщень (Notifications Page) дозволяє користувачам переглядати свої системні сповіщення, які генеруються при отриманні нових "Дропів" або запитів на дружбу (рис. 3.12, зліва).

3.2. Специфікація програмних інтерфейсів та функціональних вимог системи

3.2.1. Програмні інтерфейси (API)

Система використовує інтеграцію з кількома зовнішніми програмними інтерфейсами для забезпечення своєї функціональності:

SDK AWS S3 - використовується сторінками "Обліковий запис" та "Медіа" для завантаження зображень профілю користувачів та будь-яких медіафайлів, що прикріплюються до "Дропів".

Після успішного завантаження даних до AWS S3, система повертає унікальне посилання (URL) на збережений контент. Це URL потім передається та зберігається в базі даних MongoDB Atlas.

SDK MongoDB Realm відповідає за зберігання та синхронізацію даних користувачів. Клієнтський додаток на React Native ініціює екземпляр Realm, який забезпечує синхронізацію даних у реальному часі між усіма підключеними клієнтами. Це дозволяє користувачам миттєво отримувати оновлення даних, такі як нові сповіщення або "Дропи" від інших користувачів, що відображає архітектуру, орієнтовану на події.

Google Maps API забезпечує інтеграцію картографічних сервісів.

Використовується клієнтом React Native для відображення географічних місцезнаходжень, встановлення маркерів місця розташування "Дропів", а також для візуалізації 30-метрового діапазону навколо місця "Дропу" на відповідних сторінках інтерфейсу.

3.2.2. Функціональні вимоги

1. Аутентифікація користувачів:

- Система повинна надавати функціонал для аутентифікації користувачів, дозволяючи їм реєструвати облікові записи за допомогою електронної пошти та пароля.

- Після успішної реєстрації користувач повинен мати можливість налаштувати свій обліковий запис, надавши зображення профілю, відображуване ім'я, бажане місце розташування "Дропу" та номер телефону.

- Усі вказані дані повинні зберігатися в унікальних документах користувача у базі даних.

2. Вибір місця розташування Drop:

- Користувачі повинні мати можливість надсилати "Дропи" одному або декільком друзям, що зареєстровані в їхньому списку.

- Після ідентифікації цільового "Дропу", користувач обирає один із трьох варіантів визначення місця розташування "Дропу":

- Відправлення "Дропу" на бажані місця розташування цільових користувачів.

- Введення конкретної адреси.

- Вибір точного місця на карті з можливістю надання власного опису цього місця.

- Додатковою функцією є можливість перегляду останніх місць розташування "Дропів" та додавання улюблених місць до списку "обраних".

3. Додавання медіа до Drop:

- Після визначення місця розташування "Дропу", користувачі повинні мати можливість прикріпити текстове повідомлення та/або медіаконтент.

- Медіа може бути представлене у вигляді зображення або 10-секундного відео, з можливістю зйомки безпосередньо через додаток або вибору файлу зі сховища пристрою.

4. Додавання/видалення друзів:

- Система повинна дозволяти користувачам додавати друзів шляхом введення їхньої електронної пошти або відображуваного імені.

- Після надсилання запиту на дружбу, статус відносин у розділі "Друзі" повинен відображатися як "очікування".

- Одержувач запиту на дружбу повинен мати можливість прийняти або відхилити запит.

- Після встановлення статусу "друзі", обидва користувачі мають право розірвати дружбу.

- Усі зміни у статусі відносин у списку друзів повинні оновлюватися в реальному часі для обох залучених користувачів.

5. Розблокування Drop:

- Для розблокування отриманого "Дропу" користувач повинен фізично знаходитися в межах 30-метрового радіусу від зазначеного місця "Дропу".

- Розблокування надає доступ до вмісту "Дропу" від відправника.

- Користувачі мають можливість залишити "сердечко" (like) або коментар до "Дропу" на відповідній сторінці коментарів.

6. Зміна бажаного місця розташування та зображення профілю:

- Користувачі повинні мати доступ до сторінки "Обліковий запис" для зміни свого бажаного місця розташування "Дропу" та зображення профілю.

7. Обробка сповіщень:

- Сповіщення повинні генеруватися автоматично при отриманні нового "Дропу" або запиту на дружбу.

- Користувач може прийняти або відхилити запит на дружбу виключно на сторінці "Сповіщення".

- Сповіщення про "Дроп" повинні перенаправляти користувача на сторінку місця розташування "Дропу" для навігації до нього.

- Після взаємодії зі сповіщенням (наприклад, перегляду або прийняття/відхилення) воно повинно бути видалене.

3.2.3. Вимоги до продуктивності

Доступність та сумісність: Додаток буде доступний у Google Play і повинен коректно функціонувати на більшості версій операційних систем Android.

Продуктивність бекенду: Бекенд-система повинна демонструвати здатність обробляти велику кількість запитів з низькою затримкою, забезпечувати масштабоване зберігання даних та ефективну обробку даних у межах додатка.

3.2.4. Обмеження дизайну

Розробка буде здійснюватися за допомогою React Native для забезпечення кросплатформної сумісності. Це вимагає ретельного форматування компонентів сторінок, щоб забезпечити консистентний візуальний досвід незалежно від типу пристрою, на якому працює розроблюваний додаток.

3.2.5. Атрибути програмної системи

1. Контроль доступу до "Дропів"

Повідомлення можуть бути отримані лише в тому випадку, якщо зазначений одержувач фізично знаходиться у визначеній зоні "Дропу". Користувачі не повинні бачити вміст "Дропу" до його розблокування.

2. Обмеження надсилання "Дропів".

"Дропи" можуть бути надіслані виключно користувачам, які вже прийняли запит на дружбу. Користувачі не повинні бачити друзів зі статусом "очікування" у випадаючому меню на сторінці створення "Дропу".

3.2.6. Припущення та залежності

Передбачається, що користувач надасть додатку необхідні дозволи для доступу до Інтернету та геолокації, що є критично важливим для надсилання/отримання повідомлень та використання різних функцій системи.

Очікується, що користувачі мають базові знання щодо увімкнення GPS та пошуку бажаних місць на карті.

3.3. Тестові випадки користувача

3.3.1. Тестовий Випадок #1: Аутентифікація користувача

Мета тестового випадку:

Верифікація функціональності системи щодо ефективної обробки процедур створення облікового запису та аутентифікації користувачів, забезпечуючи інтуїтивно зрозумілий та безперебійний користувацький досвід. Зокрема, перевіряється здатність системи розрізняти нових та існуючих користувачів для відповідної навігації по додатку, а також валідація вхідних даних для забезпечення цілісності облікових записів.

