

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки
Факультет комп'ютерних наук та технологій
(повне найменування інституту, факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)
бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ НА БАЗІ
ОС ANDROID ДЛЯ ВУЛИЧНИХ ГОНЩИКІВ

Виконав: студент 4 курсу, групи КНТ-519

Спеціальності 123 «Комп'ютерна інженерія»
(код і найменування спеціальності)

Освітня програма (спеціалізація) «Комп'ютерна інженерія»

Слонєв М.А.
(прізвище та ініціали)

Керівник Скрупський С.Ю.
(прізвище та ініціали)

Рецензент Самойлик С.С.
(прізвище та ініціали)

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

Інститут, факультет інформатики та радіоелектроніки, комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти (освітній ступінь) бакалаврський

Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри Кудерметов Р.К.

“17” листопада 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА

Слонєва Максима Андрійовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи): Розробка комп'ютерної системи на базі ОС Android для вуличних гонщиків

керівник проекту (роботи) Скрупський Степан Юрійович, к. т. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “24” березня 2023 року № 71

2. Строк подання студентом проекту (роботи) 10 травня 2023 року

3. Вихідні дані до проекту (роботи) Технології створення Android-додатків для роботи з мапою Java, Kotlin, RxJava, Kotlin Coroutines, MVVM, XML, Koin, Firebase, PHP, Node JS . Необхідно обрати найкращі для побудови розподіленого додатку з можливістю легкого масштабування в майбутньому

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

1) Аналіз існуючих рішень і технологій;

2) Розробка клієнтської та серверної частин системи;

3) Випробування розробленої системи.

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

Схема розподіленої системи

Схема архітектури Android додатку

Схема авторизації та взаємодії з мапою

Схема використання соціальних функцій додатку

Схема розробленої бази даних

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	Прийняв виконане завдання
1-3	Скрупський С.Ю., к. т. н., доцент		
Нормконтроль	Щербак Н.В., ст. викл.		

7. Дата видачі завдання 23.03.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз технологій для побудови клієнтського і серверного програмного забезпечення для роботи з базою даних, перегляд отриманих результатів, виділення недоліків та переваг	25.03.2023 р.	
2	Розробка архітектури системи	30.03.2023 р.	
3	Розробка алгоритму роботи системи	10.04.2023 р.	
4	Програмно-апаратна реалізація системи	15.04.2023 р.	
5	Випробування системи	10.05.2023 р.	
6	Оформлення графічного матеріалу	12.05.2023 р.	
7	Оформлення отриманих результатів у ПЗ	15.05.2023 р.	

Студент


(підпис)

М.А.Слонев
(ініціали та прізвище)

Керівник проекту (роботи)


(підпис)

С.Ю.Скрупський
(ініціали та прізвище)

РЕФЕРАТ

ПЗ: 98 с; 20 рис; 19 джерел; 1 додаток.

ANDROID, FIREBASE, KOTLIN, MVVM, АРХІТЕКТУРА, БАЗА ДАНИХ, ДОДАТОК, КЛІЄНТ, МАПА, НАВІГАЦІЯ, ПАТТЕРНИ, СЕРВЕР, СИСТЕМА, СТРУКТУРА.

Об'єктом дослідження є розподілена комп'ютерна система на базі операційної системи Android з функціями навігатора.

Предметом дослідження є методи і засоби для проектування архітектури і побудови розподіленого додатку з можливістю легкого масштабування в майбутньому.

Метою даної дипломної роботи є проектування і створення розподіленої системи для байкерів, яка дозволить створювати та організовувати тематичні заходи, відмічати їх на карті, а також полегшити спілкування серед байкерської спільноти за допомогою мобільних пристроїв.

Для створення системи були обрана мова програмування Kotlin, технології Firebase, Kotlin Coroutines, архітектурний паттерн MVVM та бібліотека Koin для впровадження залежностей.

Робота складається з трьох розділів.

Перший розділ присвячено аналізу технологій, які застосовуються при побудові системи даної роботи, а саме серверної та клієнтської частин додатку, проведенню їх порівняння, в результаті чого зроблено висновки.

У другому розділі визначено вимоги до системи, описано процес створення серверного та клієнтського додатків, застосовуючи обрані технології з прикладами коду.

Третій розділ присвячено тестуванню системи з прикладами та діаграмами сценаріїв використання.

ЗМІСТ

Список скорочень.....	6
Вступ.....	7
1 Аналіз існуючих рішень і технологій	10
1.1 Актуальність розробки під ОС Android	10
1.2 Огляд існуючих рішень для спілкування у мережі та роботі з мапою	11
1.3 Аналіз існуючих рішень побудови архітектури розподіленої системи	13
1.4 Аналіз технологій для клієнтської частини системи.....	15
1.5 Аналіз технологій для серверної частини системи.....	22
1.6 Висновки щодо аналізу технологій	25
2 Розробка клієнтської та серверної частин системи	28
2.1 Етапи розробки розподіленого додатку	28
2.2 Визначення технічних вимог до системи	29
2.3 Визначення програмних вимог до системи	32
2.4 Моделювання об'єктів системи	36
2.5 Опис архітектури системи.....	38
2.6 Огляд бібліотеки для роботи з мапою.....	43
2.7 Опис структури системи.....	44
2.8 Розробка модулів клієнтської частини	52
2.9 Розробка бази даних додатку	70
2.10 Висновки щодо розробки розподіленої системи	73
3 Випробування розробленої системи	74
3.1 Опис сценаріїв роботи з додатком	74
3.2 Демонстрація виконання описаних сценаріїв роботи з додатком	79
3.3 Висновки щодо випробування розподіленої системи	87
Висновки	88
Перелік джерел посилання	89
Додаток А.....	91

СПИСОК СКОРОЧЕНЬ

ПЗ	– програмне забезпечення
ОС	– операційна система
UI	– User Interface, користувацький інтерфейс
БД	– база даних
СУБД	– система управління базами даних
IDE	– Integrated Development Environment, інтегроване середовище розробки
UML	– Unified Modeling Language, уніфікована мова моделювання
API	– Application Programming Interface, програмний інтерфейс додатку
XML	– Extensible Markup Language, розширювана мова розмітки
HTML	– HyperText Markup Language, мова розмітки гіпертексту
HTTP	– Hypertext Transfer Protocol, протокол передачі гіпертексту
JSON	– JavaScript Object Notation, текстовий формат обміну даними на основі JavaScript

ВСТУП

Вуличні гонки завжди були дуже популярними серед молоді. Ці змагання привертають багато уваги, і їх проводять по всьому світу. Хоча в уявленні більшості людей вони зазвичай пов'язані з автомобільним спортом, їх невід'ємною частиною завжди були байкери. Багато з них займаються вуличними гонками на своїх мотоциклах та приймають участь у різних змаганнях на вулицях міст. Нажаль, умови воєнного стану призводять до зміни звичайного способу життя та розваг, і вуличні гонки не є винятком. В умовах конфлікту, коли безпека є найважливішою проблемою, проведення вуличних гонок стає надзвичайно важкою задачею. Багато трас та місць для проведення змагань можуть бути небезпечними, а також потребувати великих витрат на організацію та забезпечення безпеки учасників та глядачів.

У таких умовах майже єдина спільнота вуличних гонщиків, яка залишається активною, є байкери, що завжди були тісно пов'язані зі своїм музичним смаком, та вважають музику одним з найважливіших аспектів своєї культури. Їхня любов до мотоциклів, музики, та байкерських активностей не залежить від умов на вулицях, адже більшість з них можна провести у закритих просторах, таких як клуби, або спеціально обладнаних майданчиках. Тематичні байкерські фестивалі залишаються однією з небагатьох можливостей для вуличних гонщиків залишатися активними та зберігати свої зв'язки з іншими членами спільноти.

Сьогодні ми живемо у часи, коли розповсюдженість обчислюваних пристроїв та мережі інтернет досягли свого піку. Інформаційні технології використовуються майже у всіх сферах життя людини, починаючи від особистого блогу, і закінчуючи складними системами автоматизації виробничих та бізнес-процесів. Вже у XXI столітті спілкування в Мережі стало невід'ємною частиною життя кожної людини. Багато людей віддають перевагу віртуальному спілкуванню, адже дістати з кишені невеличкий пристрій та написати повідомлення набагато легше ніж організувати живу зустріч.

Однак, у світі, де технології розвиваються з неймовірною швидкістю, байкери продовжують зберігати свої традиції та спілкуватись у живих зустрічах. Зараз, коли люди все більше та більше часу проводять в інтернеті, байкерські клуби стали своєрідними "островами" реального спілкування, якого часом так не вистачає в епоху соцмереж та соціальної дистанції. Організація таких фестивалів може бути важким завданням, та потребує багато часу та зусиль, включаючи пошук, вибір локації, планування та промоцію серед спільноти. Дана дипломна робота спрямована на вирішення цих проблем.

Зазвичай, для просування публічних заходів використовуються різні популярні платформи та сервіси, такі як Facebook, Instagram, Viber або Telegram, адже вони надають можливість обміну повідомленнями в реальному часі. Усі вони є якісними продуктами, що по різному вирішують поставлені задачі, але незважаючи на поширеність та доступність цих соц-мереж, створення вузькоспеціалізованого додатку матиме ряд переваг.

Перше, він спеціально націлений на потреби байкерів, що дозволить більш точно і ефективно задовольняти їх потреби. Такий додаток буде мати інтеграцію з картами для планування маршрутів, спеціальні функції для організації та участі у байкерських фестивалях, а також можливість створювати і приєднуватися до тематичних груп чатів.

Друге, мобільний застосунок для байкерів може мати унікальні функції, які недоступні в загальних соціальних мережах. Так, він може надавати користувачам спеціальні стикери, фільтри та емодзі, пов'язані з байкерською культурою, що дозволить їм виразити свою ідентичність та створити спільноту однодумців.

Третє, такий додаток може гарантувати більшу безпеку й конфіденційність для своїх користувачів, оскільки він не буде пов'язаний з великими корпораціями, що збирають дані про своїх користувачів. Це забезпечить байкерам спокій та впевненість у тому, що їх особиста інформація та спілкування залишаються в безпеці.

Четверте, застосунок може стати майданчиком для просування байкерської культури та рок-вечірок, оскільки він зосереджений на певній цільовій аудиторії.

Це дозволить легко рекламувати та просувати байкерські заходи, організаторам яких не доведеться конкурувати з безліччю інших подій, що просуваються у загальних соціальних мережах. Таким чином, застосунок сприятиме збільшенню відвідуваності та популярності байкерських заходів.

П'яте, мобільний застосунок для байкерів може мати гнучкі налаштування конфігурації та персоналізації, що дозволить кожному користувачу налаштувати додаток відповідно до власних потреб та смаків. Від користувачів, що шукають лише інформацію про майбутні заходи, до активних учасників байкерської спільноти - кожен зможе знайти в додатку щось для себе.

Шосте, спеціалізований додаток може сприяти створенню позитивного іміджу байкерської спільноти у суспільстві. Надаючи можливість користувачам історіями про свої пригоди, додаток може залучати до цієї субкультури нових зацікавлених людей та розповідати їм про позитивні аспекти байкерського життя.

Враховуючи все вище сказане, застосунок для байкерів може стати надійним помічником, допомагаючи їм спрощувати організацію та планування своїх музичних заходів. Розробка Android-додатку для байкерів, який дозволяє створювати та організовувати тематичні рок-вечірки та відмічати їх на карті, може значно полегшити спілкування та організацію заходів для байкерських клубів. Такий додаток не тільки дозволить знайти нових друзів, але й допоможе зміцнити зв'язок між існуючими клубами, зберегти та розвинути байкерської культури в цілому.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І ТЕХНОЛОГІЙ

1.1 Актуальність розробки під ОС Android

Роль мобільного телефону у сучасному житті важко переоцінити. За допомогою цього компактного приладу ми в будь-який момент можемо швидко зв'язатися зі своїми родичами та друзями, колегами по роботі, або дізнатися цікаву для нас інформацію. При цьому не треба напружувати пам'ять і згадувати телефон старого знайомого, якому вже давно не набирали, записник мобільного телефону завжди під рукою. Багато окрім контактів зберігають на телефоні ще масу іншої інформації – записують свої ідеї та думки, номери кредитних карток, пам'ятні дати, різні файли тощо. Тобто телефон замінює нам відразу записник і flash-накопичувач. А всілякі корисні програми, функції, опції якими оснащені сучасні телефони, розповсюдженість роблять цей невеликий за розміром пристрій багатofункціональним інструментом для вирішення багатьох задач.

На сьогоднішній день Android це вже не просто операційна система для смартфона, а ціла інфраструктура. На «зеленому роботі» працюють телефони, планшети, телевізори, розумні годинники та інші гаджети, а незабаром і автомобілі будуть керуватися за допомогою Android, який сьогодні завдяки різноманітності пристроїв та підтримці Google є найпоширенішою мобільною ОС у світі.

Окрім цього, одним із визначальних факторів популярності цієї системи є те, що для Android-розробників створено безліч документації, яка спрощує комунікації між фахівцями по всьому світу та дозволяє новачкам швидше опанувати навички розробки. Завдяки цьому щодня у Google Play Market з'являється безліч нових застосунків, які спрощують життя мільйонів людей.

Крім зазначеної вище популярності та поширеності, актуальність розробки саме під ОС Android обумовлена попитом цієї платформи серед байкерської спільноти. Багато мотоциклістів користуються Android-пристроями як навігаторами та плеєрами для музики під час їзди, тому, що байкерська культура відома своїм бажанням кастомізувати все, що використовують її послідовники,

починаючи від одягу, та закінчуючи мотоциклами. Android забезпечує можливість налаштування та персоналізації системи під конкретні потреби користувача, що відповідає байкерській філософії творчого та індивідуального підходу до життя. Таким чином, розробка додатку саме на ОС Android може забезпечити додаткову залученість та задоволення користувачів з байкерської спільноти.

Враховуючи усі попередні особливості системи, створення додатку для цієї ОС дозволить охопити якомога більше користувачів серед байкерів та забезпечити зручне та безпечне спілкування у мережі під час їзди.