Очікувані результати:

1. Система повинна забезпечити всі функціональні цілі, визначені для аутентифікації, з візуально привабливим та практичним відображенням усіх інтерфейсних компонентів.

2. Користувацький інтерфейс повинен бути інтуїтивно зрозумілим, не залишаючи користувачеві сумнівів щодо призначення або функціональності будь-якого візуального елементу.

3. Після успішної аутентифікації екземпляр Realm повинен коректно створити порожній об'єкт користувача в базі даних, готовий для подальшого завантаження профільної інформації.

4. Система повинна запобігати створенню облікових записів з невалідною адресою електронної пошти або порожнім паролем.

5. Система повинна запобігати входу в систему з недійсними обліковими даними.

6. Нові користувачі повинні бути автоматично перенаправлені на сторінку налаштування облікового запису для заповнення додаткової інформації.

7. Існуючі користувачі повинні бути перенаправлені безпосередньо на головний екран.

Процес та фактичний результат:

Початковий стан: При запуску додатка користувач автоматично потрапляє на сторінку входу.

Створення нового облікового запису:

1. Користувач має можливість перейти на сторінку створення облікового запису, як проілюстровано на рисунку 3.1.

2. На цій сторінці користувач вводить бажану адресу електронної пошти та пароль для асоціації з новим обліковим записом.

3. При натисканні кнопки "Створити обліковий запис" клієнтський додаток взаємодіє з MongoDB Realm SDK для ініціалізації нового користувача з наданими обліковими даними та створення відповідного об'єкта користувача в базі даних.

Успішно створені нові користувачі додаються до списку аутентифікованих користувачів (рис. 3.3).

Перенаправлення та завершення налаштування:

1. Після успішного створення екземпляра Realm, клієнтський додаток перенаправляє користувача назад на сторінку входу для авторизації за допомогою щойно створених облікових даних.

2. Після входу, система ідентифікує користувача як нового та автоматично скеровує його на сторінку вітання (Welcome page). На цій сторінці користувач надає додаткову інформацію: ім'я для відображення, номер телефону та вибирає бажане місце розташування.

3. Після верифікації цієї додаткової інформації на екземплярі Realm, користувач остаточно перенаправляється на головну сторінку для продовження взаємодії з функціоналом додатку.

Обробка помилок аутентифікації:

У випадку порушення умов валідації (наприклад, порожні поля електронної пошти/пароля, спроба входу з неіснуючим обліковим записом

або невірними обліковими даними), користувачеві буде представлено вікно сповіщення, що чітко описує причину помилки.

3.3.2. Тестовий Випадок #2: Взаємодія з друзями

Мета тестового випадку:

Верифікація функціональності системи щодо управління соціальними зв'язками, включаючи додавання, видалення та обробку запитів на дружбу. Тест має підтвердити коректну обробку валідації вхідних даних, синхронізацію статусів дружби між користувачами та адекватне відображення цих змін у користувацькому інтерфейсі.

Очікувані результати:

1. Система повинна успішно виконувати операції додавання друзів за допомогою наданої електронної пошти або відображуваного імені.

2. Система повинна своєчасно сповіщати користувача про будь-які помилки, такі як недійсна електронна пошта або відображуване ім'я, що запобігає некоректним діям.

3. Користувачі не повинні мати можливості надсилати запити на дружбу особам, з якими вони вже є друзями, або якщо вказані облікові дані є недійсними.

4. У разі відхилення запиту на дружбу одержувачем, система повинна автоматично оновити список друзів і видалити очікуваний запит зі списку ініціатора.

5. Система повинна дозволяти користувачам видаляти будь-якого друга зі свого списку.

6. Після успішного видалення друга додаток повинен оперативно оновлювати статус дружби для обох залучених користувачів у їхніх відповідних списках друзів.

7. Користувацький інтерфейс повинен оновлюватися в реальному часі, відображаючи всі зміни в статусі дружби.

Процес та фактичний результат:

Ініціація додавання друга:

1. Користувач переходить на сторінку "Друзі" з "Головної сторінки".

2. На сторінці "Друзі" користувач натискає кнопку "Додати друга", розташовану у верхньому правому куті екрана.

3. Після натискання з'являється спливаючий віджет (popup widget), що дозволяє користувачеві ввести електронну пошту або ім'я для відображення цільового друга (рис. 3.6).

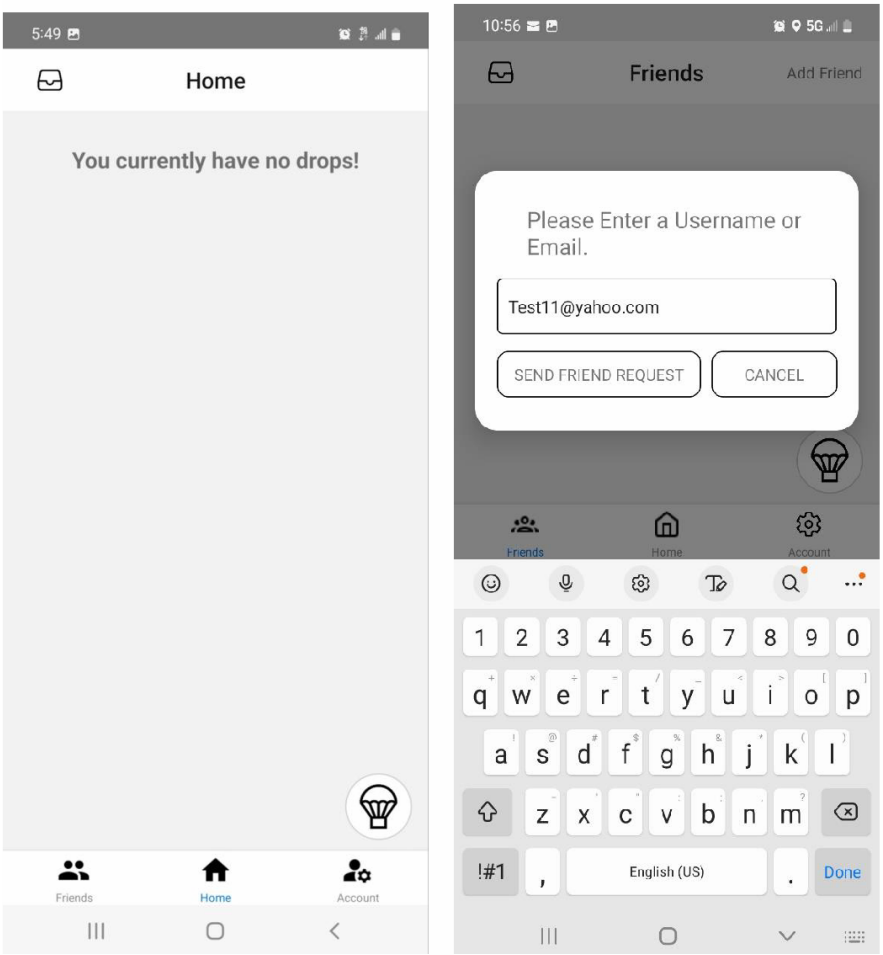

Рисунок 3.6 - Головна сторінка та сторінка друзів

Обробка запиту на додавання:

Успішний пошук: Якщо введений користувач знайдений у системі, спливаючий віджет закривається. Список друзів користувача оновлюється

новим компонентом, що відображає бажане ім'я знайденого користувача зі статусом "очікування".

Невдалий пошук: Якщо користувач не знайдений або введені дані недійсні, спливаючий віджет відображає повідомлення про помилку червоним текстом, вказуючи на невідповідність запиту та надаючи користувачеві можливість повторити спробу.

Оновлення статусу дружби:

1. Після того, як одержувач приймає запит на дружбу, система негайно оновлює статус дружби для обох користувачів.

2. Зміни відображаються на їхніх відповідних сторінках "Друзі" (рис. 3.7), відображаючи активний статус дружби.

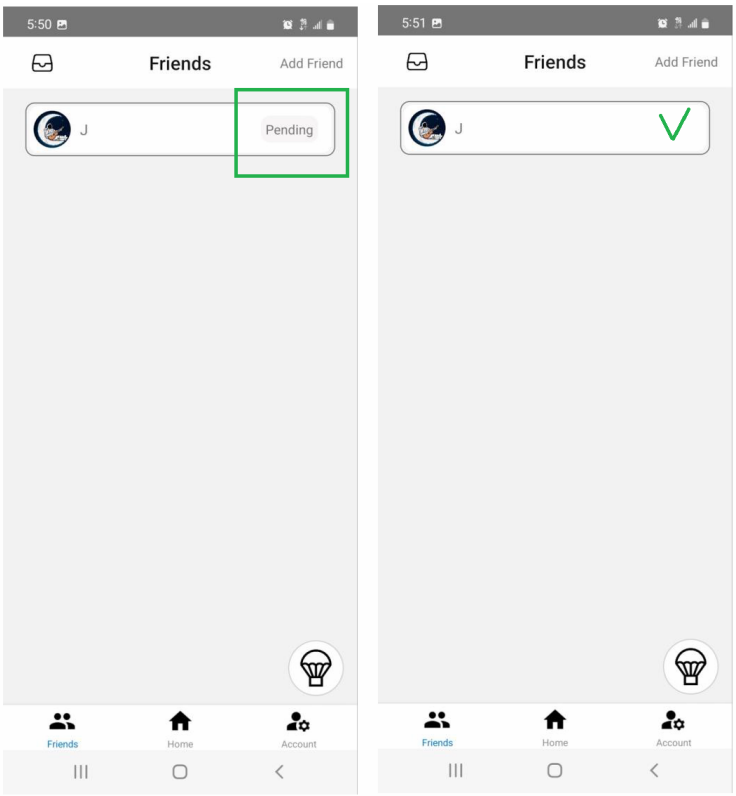

Рисунок 3.7 - Зміна статусу друга

Видалення друга:

1. Користувач має можливість видалити друга, натиснувши на відповідний компонент друга у списку.

2. З'являється спливаючий віджет з опцією "Видалити з друзів" (рис. 3.8).

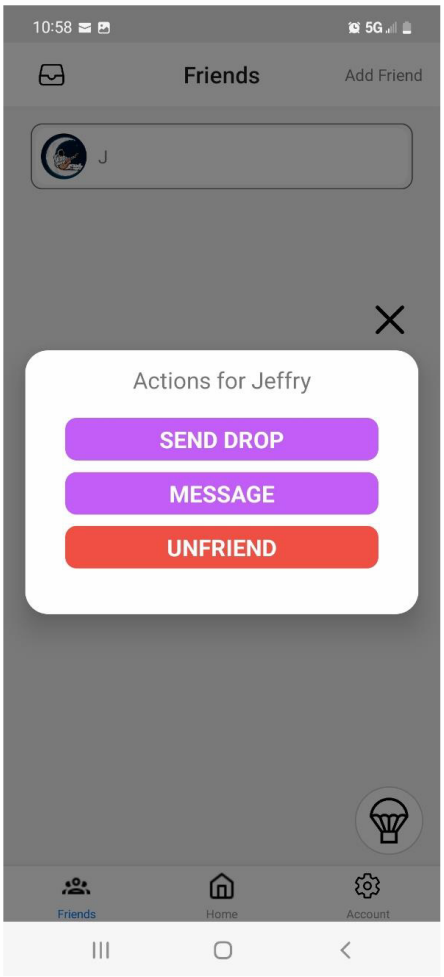

Рисунок 3.8 - Дії для користувача

3. При виборі цієї опції відбувається взаємне видалення статусу дружби між обома користувачами в системі.

4. Щоб відновити дружбу, користувачам необхідно буде ініціювати новий запит на дружбу.

3.3.3. Тестовий Випадок #3: Надсилання Drop користувачам

Мета тестового випадку:

Верифікація комплексного функціоналу надсилання "Дропів" (Drops) у системі, включаючи вибір одержувачів, визначення місцезнаходження

"Дропу", додавання медіаконтенту та обробку асинхронних операцій збереження даних. Також перевіряється коректність сповіщень та розблокування "Дропів" для одержувачів.

Очікувані результати:

1. Система повинна забезпечити можливість надсилання соціальних медіа "Дропів" одному або кільком обраним друзям.

2. Користувачі повинні мати можливість вибору одного з трьох визначених варіантів для вказівки місця розташування "Дропу": бажане місце розташування цільового одержувача, конкретна адреса або довільне місце з користувацьким описом.

3. Система повинна надавати функціонал для повторного використання "останнього" або "улюбленого" місця розташування "Дропу".

4. Система повинна запобігати переходу до наступного етапу створення "Дропу" без попереднього вибору одержувачів та/або визначення місця розташування.

5. Користувачеві повинна бути надана можливість включення текстового повідомлення та/або прикріплення зображення чи 10-секундного відео до "Дропу".

6. Клієнтський додаток не повинен завершувати процес надсилання "Дропу" до тих пір, поки весь медіаконтент не буде успішно збережений у базі даних та доступний для подальшого отримання.

7. Додаток повинен відповідати всім вищезазначеним вимогам, підтримуючи візуально привабливий та інтуїтивно зрозумілий користувацький інтерфейс.

У разі невиконання будь-яких обов'язкових умов система повинна негайно повідомляти користувача про необхідність коригування.

Всі візуальні компоненти повинні бути легко ідентифікованими з чітко визначеною функціональністю.

Процедура та фактичні результати:

1. Вибір одержувача(ів) "Дропу":

Варіант 1 (через сторінку "Друзі"): користувач переходить на сторінку "Друзі", обирає віджет друга та активує опцію "Надіслати Drop" (рис. 3.8). Це автоматично перенаправляє користувача на сторінку "Створити Drop" з попередньо вибраним одержувачем.

Варіант 2 (через плаваючу кнопку): користувач натискає плаваючу кнопку "+" на будь-якій основній сторінці (Головна, Друзі, Обліковий запис). Це перенаправляє його на сторінку "Створити Drop", де він може натиснути додаткову кнопку "+" для відкриття спливаючого віджета зі списком друзів та вибору одержувачів (рис. 3.9).

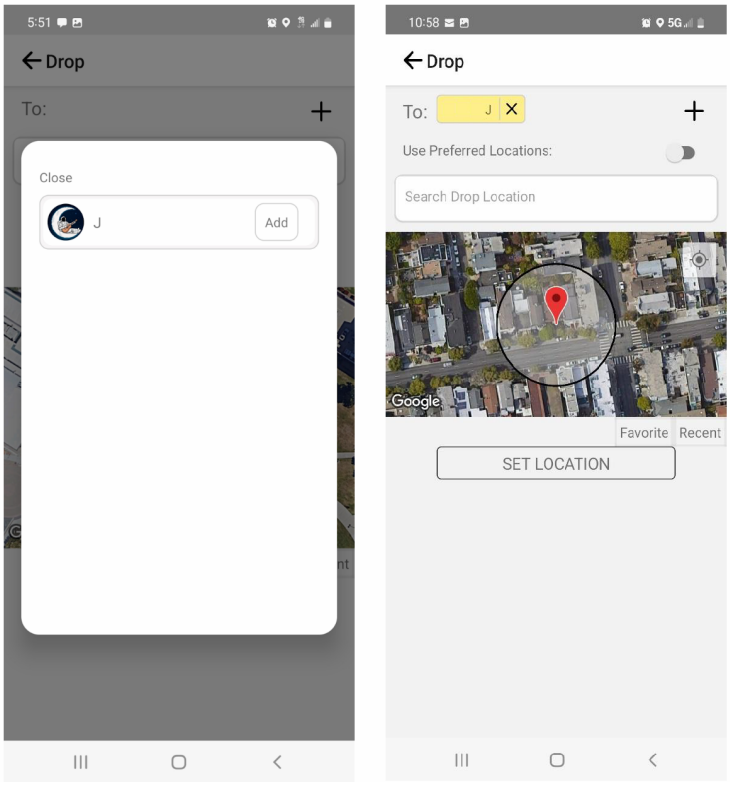

Рисунок 3.9 - Сторінка створення Drop

2. Визначення місця розташування "Дропу":

- Після вибору одержувача(ів) користувач може обрати один з трьох варіантів визначення місця розташування "Дропу":

- Бажане місце розташування одержувачів: Активація кнопки перемикача дозволяє автоматично встановити місцезнаходження "Дропу" на основі бажаних локацій одержувачів. При цьому інші опції вибору місцезнаходження візуально деактивуються до вимкнення перемикача.

- Введення конкретної адреси: Користувач може ввести точну адресу, що спричинить оновлення інтегрованої карти Google Maps для відображення обраного місцезнаходження.

- Вибір на карті з описом: Користувач може вручну вказати будь-яке місце на картах Google та надати довільний опис для цієї зони "Дропу".

- Додаткові опції: Якщо користувач раніше надсилав "Дропи", йому надається можливість повторно використати "останнє" або "улюблене" місце розташування "Дропу".

3. Додавання медіа та повідомлення:

- Після підтвердження місця розташування (натискання кнопки "Встановити місце розташування", рис. 3.9, справа) користувач перенаправляється на сторінку "Додати медіа".

- Користувач зобов'язаний ввести текстове повідомлення. Система повідомить користувача, якщо поле повідомлення буде порожнім.

- Користувач може додатково прикріпити медіа у вигляді зображення або 10-секундного відео, з можливістю зйомки безпосередньо через додаток або вибору з галереї пристрою. Рисунок 3.10 демонструє адаптацію інтерфейсу при додаванні медіа.

- Медіа-контент може бути видалений натисканням кнопки "X" поруч з відповідним контейнером.

4. Надсилання "Дропу":

- Після остаточного формування "Дропу" користувач натискає кнопку "Надіслати Drop".