1.2 Огляд існуючих рішень для спілкування у мережі та роботі з мапою

На сьогоднішній день люди в Україні мають обмежений вибір додатків для комунікації та навігації на дорозі. Існуючі рішення, такі як Harley-Davidson App, BMW Motorrad Connected і Indian Motorcycle Ride Command, пропонують користувачам можливість отримувати сповіщення про стан їхніх мотоциклів, а також відстежувати свій пробіг та налаштовувати режими їзди. Однак, жоден з цих додатків не є доступним українським користувачам.

Таким чином, байкери в Україні змушені використовувати загальнодоступні соціальні мережі, серед яких можна виділити найпопулярніші: Telegram, Facebook, Viber. Якщо ми говоримо про застосунки для роботи з мапою, то найпоширенішим інструментом є Google Maps, враховуючи, що пакет сервісів та додатків від Google за замовчуванням встановлюється разом із ОС Android.

1.2.1 Telegram

Telegram - це безкоштовний месенджер, який має велику кількість користувачів. Ця система пропонує високий рівень безпеки та приватності, проте не має вбудованої карти. Отже, в Telegram відсутні спеціальні функції для байкерів, такі як можливість відстежувати маршрут, планувати заходи чи знаходити місця для зустрічей.

1.2.2 Facebook

Facebook - це соціальна мережа, яка також має свій власний месенджер. Він має високу кількість користувачів та різноманітні функції, але не завжди гарантує приватність та безпеку даних юзерів. Нажаль, як і Telegram, Facebook не задовольняє потребам байкерів, що так полюбляють планувати спільні поїздки на карті, зберігати улюблені маршрути та створювати закриті спільноти.

1.2.3 Viber

Viber - це ще один месенджер з можливістю дзвінків та відеозв'язку. Він також має високий рівень безпеки, але не завжди надійно працює в умовах слабкої мережі. Враховуючи, що основна мета цього продукту – це обмін повідомленнями, то розробниками не було не було передбачено можливості відстежувати та побудувати оптимальні маршрути за допомогою карти.

1.2.3 Google Maps

Google Maps - це сервіс карт, який має широкий функціонал для навігації. Він дозволяє побудувати оптимальний маршрут, відстежувати маршрут та отримувати інформацію про дорожні затори та пробки. Проте, додаток від пошукового гіганту не має можливості комунікувати з іншими користувачами та не має вбудованого месенджера, а також не пропонує функцій, специфічно розроблених для байкерів, будь то співпраця з мотоциклетними клубами, чи організація та відвідування мотоциклетних заходів.

Враховуючи ці недоліки, вузькоспеціалізований додаток для байкерів міг би запропонувати більш комплексне рішення, що включає як навігаційні функції, так і можливості для спілкування та співпраці в мотоциклетній спільноті. Такий додаток може мати вбудований месенджер з можливістю відправки повідомлень, спеціальні функції для планування та організації спільних поїздок, підтримку мотоциклетних клубів та організацій, а також інтеграцію з інтерактивною цифровою картою для візуалізації поїздок і заходів для забезпечення більшої видимості та залучення нових учасників.

1.3 Аналіз існуючих рішень побудови архітектури розподіленої системи

Розробку будь-якої системи необхідно починати з проектування.

У сфері інформаційних технологій для створення великих і складних систем існують певні рекомендації та стандарти, що називаються архітектурою.

Існує досить багато підходів для побудови складних систем із гарною архітектурою. Незважаючи на невеликі відмінності цих підходів, вони мають багато спільного. Зрозуміло, що всі вони задають способи розбиття програми на окремі модулі. При цьому в кожній системі є як мінімум модулі, що містять бізнес-логіку програми, і модулі для відображення даних.

Кожна архітектура несе у собі рекомендаційний характер, вона лише описує як краще організувати роботу систему, щоб її можна було легше підтримувати та масштабувати. У розробці даної буде розглядатися багаторівнева монолітна архітектура.

Моноліт означає, що складається все в одній частині. Монолітна архітектура описує однорівневий програмний додаток, у якому різні компоненти об'єднані в одну програму з однієї платформи. Компонентами можуть бути:

- авторизація — відповідає за авторизацію користувача;
- презентація — відповідає за обробку HTTP-запитів і відповіді за допомогою HTML або JSON/XML (для API веб-сервісів);
- API (application programming interface) – набір чітко визначених методів, функцій, протоколів, процедур або команд, які використовує програмне забезпечення з можливостями мов програмування для виклику служб;
- бізнес-логіка — бізнес-логіка програми;
- рівень бази даних — об'єкти доступу до даних, відповідальні за доступ до бази даних;
- інтеграція додатків — інтеграція з іншими сервісами або інтеграція з будь-якими іншими джерелами даних [1].

Переваги:

- проста розробка – метою сучасних інструментів розробки та IDE є підтримка розробки монолітних додатків;
- просте розгортання – просте розгортання файлу або ієрархію каталогів у відповідному середовищі виконання;
- просте масштабування – масштабування програми, запустивши кілька копій програми за балансувальником навантаження.

Недоліки:

- велика монолітна база коду лякає розробників, особливо тих, хто є новичком у команді, програму може бути важко зрозуміти та змінити, як наслідок, розвиток зазвичай сповільнюється;
- перевантажена IDE – чим більша база коду, тим повільніше IDE та менш продуктивні розробники;
- перевантажений веб-контейнер – чим більше програма, тим більше часу потрібно для її запуску, це дуже вплинуло на продуктивність розробників, оскільки час витрачався на очікування запуску контейнера;
- постійне розгортання є складним – велика монолітна програма також є перешкодою для частих розгортань, щоб оновити один компонент, потрібно повторно розгорнути всю програму.
- потрібна довгострокова прихильність до стеку технологій — монолітна архітектура змушує бути із технологічним стеком, який був вибраний на початку розробки, при монолітному застосуванні може бути важко поступово впроваджувати новішу технологію [2].

Зважаючи на переваги та недоліки, монолітна архітектура задовольняє багатьом вимогам малих та середніх проектів, яким є ця дипломна робота. Програмний продукт не є занадто складним і на разі знаходиться на початковій стадії, в якій немає багато функціоналу для необхідності його розподілення за мікросервісною архітектурою. Однак архітектура має бути ретельно обдумана, щоб мати можливість подальшого масштабування без втрат швидкодії системи.

Отже, майбутня розподілена система складається з трьох рівнів: клієнтська частина, сервер додатків та сервер бази даних.

Клієнтська частина – рівень відображення, що відповідальний за відображення інформації та забезпечення людино-машинної взаємодії. Клієнтська частина представляє собою графічний інтерфейс, за допомогою якого людина може відправляти запити на сервер і використовувати систему. Протоколом передачі даних є HTTP з використанням REST архітектури. Дані передаються у форматі JSON [4].

Сервер додатку – серверна частина системи, що відповідальна за коректну обробку усіх запитів користувача та обробку даних у базі. Серверна частина виконує основну логіку роботи системи.

Сервер бази даних – рівень бази даних, на якому знаходиться СКБД та сама база даних.

БД – база даних, збір даних, організованих відповідно до концептуальної структури, що описує характеристики цих даних і відносини між їх відповідними сутностями, що підтримують одну або кілька областей застосування [18].

СКБД – це система керування базами даних, заснована на апаратному та програмному забезпеченні, для визначення, створення, маніпулювання, контролю, керування та використання баз даних [18].

Від серверу за допомогою підключення по TCP/IP надходять запити до СКБД, яка їх виконує і віддає дані згідно запиту.

1.4 Аналіз технологій для клієнтської частини системи

Існує велика кількість мов, за допомогою яких можна створити якісну програму. Деякі мови мають вбудоване середовище розробки, що значно полегшує процес створення програми. Інші потребують додаткових інструментів, проте їх функціонал не менш широкий.

1.4.1 Java

Java – один із найстаріших та перевірених методів створення якісної програми. Це популярна мова програмування для андроїд, яка у 2019 році увійшла до ТОП 5 найкращих та функціональних мов. Проте в той же час вона досить складна, і для новачків не завжди підходить. Для того, щоб навчитися програмувати на Java необхідні знання з об'єктно-орієнтованого програмування. Це означає, що він працює з класами, винятками, конструкторами, за якими потрібно ретельно стежити та продумувати логіку.

Головна перевага Java - наявність власного середовища розробки під назвою Android Studio. У 2014 році вона була визнана компанією Google як офіційне середовище андроїд програмування, що значно полегшило життя розробникам. Процес розробки програм спрощується за рахунок візуального UI-редактора, функції автодоповнення коду та інших можливостей [5].

Що стосується функціональності, Java відноситься до однієї з найпотужніших мов. Вона дозволяє реалізувати найскладніші завдання та проекти, тому користується такою популярністю.

1.4.2 Kotlin

Kotlin – це відносно молода мова програмування, яка з'явилася у 2017 році. Але за такий короткий проміжок вона змогла завоювати любов багатьох розробників, і в 2019 була визнана компанією Google кращою мовою програмування для андроїд, відсунувши Java на друге місце.

Kotlin увібрав у собі найкращі якості інших мов. Він має масу переваг, серед яких автоматичне визначення типів даних, функції-розширення та інші. Він простий у розумінні, освоєнні та доступний для кожного. Kotlin створено на основі Java, тому переходити з Java дуже просто [6].

Мова легко інтегрується з багатьма фреймворками, проста для вивчення, її код відкритий. Завдяки зрозумілому синтаксису підвищується продуктивність при створенні додатків. Легкість читання та лаконічність – основні якості Kotlin.

Але при цьому він має недоліки. Швидкість компіляції коду нестабільна, може проходити як швидко, так і зі значними затримками. Мова нова, тому на неї

не так багато навчального матеріалу, тому в ході виникнення складнощів або питань розробникам нерідко доводиться самотійно шукати шляхи вирішення. І все ж, Kotlin по праву вважається одним із найкращих інструментів в Android програмуванні.

Враховуючи зручність мови, масовий перехід розробників з Kotlin на Java, а також підтримку Google, в цій роботі у якості основної мови програмування був обраний Kotlin.

1.4.3 Xml

Як правило, для визначення візуального інтерфейсу в проектах під Android використовують спеціальні файли xml. Ці файли є ресурсами розмітки та зберігають визначення візуального інтерфейсу як код XML. Подібний підхід нагадує створення веб-сайтів, коли інтерфейс визначається у файлах html, а логіка програми – у коді javascript.

Оголошення інтерфейсу користувача у файлах XML дозволяє відокремити інтерфейс програми від коду. Що означає, що ми можемо змінювати визначення інтерфейсу без зміни коду java чи kotlin. Наприклад, у програмі можуть бути визначені розмітки у файлах XML для різних орієнтацій монітора, різних розмірів пристроїв, різних мов тощо. Крім того, оголошення розмітки в XML дозволяє легше візуалізувати структуру інтерфейсу та полегшує налагодження.

1.4.4 Jetpack Compose

Jetpack Compose представляє сучасний тулкіт від компанії Google для створення програм під ОС Android мовою програмування Kotlin. Jetpack Compose спрощує написання та оновлення візуального інтерфейсу програми, надаючи декларативний підхід.

Щоб спростити розробку, зробити її швидшою, простішою, спростити підтримку компанія Google у травні 2019 року анонсувала новий тулкіт - Jetpack Compose.

Jetpack сумісний з наявним набором бібліотек Android, які можна використовувати в стандартних проектах на Java та Kotlin для написання програм під Android. Відмінною рисою Jetpack Compose є те, що він пропонує кардинально

інший підхід до створення додатків. Перш за все, Jetpack Compose пропонує використовувати мову Kotlin та всі його переваги. Відповідно до роботи з тулкітом необхідно мати базові знання цієї мови. Цей інструмент зменшує обсяг коду та пропонує декларативний API, який є інтуїтивнішим.

Ключовою концепцією тулкіта Jetpack Compose є функція `composable` (функція, яка має анотацію `@Composable`). Такі функції представляють деякі частини візуального інтерфейсу, у тому числі з яких будується додаток. Це спрощує побудову та оновлення складних інтерфейсів, тестування та підтримку самих компонентів.

Нажаль, на поточний момент технологія Jetpack Compose дуже погано інтегрується з існуючими рішеннями вбудованих карт на Android через те, що усі бібліотеки для розгортання мапи створювалися для співпраці з життєвим циклом таких Android компонентів, як Activity або Fragment, що відсутні при побудові додатку на Compose. Враховуючи це обмеження, у дипломній роботі було прийнято рішення використовувати XML як основну технологію для створення UI-компонентів системи.

1.4.5 JSON

JSON (JavaScript Object Notation) - простий формат обміну даними, зручний для читання та написання як людиною, так і комп'ютером. Він заснований на підмножині мови програмування JavaScript, визначеної у стандарті ECMA-262 3rd Edition - December 1999. JSON - текстовий формат, повністю незалежний від мови реалізації, але він використовує угоди, знайомі програмістам С-подібних мов, таких як C, C++, C# , Java, JavaScript, Perl, Python та багато інших. Ці властивості роблять JSON ідеальною мовою обміну даними.

JSON заснований на двох структурах даних:

Колекція пар ключ/значення. У різних мовах ця концепція реалізована як об'єкт, запис, структура, словник, хеш, іменованний список або асоціативний масив.

Упорядкований перелік значень. У більшості мов це реалізовано як масив, вектор, список чи послідовність.

Це є універсальні структури даних. Майже всі сучасні мови програмування підтримують їх у будь-якій формі. Логічно припустити, що формат даних, незалежний від мови програмування, має ґрунтуватися на цих структурах.

У даній дипломній роботі формат JSON використовується для обміну інформацією між клієнтською та серверною частинами розподіленого додатку.

1.4.6 MVVM

Для підвищення стабільності, читабельності та підтримуваності програми варто дотримуватися чіткої архітектури.

MVC та MVP - це шаблони архітектури, які були популярні в Android-розробці до введення MVVM. В шаблоні MVC, модель містить бізнес-логіку та дані, контролер забезпечує зв'язок між моделлю та уявленням, а уявлення відповідає за відображення даних користувачеві. У MVP, презентер відповідає за зв'язок між моделлю та уявленням, а уявлення забезпечує відображення даних.

Model-View-ViewModel (MVVM) - це шаблон архітектури клієнтських додатків, запропонований Джоном Госсманом (John Gossman) як альтернатива шаблонам MVC і MVP під час використання технології зв'язування даних (Data Binding). Його концепція полягає у відділенні логіки уявлення даних від бізнес-логіки шляхом винесення їх у окремий клас для чіткішого розмежування [3].