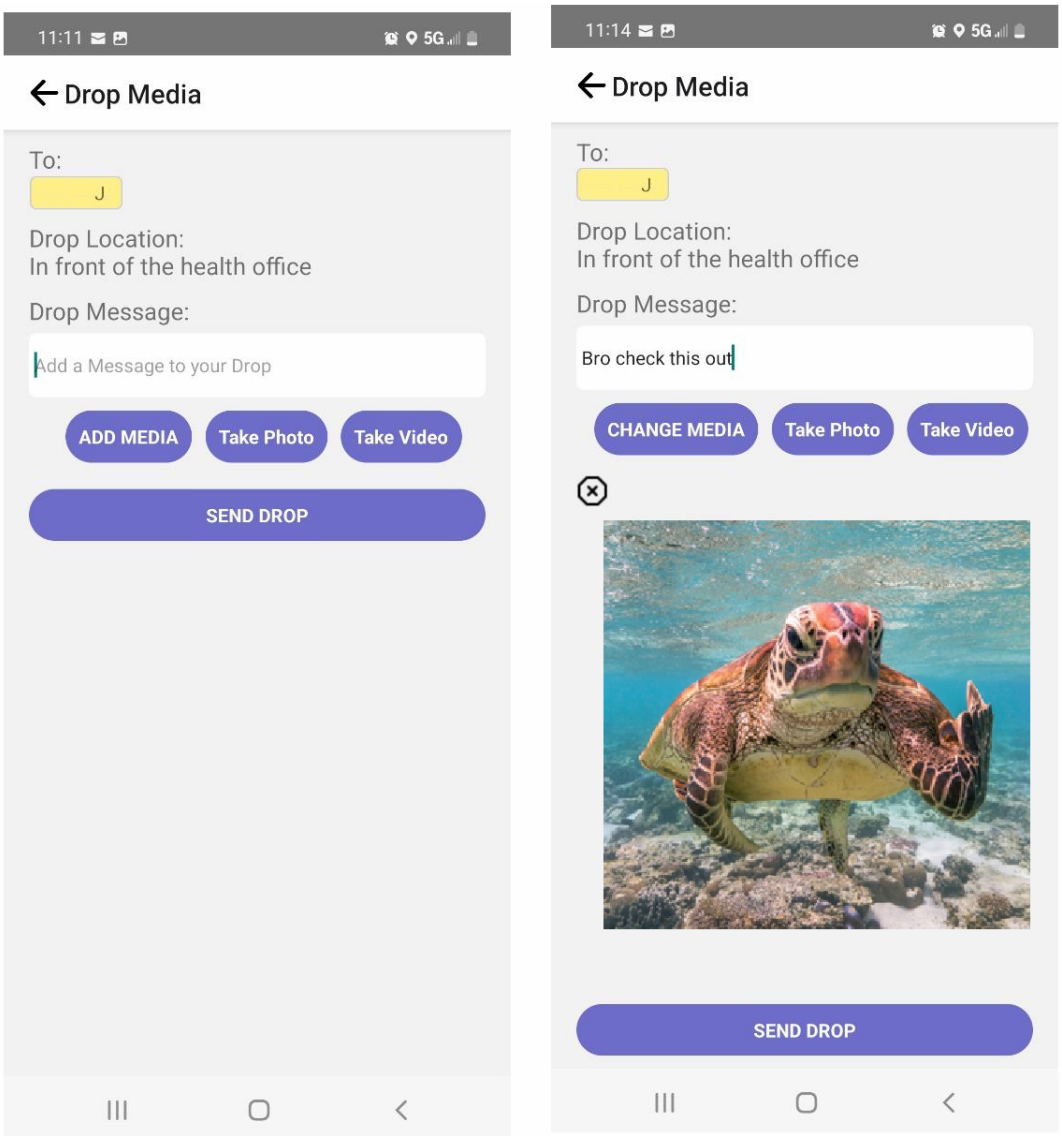

Рисунок 3.10 - Сторінка медіа Drop після надання користувальницького місця розташування

- Залежно від розміру медіафайлу, може відображатися екран завантаження, що інформує про процес завантаження контенту на бекенд.

Після успішного завершення завантаження користувач автоматично перенаправляється на головну сторінку, де його новостворений "Дроп" буде відображений (рис. 3.11).

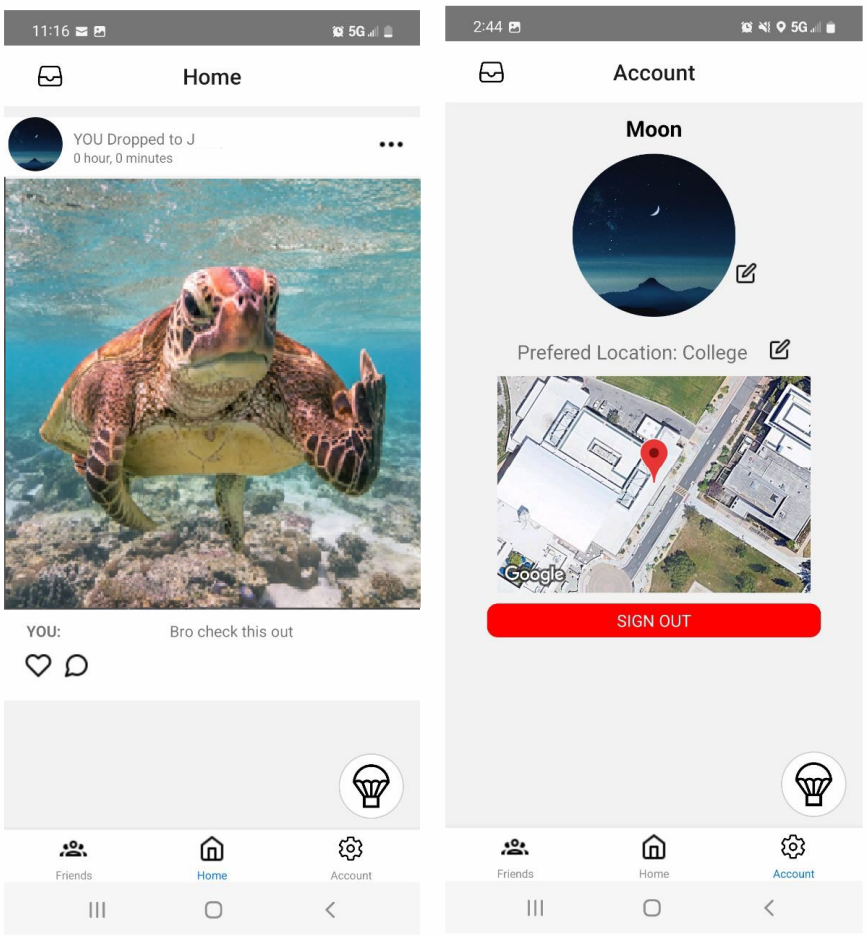

Рисунок 3.11 - Головна сторінка після надсилання Drop користувачу

5. Отримання та розблокування "Дропу" одержувачем:

- Після надсилання "Дропу" одержувач негайно отримує сповіщення.

Одержувач може перейти на сторінку сповіщень через локальне сповіщення телефону (див. рис. 3.12, зліва) або переглянути "Дроп" безпосередньо з головної сторінки (див. рис. 3.12, справа).

Незалежно від методу, одержувач буде перенаправлений на сторінку місця розташування Drop (див. рис. 3.13, зліва), де він може візуально відстежувати своє поточне місцезнаходження відносно "Дропу" на карті.