З MVVM процес розробки графічного інтерфейсу для користувачів поділяється на дві частини. Перша – це робота з мовою розмітки або кодом GUI. Друга – розробка бізнес-логіки або логіки бекенда (модель даних). Частина ViewModel у MVVM – це конвертер значень. Це означає, що ViewModel відповідає за конвертування об'єктів даних із моделі в такий вигляд, щоб з об'єктами було легко працювати. Якщо дивитися з цієї сторони, то ViewModel це скоріш модель, ніж уявлення. Вона контролює більшу частину логіки відображення.

Схематична взаємодія компонентів MVVM наведена на рис. 1.1.



Рисунок 1.1 - Схематична взаємодія компонентів MVVM

Уявлення (View) містить структурне визначення, що користувачі отримають на екранах. Сюди можна помістити статичний та динамічний вміст (анімацію та зміну станів). Тут не може бути ніякої логіки додатку. Android представленням може бути активність або фрагмент.

Модель-уявлення (ViewModel) пов'язує модель та уявлення. Відповідає за керування посиланнями даних та можливих конверсій. Тут з'являється біндінг. В Android ми не турбуємося про це, тому що можна використовувати клас `AndroidViewModel` або `ViewModel`.

Модель (Model) - це рівень бізнес-даних і він не пов'язаний з жодним особливим графічним уявленням. В Android, згідно з “чистою” архітектурою, модель може містити базу даних, репозиторій та клас бізнес-логіки.

1.4.7 Dependency injection

Dependency injection (DI) – це процес надання зовнішньої залежності програмному компоненту. У повній відповідності до принципу єдиного обов'язку об'єкт делегує побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього загальному механізму.

Ін'єкція залежностей широко використовується в програмуванні і добре підходить для розробки Android. Дотримуючись принципів DI, розробник закладає основу для гарної архітектури програми.

Реалізація ін'єкції залежностей надає наступні переваги: можливість повторного використання коду, простота рефакторингу, простота тестування.

Популярний спосіб реалізації DI в Android-додатках заснований на фреймворку `Dagger`. Але він потребує глибокого вивчення. Одна з найкращих

альтернатив цьому фреймворку - Koin, бібліотека, написана на чистому Kotlin, яка і використовуватиметься в даній роботі.

1.4.8 RxJava

Останнім часом підтримка асинхронності та паралельних обчислень стала невід'ємною рисою багатьох мов програмування. Паралельні обчислення дозволяють виконувати кілька завдань одночасно, а асинхронність дозволяє не блокувати основний хід програми під час виконання завдання, яке займає тривалий час. Наприклад, потрібно створити графічний додаток для робочого столу або мобільного пристрою. Після натискання кнопки необхідно надсилати запит до інтернет-ресурсу. Однак подібний запит може тривати тривалий час. І щоб програма не зависала на період відправки запиту, подібні запити до інтернет-ресурсів слід надсилати асинхронно. При асинхронних запитах користувач не чекає, поки прийде відповідь від інтернет-ресурсу, а продовжує роботу з додатком, а при отриманні відповіді отримає відповідне повідомлення.

RxJava – один із прикладів реалізації шаблону спостерігач, що включає механізми асинхронної роботи. Це бібліотека для програмування реактивних програм на мові Java, яка дозволяє асинхронно та ефективно працювати з потоками даних та подій. RxJava реалізує паттерн Reactive Extensions, який відповідає за обробку подій та потоків даних в асинхронному стилі [11].

Деякі з основних переваг цієї технології включають покращення продуктивності додатків, забезпечення більшої ефективності мережевих викликів та спрощення розробки асинхронного коду. Крім того, RxJava дозволяє розробникам використовувати більшість стандартних операцій для роботи з даними, таких як фільтрація, мапування та згортання. Проте, з використанням RxJava можуть виникати деякі проблеми. Недостатнє розуміння принципів реактивного програмування та невірне використання потоків може призвести до проблем з продуктивністю додатків та несподіваної поведінки програми. Крім того, використання RxJava може стати складним завданням для початківців у програмуванні та вимагати великої кількості часу та зусиль для вивчення її функціональності. Враховуючи те, що Android-додаток буде написаний на мові

Kotlin, має сенс розглянути вбудований в цю мову інструмент для асинхронної роботи – Kotlin Coroutines.

1.4.9 Kotlin Coroutines

Kotlin Coroutines - це механізм у мові Kotlin, що підтримує асинхронність та паралельні обчислення через використання співпрограм. Корутина - це блок коду, який може працювати асинхронно, тобто його виконання може зупинятись та продовжуватись згідно з розкладом. Це дозволяє виконувати кілька функцій одночасно та зменшує час очікування на завершення завдань. Корутини дозволяють забезпечити асинхронність коду без використання колбеків або промісів, що робить код більш зрозумілим та зручним у розробці та підтримці..

Крім корутин, Kotlin також має підтримку асинхронних потоків даних за допомогою Kotlin Flows. Flow - це асинхронний потік даних, який може випускати нуль або більше значень на споживача в будь-який момент часу [12]. Kotlin Flows надають зручний і безпечний спосіб передачі потоку даних між компонентами додатка.

Основна відмінність між Kotlin Flows та RxJava полягає в тому, що Kotlin Flows є частиною мови Kotlin, тоді як RxJava є окремою бібліотекою. Kotlin Flows засновані на корутинах та працюють з ними в синхронному режимі, що робить їх більш легкими та швидкими. Також, Kotlin Flows мають вбудовану підтримку багатопоточності та асинхронної обробки потоків даних, що робить їх зручними для розробки серверних додатків. Однак, RxJava має більше можливостей та більший спектр операторів для роботи з потоками даних, що робить його корисним для складних випадків.

1.5 Аналіз технологій для серверної частини системи

Існує безліч способів створення серверної частини системи для розробки розподілених додатків. Для створення гнучкої логіки, яка може масштабуватися з

часом при зміні бізнес-вимог, має сенс використання серверних мов програмування, таких як PHP, Java, Python, Node.js та інших. Кожна з них має свої переваги та недоліки, і вибір конкретної мови залежить від вимог проекту та власних уподобань.

Однак, на сьогоднішній день все більш популярними стають готові рішення, такі як Firebase, які дозволяють швидко та ефективно розгорнути свій бекенд без необхідності займатися його написанням та обслуговуванням. Firebase надає доступ до різноманітних сервісів, таких як бази даних, зберігання файлів, автентифікація користувачів, push-сповіщення та інші, що дозволяє зосередитися на розробці функціоналу додатку, а не на розгортанні та налаштуванні серверів.

1.5.1 PHP

PHP (рекурсивний акронім словосполучення PHP: Hypertext Preprocessor) – це поширена мова програмування загального призначення з відкритим вихідним кодом. PHP спеціально сконструйований для веб-розробок, і його код може впроваджуватися безпосередньо в HTML. PHP відрізняється від JavaScript тим, що PHP-скрипти виконуються на сервері та генерують HTML, який надсилається клієнту. Замість рутинного виведення HTML-коду командами мови (як це відбувається, наприклад, Perl або C), скрипт PHP містить HTML з вбудованим кодом. Код PHP відокремлюється спеціальними початковими та кінцевими тегами `<?php і ?>`, які дозволяють "перемикатися" в "PHP-режим" і виходити з нього [13].

PHP орієнтований на розробку веб-додатків та веб-сайтів, отже має сенс розглянути інші рішення, які будуть більш підходити для створення розподіленого додатку з клієнтською частиною на мобільній платформі.

1.5.2 Java Servlets

Java Servlets - це технологія Java, яка дозволяє писати веб-додатки, які можуть оброблювати запити HTTP і відповідати на них. З допомогою Java Servlets можна реалізувати багато функцій, таких як формування відповідей, обробка даних форм, авторизація та аутентифікація, зберігання та доступ до даних, а також інші функції, що забезпечують повноцінну роботу веб-додатку. Java Servlets є однією з найпопулярніших технологій веб-розробки в сфері корпоративної розробки.

1.5.3 Node.js

Node.js - це середовище виконання мови JavaScript, яке дозволяє виконувати JavaScript на стороні сервера. Ця технологія є дуже ефективною, оскільки вона дозволяє писати серверний код на тій же мові, що використовується на стороні клієнта, що робить розробку веб-додатків більш зручною та продуктивною. Node.js також дозволяє забезпечувати взаємодію між клієнтом та сервером у реальному часі, що зробило його популярним у сфері розробки веб-додатків [14].

1.5.4 Firebase

Firebase - це хмарна платформа від компанії Google, що дозволяє розробникам створювати серверну частину веб та мобільних застосунків без необхідності писати власний бекенд. Firebase пропонує ряд сервісів, таких як хмарне зберігання даних, аутентифікація користувачів, аналітика, хостинг, push-сповіщення та інші. Завдяки простоті використання та інтеграції з іншими сервісами Google, Firebase є популярним рішенням для створення прототипів веб та мобільних застосунків, особливо для стартапів та розробників з обмеженими ресурсами, адже дозволяє сконцентруватися на функціональності клієнтської сторони додатку.

Основними сервісами Firebase є Storage, Firebase Database та Authentication. Розглянемо кожен з цих сервісів детальніше.

Firebase Storage - це облачний сервіс зберігання файлів, який дозволяє розробникам зберігати та отримувати файли (наприклад, зображення, відео, аудіо) з будь-якої мобільної платформи. Firebase Storage забезпечує безпеку та надійність збереження файлів, а також забезпечує швидкий доступ до них. Розробники можуть використовувати Firebase Storage для зберігання користувацьких даних, таких як фотографії профілю або різноманітні типи файли [16].

Firebase Database - це облачна база даних реального часу, яка надає швидкий та надійний доступ до даних на будь-якому пристрої. Firebase Database дозволяє розробникам зберігати та синхронізувати дані між різними пристроями та платформами. Це надає можливість створювати реактивні мобільні додатки з функцією обміну даними в режимі реального часу [16].

Firebase Authentication - це сервіс, який забезпечує аутентифікацію користувачів у мобільних додатках. Він дозволяє користувачам увійти в додаток за допомогою різних методів, таких як адреса електронної пошти, номер телефону або обліковий запис Google або Facebook. Розробники можуть використовувати Firebase Authentication для забезпечення безпечного з'єднання та аутентифікації користувачів у застосунку [16].

Враховуючи, що ця платформа від Google надає повний стек інструментів для створення та супроводу серверної частини додатку в одному місці, а також можливість надання реєстрації через облікові записи Google та Facebook, що значно збільшить зручність входу у додаток для нових користувачів, у якості серверної частини для дипломного проекту буде використовуватися Firebase.

1.6 Висновки щодо аналізу технологій

Провівши аналіз клієнтських та серверних технологій, можна зробити певні висновки щодо того, яку технологію слід використовувати при розробці системи.

Один з головних аспектів, які було враховано при виборі технологій для розробки системи для байкерів - це популярність та поширеність серед потенційних користувачів. Сьогодні, Android є однією з найбільш поширеною ОС для смартфонів та інших мобільних пристроїв у світі. Він займає більш як 70% ринку мобільних пристроїв, що робить його вибором номер один для розробки додатків. Таким чином, розробка системи на базі Android дає можливість залучити велику кількість користувачів та забезпечити максимальне охоплення аудиторії.

Під час проектування системи було обрано монолітну архітектуру через її простоту та зручність у розробці. У монолітній архітектурі всі компоненти додатку належать до одного цілого та знаходяться в одному місці, що спрощує процес розробки та тестування. Крім того, монолітна архітектура дозволяє швидко внести зміни у систему, так як не потрібно робити складні налаштування та розгортання

компонентів додатку на різних серверах. Враховуючи масштаб проекту, що розробляється у даній дипломній роботі, така архітектура відповідає усім вимогам додатку.

Для розробки системи було обрано мову програмування Kotlin. Однією з головних причин є те, що вона базується на JVM, тобто вона може взаємодіяти з більшістю інших мов, що працюють на цій платформі. Також Kotlin є більш сучасною мовою програмування, порівняно з Java, тому має ряд переваг. Зокрема, Kotlin надає більше можливостей для роботи з нульовими значеннями та забезпечує безпеку виконання коду, завдяки системі типів. Також, Kotlin є більш ефективною мовою, що дозволяє зменшити кількість написаного коду порівняно з Java, та зменшити час на розробку й кількість можливих помилок. Kotlin активно просувається Google, що гарантує, що ця мова буде розвиватися та підтримуватися в майбутньому.

Для верстки інтерфейсу системи була обрана мова XML, оскільки вона має ряд переваг порівняно з альтернативними рішеннями. XML-файли зберігають визначення візуального інтерфейсу як код XML, що дозволяє відокремити його від коду програми. Це забезпечує більшу гнучкість у зміні визначення інтерфейсу без зміни коду на Java чи Kotlin. Крім того, використання XML спрощує налагодження інтерфейсу, полегшує візуалізацію його структури та дозволяє визначати розмітку для різних орієнтацій монітора та різних розмірів пристроїв. Хоча Jetpack Compose є новітнім тулкітом для створення програм під ОС Android, і пропонує кардинально інший підхід до створення додатків, на поточний момент він досить обмежений, зокрема у питанні інтеграції з існуючими рішеннями вбудованих карт на Android, тому у дипломній роботі було прийнято рішення використовувати XML як основну технологію для створення UI-компонентів системи.

Паттерн архітектури MVVM був обраний для додатку на Android через його переваги порівняно з іншими шаблонами архітектури, такими як MVP і MVC. MVVM надає чітке відокремлення логіки уявлення від бізнес-логіки, що дозволяє розробникам більш ефективно працювати зі створенням та тестуванням компонентів. Крім того, MVVM підтримує зв'язування даних (Data Binding), яке

дозволяє автоматично оновлювати відображення даних відповідно до їх змін в моделі. Це дозволяє зменшити кількість коду, який потрібен для взаємодії між компонентами, та зробити процес розробки більш простим та швидким.