Як тільки користувач потрапляє в 30-метровий діапазон від місця розташування "Дропу" (що індикується графічним компонентом "зони Дропу"), система автоматично перенаправляє його назад на головну сторінку з розблокованим "Дропом" (див. рис. 3.13, справа), надаючи доступ до його вмісту.

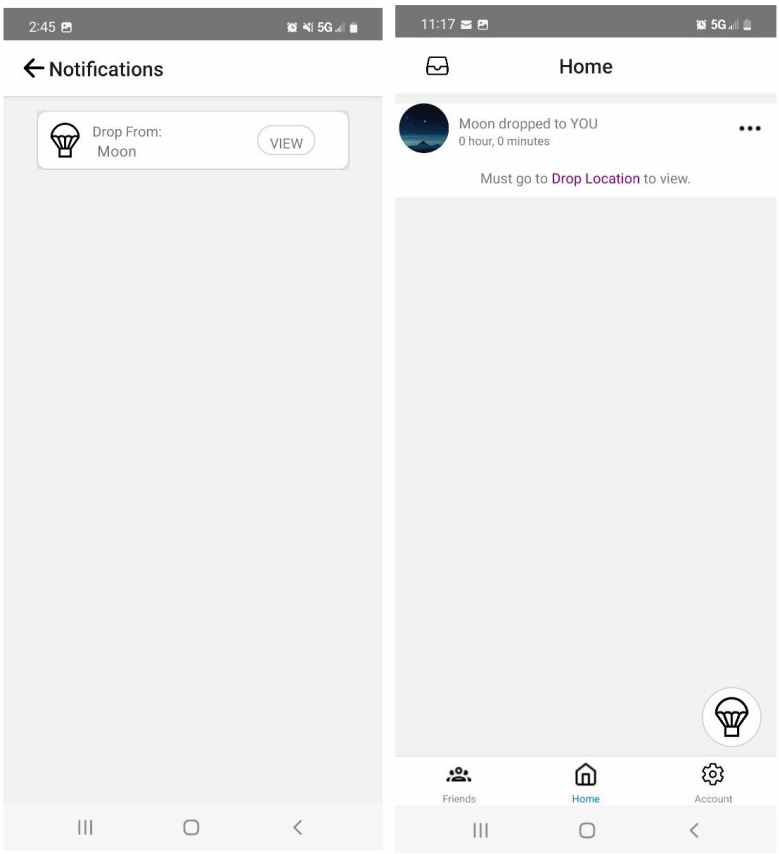

Рисунок 3.12 - Сторінка сповіщень та головна (заблокована) сторінка

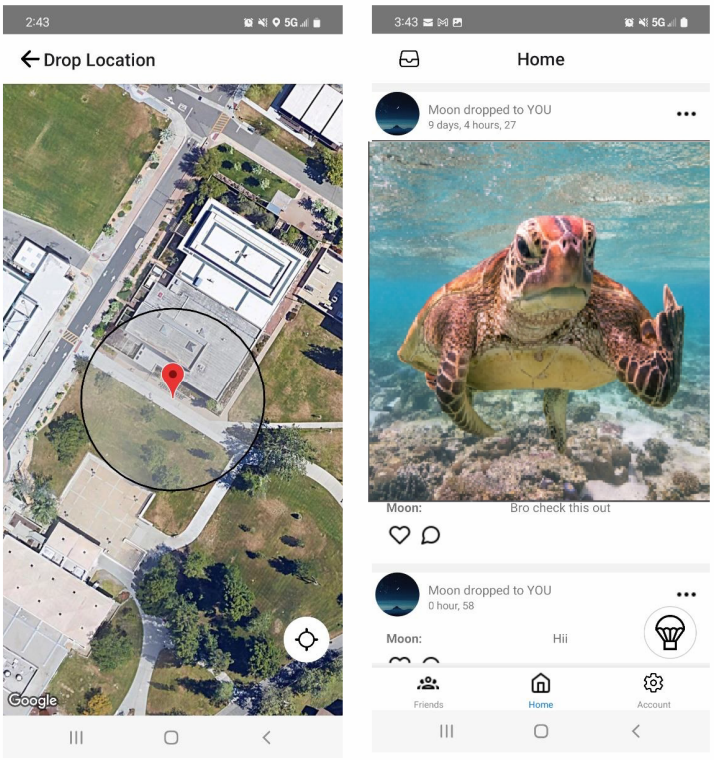

Рисунок 3.13 - Сторінка розташування Drop та головна (розблокована) сторінка

6. Взаємодія зі зворотним зв'язком:

Як відправник, так і одержувач "Дропу" мають можливість залишити зворотний зв'язок (наприклад, "сердечко" або коментар) на відповідній сторінці коментарів (рис. 3.14).

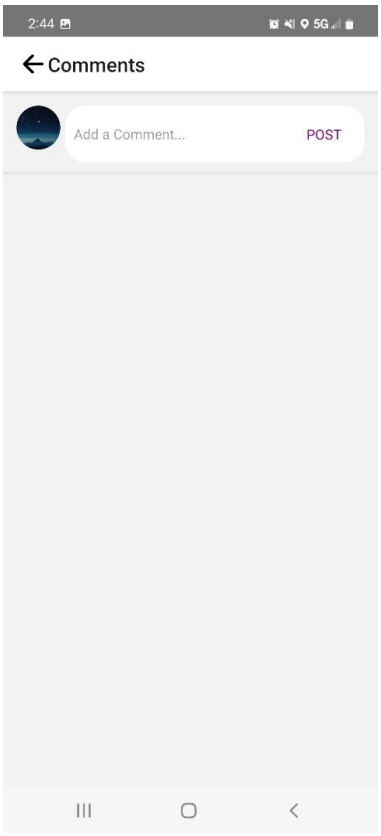

Рисунок 3.14 - Сторінка коментарів

Отже, в розділі описано архітектуру та функціонал системи, зосереджуючись на взаємодії користувацького інтерфейсу, інтеграції з хмарними сервісами та детальних тестових випадках.

Архітектурно, додаток реалізовано за принципом мікросервісів, що забезпечує гнучкість та масштабованість. Клієнтський додаток на React Native взаємодіє з MongoDB Realm для управління даними користувачів та синхронізації в реальному часі, AWS S3 для зберігання медіаконтенту, а також Google Maps API для функцій геолокації та відображення карт.