Для асинхронної роботи на Android, була обрана технологія Kotlin Coroutines, що пропонує зрозумілий та приємний для програміста синтаксис, який дозволяє писати більш чистий та зрозумілий код. Крім того, Kotlin Coroutines є частиною мови Kotlin, тому не потребує додаткових бібліотек, що зменшує кількість залежностей та спрощує процес розробки. Також вони мають вбудовану підтримку життєвого циклу Android та інших компонентів фреймворку, що дозволяє легко і безболісно інтегрувати їх у проект. Врешті-решт, корутини мають підтримку Kotlin Flows, що дозволяє зручно та ефективно працювати з потоками даних. Всі ці переваги дозволяють використовувати Kotlin Coroutines як більш просту та зручну альтернативу RxJava для асинхронних операцій.

Під час аналізу технологій для реалізації ін'єкції залежностей був зроблений вибір на користь бібліотеки Koin. Хоча Dagger має більше можливостей та підтримку від Google, його складність може бути перешкодою для новачків в Android-розробці. Крім того, для менших та середніх проектів використання такого важкого фреймворку може бути надмірним. Однією з основних переваг Koin є його легкість використання та конфігурування, що дозволяє швидко налаштувати ін'єкцію залежностей без необхідності вивчення складної документації. Крім того, Koin є більш гнучким та має більшу інтеграцію з Kotlin, що дозволяє простіше використовувати функціональні можливості мови.

Для розробки серверної частини на Android була обрана популярна платформа від компанії Google - Firebase, яка надає безкоштовні та платні сервіси для розробки мобільних та веб-додатків. Використання Firebase дозволить значно скоротити час розробки та підвищити продуктивність, оскільки багато важливих задач, таких як автентифікація, зберігання даних та реалізація push-сповіщень вже імплементовані та легко налаштовуються. Окрім цього, Firebase має детальну документацію та готові приклади, що допоможе розробникам швидко зрозуміти, як використовувати платформу в їх проектах.

2 РОЗРОБКА КЛІЄНТСЬКОЇ ТА СЕРВЕРНОЇ ЧАСТИН СИСТЕМИ

2.1 Етапи розробки розподіленого додатку

Процес розробки розподіленого додатку проходить в декілька етапів:

- Backend розробка - ключовий етап створення внутрішньої частини, який визначає всю роботу програми. На цьому етапі створюють невидиму для клієнта структуру – базу даних, сервер, бізнес-логіку. Програмісти використовують мови програмування для серверних операцій: PHP, Python, Java, C# та навіть Kotlin. Під час внутрішньої розробки виконують обслуговування запитів зовнішнього інтерфейсу, створення, читання, оновлення даних тощо.

- Створюється інтуїтивно зрозумілий інтерфейс робочого типу програми без етапів програмування. Дизайн - це те, що бачить кінцевий користувач системи, і саме він відповідає за перше враження від програми. На цьому етапі продумуються переходи між екранами, графіка зображень та кнопки;

- Розробка клієнтського додатку - створюється зовнішня інтерфейсна частина мовами програмування, у випадку ОС Android - Xml, Java або Kotlin. На цьому етапі програмісти перевіряють чуйність та зручність інтерфейсу на різних типах пристроїв, щоб взаємодія з користувачем була узгодженою. Швидкість роботи не повинна залежати від розміру чи роздільної здатності екрана.

- Випробування ПЗ – проводиться випробування всіх модулів програми. Перевіряється зв'язок, підключення до бази даних, навігація між сторінками, продуктивність, сумісність з різними пристроями, а також загальний досвід користувача.

- Розміщення та розгортання (у випадку клієнтського мобільного додатку) – готовий проект завантажується в магазин програм. AppStore та Google Play для платформ iOS та Android відповідно. Перш ніж програма стане доступною для користувачів, вона проходить модерацію. Тривалість перевірки залежить від багатьох факторів: категорії програми, її історії, розробника та завантаження команди модерації.

2.2 Визначення технічних вимог до системи

При проектуванні та розробці розподілених систем не менш важливим аспектом, ніж архітектура, є врахування технічних вимог до системи. Ці вимоги визначають мінімальний рівень ресурсів, необхідний для нормальної роботи системи. Зазвичай, вони залежать від типу додатка, його функцій та масштабів, а також від обраної технології. У даному розділі буде розглянуто основні аспекти визначення технічних вимог до розподіленої системи та вказівки щодо їх коригування залежно від специфіки проекту.

2.2.1 Вимоги до робочих характеристик серверної частини

Технічні вимоги до серверної частини додатку, що буде використовувати безкоштовний план Firebase, передбачають деякі обмеження, які необхідно врахувати. Зокрема, максимальна кількість запитів до бази даних Firebase Firestore становить 50 тисяч на день, а обсяг збереження даних не повинен перевищувати 1 ГБ. Враховуючи зазначені обмеження, можна визначити, що для стабільної роботи додатку кількість користувачів системи не має перевищувати 1000. Також важливо враховувати швидкість з'єднання з Інтернетом, оскільки вона може вплинути на продуктивність додатку.

Для забезпечення найкращої продуктивності рекомендується використовувати оптимізовані запити до бази даних та зменшувати кількість запитів у місцях, де це можливо, до мінімуму. Також важливо пам'ятати про безпеку даних та забезпечення їх конфіденційності під час передачі.

З урахуванням цих вимог та обмежень плану Firebase, можна переходити до підбору апаратних компонентів серверної частини. При їх підборі необхідно враховувати вимоги до продуктивності, надійності, доступності та ціни. У наступних пунктах наведено приклади середньобюджетних та бюджетних пристроїв, які усі в залежності від фінансового стану можуть бути використані для побудови функціонуючої системи.

Варто зазначити, що за допомогою Firebase і його готової хмарної платформи можна задовольнити всі вимоги до робочих характеристик серверної частини, забезпечуючи масштабовану, надійну та безпечну інфраструктуру для додатку, адже усі інструменти і сервіси розміщуються на готових серверах компанії Google.

2.2.1.1 Процесор

Для забезпечення вимог системи підійде чотирьохядерний процесор з достатньою частотою тактовою частотою для обробки клієнтських запитів. Процесор Intel Xeon E-2274G є надійним варіантом середнього рівня з чотирьохядерною конфігурацією і базовою частотою 3.8 ГГц, що забезпечує високу продуктивність для обробки запитів у реальному часі. Для тих, хто має обмежений бюджет, AMD Ryzen 5 3600 є відмінним вибором з шестиядерним процесором та базовою частотою 3.6 ГГц, забезпечуючи достатню потужність для серверних завдань.

2.2.1.2 Оперативна пам'ять

При виборі оперативної пам'яті враховувалися потреби в ємності та продуктивності. Для системи, що розроблюється у даній дипломній роботі, необхідно мінімум 8 ГБ оперативної пам'яті для забезпечення плавної роботи сервера та ефективної обробки запитів. Модуль Corsair Vengeance LPX DDR4 16 ГБ є надійним із великою ємністю, гарантуючи швидку та стабільну роботу системи. Але для тих, хто шукає більш доступний варіант, Crucial Ballistix DDR4 8 ГБ забезпечує достатню кількість оперативної пам'яті за доступну ціну.