Функціонал охоплює повний життєвий цикл користувача: від реєстрації та налаштування облікового запису (включаючи візуально привабливі сторінки входу, створення облікового запису та налаштування профілю з вибором бажаного місцезнаходження), до складних соціальних взаємодій. Ключовою особливістю є можливість надсилання "Дропів" — географічно прив'язаних повідомлень з медіаконтентом, які можуть бути розблоковані лише при фізичному наближенні до зазначеної локації. Додаток підтримує керування друзями, систему сповіщень та механізми зворотного зв'язку.

Вимоги до якості акцентують на забезпеченні надійного користувацького досвіду, високої продуктивності бекенду з низькою затримкою та ефективною обробкою даних. Застосування React Native як кросплатформного фреймворку підкреслює прагнення до уніфікованого візуального стилю.

Тестові випадки деталізують сценарії використання, підтверджуючи коректність аутентифікації користувачів, управління друзями та ключовий функціонал надсилання/отримання "Дропів". Особлива увага приділяється валідації вхідних даних та оперативному інформуванню користувача про помилки, а також синхронізації даних у реальному часі.

Загалом, розроблюваний додаток представляє собою інноваційну соціальну платформу з унікальною концепцією локаційно-залежного обміну контентом, що спирається на сучасну мікросервісну архітектуру та забезпечує високий рівень взаємодії.

ВИСНОВКИ

У процесі виконання дипломної роботи на тему “Реалізація проєкту соціального програмного забезпечення” було комплексно досліджено, спроєктовано та реалізовано програмну систему для соціальної взаємодії користувачів з можливістю обміну контентом. Усі етапи роботи – від аналізу предметної області до практичного впровадження архітектурних рішень – були послідовно виконані відповідно до методологічних засад інженерії програмного забезпечення.

У першому розділі було проведено глибокий аналіз предметної області та сучасного стану соціального програмного забезпечення. Особливу увагу приділено функціональним можливостям систем обміну контентом, принципам їх роботи, вимогам до безпеки, масштабованості, сумісності та інтеграції з хмарними сервісами. Розглянуто парадигми взаємодії користувачів у соціальних мережах, а також обґрунтовано актуальність і значущість створюваного програмного продукту. Також проаналізовано децентралізовані підходи до поширення контенту, що дозволило сформувати сучасне бачення архітектури системи.

У другому розділі детально описано процес побудови архітектури програмного забезпечення, в основі якої лежить принцип мікросервісної взаємодії. Було розроблено карту потоків користувацького інтерфейсу, визначено цільову аудиторію та системні обмеження, що враховано при моделюванні вимог до системи. Застосовано нотації UML для створення діаграм класів та варіантів використання, що дало змогу формалізувати функціональні та нефункціональні вимоги до майбутнього застосунку. Специфічні вимоги реалізації враховували особливості безпечної та ефективної взаємодії користувачів із системою.

У третьому розділі представлено програмну реалізацію системи. Було розроблено прототип інтерфейсу користувача з урахуванням принципів

зручності, доступності та інтуїтивної взаємодії. Реалізовано ключові елементи системи, зокрема механізми аутентифікації, налаштування профілю, обрання переваг користувача, взаємодії з друзями та обміну контентом (Drop-систему). Розроблено специфікацію програмних інтерфейсів (API), визначено атрибути програмної системи, обмеження дизайну та продуктивності. Проведено тестування основних сценаріїв використання системи, що підтвердило працездатність реалізованих функцій.

Узагальнюючи результати роботи, можна зробити висновок, що розроблена система відповідає поставленим вимогам щодо створення сучасного, масштабованого, безпечного і функціонального соціального програмного забезпечення. Вона має потенціал для подальшого розвитку, зокрема шляхом інтеграції алгоритмів персоналізації контенту, розширення можливостей для групової взаємодії, а також впровадження технологій штучного інтелекту для аналізу поведінки користувачів.

Таким чином, поставлену мету досягнуто: реалізовано працездатний прототип програмного продукту, що поєднує функції соціальної мережі з механізмами децентралізованого обміну контентом, відповідає сучасним тенденціям у галузі та має практичну цінність для кінцевих користувачів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Davies, T., & Mintz, M. D. (2013). Design Features for the Social Web: The Architecture of Deme. arXiv preprint arXiv:1302.4765. https://arxiv.org/abs/1302.4765
2. Montesi, F., & Weber, J. (2016). Circuit Breakers, Discovery, and API Gateways in Microservices. arXiv preprint arXiv:1609.05830. https://arxiv.org/abs/1609.05830
3. Zhong, C., & Sastry, N. (2017). Systems Applications of Social Networks. arXiv preprint arXiv:1707.05104.
4. Jahromi, N. T., Glitho, R. H., Larabi, A., & Brunner, R. (2017). An NFV and Microservice Based Architecture for On-the-fly Component Provisioning in Content Delivery Networks. arXiv preprint arXiv:1710.04991.
5. Iber, J., Rauter, T., & Kreiner, C. (2018). Microservice Architecture for the Industrial Internet-Of-Things. Proceedings of the 23rd European Conference on Pattern Languages of Programs. https://dl.acm.org/doi/10.1145/3282308.3282320
6. Kumar, S., & Goyal, S. K. (2022). Social Cloud Computing: Architecture and Application. In N. Marriwala et al. (Eds.), Emergent Converging Technologies and Biomedical Systems (pp. 13–24). Springer, Singapore. DOI: https://doi.org/10.1007/978-981-16-8774-7_2
7. Chen, L. (2018). Microservices: Architecting for Continuous Delivery and DevOps. IEEE Software, 35(5), 50–55. DOI: https://doi.org/10.1109/MS.2018.2901010 rd.springer.com
8. 8. Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., & Mustafin, R. (2017).

9. Microservices: Yesterday, Today, and Tomorrow. In Present and Ulterior Software Engineering (pp. 195–216). Springer. DOI: https://doi.org/10.1007/978-3-319-67425-4_12
10. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media. ISBN: 978-1-4919-0411-9
11. Wolff, E. (2016). Microservices: Flexible Software Architecture. Addison-Wesley Professional. ISBN: 978-0-13-460241-7
12. Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. arXiv preprint arXiv:1606.04036. https://arxiv.org/abs/1606.04036
13. Hannousse, A., & Yahiouche, S. (2020). Securing microservices and microservice architectures: A systematic mapping study. arXiv preprint arXiv:2003.07262. https://arxiv.org/abs/2003.07262
14. Laigner, R., Zhou, Y., Vaz Salles, M. A., Liu, Y., & Kalinowski, M. (2021). Data management in microservices: State of the practice, challenges, and research directions. arXiv preprint arXiv:2103.00170. https://arxiv.org/abs/2103.00170
15. Hoque, S., & Miranskyy, A. (2018). Architecture for analysis of streaming data. arXiv preprint arXiv:1805.01025. https://arxiv.org/abs/1805.01025
16. Razzaq, A. (2020). A systematic review on software architectures for IoT systems and future direction to the adoption of microservices architecture. SN Computer Science, 1(6), 1–30. https://doi.org/10.1007/s42979-020-00359-w
17. Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., & Mustafin, R. (2017). Microservices: Yesterday, today, and tomorrow. In Present and Ulterior Software Engineering (pp. 195–216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12

18. Chen, L. (2018). Microservices: Architecting for continuous delivery and DevOps. IEEE Software, 35(5), 50–55. https://doi.org/10.1109/MS.2018.2901010
19. Newman, S. (2015). Building microservices: Designing fine-grained systems. O'Reilly Media.
20. Montesi, F., & Weber, J. (2016). Circuit breakers, discovery, and API gateways in microservices. arXiv preprint arXiv:1609.05830. https://arxiv.org/abs/1609.05830
21. Iber, J., Rauter, T., & Kreiner, C. (2018). A microservice architecture for the industrial internet-of-things. In Proceedings of the 23rd European Conference on Pattern Languages of Programs. https://dl.acm.org/doi/10.1145/3282308.3282320
22. Kumar, S., & Goyal, S. K. (2022). Social cloud computing: Architecture and application. In N. Marriwala et al. (Eds.), Emergent Converging Technologies and Biomedical Systems (pp. 13–24). Springer, Singapore. https://doi.org/10.1007/978-981-16-8774-7_2
23. Jahromi, N. T., Glitho, R. H., Larabi, A., & Brunner, R. (2017). An NFV and microservice based architecture for on-the-fly component provisioning in content delivery networks. arXiv preprint arXiv:1710.04991. https://arxiv.org/abs/1710.04991
24. Zhong, C., & Sastry, N. (2017). Systems applications of social networks. arXiv preprint arXiv:1707.05104. https://arxiv.org/abs/1707.05104
25. Davies, T., & Mintz, M. D. (2013). Design features for the social web: The architecture of Deme. arXiv preprint arXiv:1302.4765. https://arxiv.org/abs/1302.4765
26. Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Software, 33(3), 42–52. https://doi.org/10.1109/MS.2016.64

27. Gannon, D., Barga, R., & Sundaresan, N. (2017). Cloud-native applications. IEEE Cloud Computing, 4(5), 16–21. https://doi.org/10.1109/MCC.2017.4250932
28. Fowler, M., & Lewis, J. (2014). Microservices: A definition of this new architectural term. martinfowler.com. https://martinfowler.com/articles/microservices.html
29. Pautasso, C., Zimmermann, O., & Leymann, F. (2017). RESTful web services vs. “big” web services: Making the right architectural decision. In Proceedings of the 17th International Conference on World Wide Web (pp. 805–814). https://doi.org/10.1145/1367497.1367606
30. Thönes, J. (2015). Microservices. IEEE Software, 32(1), 116–116. https://doi.org/10.1109/MS.2015.11

ДОДАТКИ

Додаток А

Фрагмент коду створення сторінки надсилання дропів

```
import { useState } from 'react';
const CameraIcon = () => (
  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" className="w-4 h-4">
    <path d="M14.5 4H5.5a1 1 0 0 0-1 1v14a1 1 0 0 0 1 1h13a1 1 0 0 0 1-1V5a1 1 0 0 0-1-1H9.5" />
    <circle cx="12" cy="12" r="3" />
  </svg>
);

const CloseIcon = () => (
  <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" className="w-4 h-4">
    <path d="M18 6L6 18M6 6l12 12" />
  </svg>
);

const DropMedia = () => {
  const [media, setMedia] = useState<File | null>(null);
  const [message, setMessage] = useState('');

  const handleMediaChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    if (event.target.files && event.target.files.length > 0) {
      setMedia(event.target.files[0]);
    }
  };
```

```
       return (
         <div  className="w-full  max-w-sm  mx-auto  p-4  bg-white
rounded-lg shadow-md">
           <div className="flex items-center justify-between mb-
4">
             <span   className="text-lg   font-bold">&larr;   Drop
Media</span>
           </div>

           <form>
             <div className="mb-4">
               <label  className="block  text-sm  font-medium  mb-
1">To:</label>
               <input  type="text"  className="w-full  p-2  border
border-gray-300 rounded" />
             </div>

             <div className="mb-4">
               <label  className="block  text-sm  font-medium  mb-
1">Drop Location:</label>
               <input type="text" value="In front of the health
office" className="w-full p-2 border border-gray-300 rounded bg-
gray-100" readOnly />
             </div>

             <div className="mb-4">
               <label  className="block  text-sm  font-medium  mb-
1">Drop Message:</label>
               <textarea   value={message}   onChange={(e)   =>
setMessage(e.target.value)} className="w-full p-2 border border-
gray-300 rounded" rows={3} />
             </div>

             {media ? (
               <div className="relative mb-4">
```

```
                <img            src={URL.createObjectURL(media)}
alt="Selected   Media"   className="w-full   h-48   object-cover
rounded" />
                <button     type="button"     onClick={()     =>
setMedia(null)}   className="absolute   top-2   right-2   bg-white
rounded-full p-1">
                  <CloseIcon />
                </button>
              </div>
            ) : (
              <div className="flex space-x-2 mb-4">
                <input        type="file"        accept="image/*"
onChange={handleMediaChange}    className="hidden"    id="media-
upload" />
                <label  htmlFor="media-upload"  className="flex
items-center  justify-center  px-4  py-2  border  border-gray-300
rounded cursor-pointer">
                  <CameraIcon />
                  <span       className="ml-2       text-sm">ADD
MEDIA</span>
                </label>
                <button   type="button"   className="px-4   py-2
border border-gray-300 rounded">Take Photo</button>
                <button   type="button"   className="px-4   py-2
border border-gray-300 rounded">Take Video</button>
              </div>
            )}

            <button  type="submit"  className="w-full  py-2  bg-
purple-500 text-white rounded">SEND DROP</button>
          </form>
        </div>
      );
    };
    export default DropMedia;
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Реалізація проекту соціального програмного забезпечення ”

Обсяг пояснювальної записки: 77 аркушів.

Дата закінчення роботи: 9 червня 2025 р.

Підпис студента _____