2.2.1.3 Сховище даних

У плануванні сховища даних було враховано швидкість та надійність. Для цього підійде використання SSD-накопичувача з мінімальним обсягом, достатнім для зберігання даних, що потрібні для додатку. Samsung 970 EVO Plus NVMe PCIe M.2 500 ГБ є швидким SSD-накопичувачем великого обсягу, забезпечуючи швидку та надійну роботу з даними. Але для більш обмеженого бюджету Kingston A2000 NVMe PCIe M.2 250 ГБ є відмінним варіантом з адекватною продуктивністю при достатньому обсязі даних.

2.2.1.4 Мережеві можливості

У виборі мережевих можливостей достатньо наявності гігабітного Ethernet-порту для швидкого та стабільного з'єднання з мережею Інтернет. Intel Gigabit Ethernet Network Adapter є надійним адаптером з підтримкою гігабітного Ethernet, забезпечуючи стабільне з'єднання з мережею Інтернет. Також, як альтернатива з більш доступною ціною, TP-Link Gigabit Ethernet PCI Network Adapter пропонує задовільну продуктивність і підтримує гігабітну швидкість передачі даних.

2.2.1.5 Операційна система

Операційна система Ubuntu Server 20.04 LTS є рекомендованим вибором для серверного оточення. Вона надійна, безпечна і ефективно працює на різних серверних пристроях, забезпечуючи стабільну роботу додатку.

2.2.3 Вимоги до робочих характеристик клієнтської частини

Якщо мова йде про клієнтську частину, то мінімальною версією Android є 6.0 (Marshmallow), що дає можливість користуватись більшістю функцій та API, які необхідні для розробки мобільного додатку. Рекомендованим бюджетним рішенням, на якому була протестована система, є смартфон Redmi 9a, що забезпечить стабільну та безперебійну роботу додатку. Технічні характеристики Redmi 9a включають процесор MediaTek Helio G25 з тактовою частотою 2.0 ГГц, а також 2GB оперативної пам'яті. Для встановлення додатку пристрій повинен мати принаймні 100 мегабайт вільного простору. Такі параметри гарантують плавну роботу додатку та його стабільність на смартфоні з подібними характеристиками.

Враховуючи контекст використання додатку, для функціонування програмного забезпечення потрібне постійне інтернет-з'єднання. У випадку поганого зв'язку програмне забезпечення не може працювати, адже доступ до мережі необхідний для завантаження даних на клієнті з боку сервера, та авторизації у системі.

2.2.4 Зручність супроводження

Кожна з частин системи є окремим модулем, який повинен працювати автономно і бути реалізованим за стандартними архітектурними шаблонами, які

дозволять легко масштабувати проект, вводячи новий функціонал або змінюючи існуючий.

2.3 Визначення програмних вимог до системи

Визначившись з технічними характеристиками системи, потрібно визначити вимоги, яким ця система повинна відповідати з програмної точки зору. До таких вимог може відноситися складність структури користувацького інтерфейсу, швидкість роботи, стійкість до помилок та безпека цілісності даних.

2.3.1 Визначення вимог до безпеки даних

З точки зору захисту користувацьких даних розробка системи авторизації для клієнтського додатку на Android є важливим етапом. Авторизація описує процес аутентифікації користувача та його вхід до системи, а також має розширення відмова доступу, що при невірних даних авторизації забороняє доступ до системи.

Однією з основних причин розробки системи авторизації є захист конфіденційної інформації користувача. За допомогою авторизації можна забезпечити обмежений доступ до особистих даних, таких як ім'я, адреса електронної пошти, пароль і інші особисті дані. Це дозволяє уникнути несанкціонованого доступу та зловживання інформацією користувача.

Крім того, система авторизації допомагає забезпечити безпеку функціоналу додатку. Шляхом обмеження доступу до певних функцій та ресурсів на основі авторизаційних привілеїв можна попередити несанкціоновану зміну даних, виконання небажаних операцій або втрату конфіденційної інформації.

Додатково, система авторизації дозволяє керувати правами доступу для різних користувачів або груп користувачів. Це може бути особливо важливо в майбутньому, у разі масштабування системи, в результаті якої з'явиться

можливість спільної роботи, де потрібно різним користувачам надавати різні рівні доступу та функціоналу.

Задля забезпечення зручності процесу аутентифікації, в системі необхідно передбачити авторизацію через обліковий запис Google, що звільнить кінцевого користувача від необхідності вводу логіну та паролю, а також автоматичну авторизацію, за допомогою якої можна автоматично входити до системи без необхідності введення своїх облікових даних кожного разу при відвідуванні додатку. Цей механізм забезпечує зручність і зберігає час для користувача, оскільки він може уникнути повторного введення своїх ідентифікаційних даних.

2.3.2 Визначення вимог до логічного устрою бази даних

СКБД має задовольняти вимогам цілісності та безпеки даних. Усі сутності системи мають бути чітко пов'язані один з одним і не має бути жодних логічних помилок.

СКБД має витримувати часті стандартні операції щодо перегляду, додавання, редагування та видалення даних, що зв'язані з предметною областю системи.

2.3.3 Зовнішні інтерфейси

2.3.3.1 Визначення вимог до інтерфейсу програми

Для взаємодії з системою користувач має мати графічний інтерфейс, який буде задовольняти його потребам та вимогам.

Графічний інтерфейс (UI – user interface) – компоненти інтерактивної системи (програмне або апаратне забезпечення), які надають інформацію та засоби керування для виконання користувачем конкретних завдань з інтерактивною системою; комплекс програмно-апаратних засобів, що дозволяє користувачеві взаємодіяти з комп'ютерною системою [17].

Проектування UI характеризується створенням інтуїтивно зрозумілого інтерфейсу робочого типу програми без етапів програмування. Дизайн - це те, що бачить кінцевий користувач системи, і саме він відповідає за перше враження від програми. Він має бути мінімалістичним та інтуїтивно-зрозумілим, а також мати властивість адаптивності, що дозволить підлаштовуватися під розмір будь-якого

пристрою, не втрачаючи інформації та доступності елементів. Для забезпечення максимальної зручності користувача під час взаємодії з програмою на мобільному пристрої, необхідно враховувати специфіку екранів OLED, які використовуються на більшості сучасних смартфонів. Для цього сьогодні активно використовується темна тема, що полегшує сприйняття інформації в умовах недостатнього освітлення і зменшує споживання заряду батареї. Під час розробки важливо забезпечити легкий доступ до всіх функцій додатку та не перевантажувати інтерфейс складними елементами, що може призвести до зниження швидкості його роботи.

2.3.3.2 Функціональність інтерфейсу мобільного додатку

Програмний продукт виконує наступні функції:

ФІМД-1. Авторизація. Після відкриття додатку необхідно натиснути на кнопки “Continue with Google”, та обрати відповідний обліковий запис, дані якого будуть використовуватися для ідентифікації користувача в додатку. Після успішного входу, або автоматичної авторизації, він потрапляє на головний екран з мапою.

ФІМД-2. Показати моє поточне місце розташування. На головному екрані користувач пересував карту. Після натискання на відповідну кнопку, мапа показує поточну локацію користувача.

ФІМД-3. “Шари” мапи. Користувач на головному екрані, натискає на UI-елемент "Шари", який розташовується над мапою. Розкривається UI-елемент, з вибором доступних шарів і видів карти (супутник, рельєф, пробки і т.д.)

ФІМД-4. Пошук місць. Користувач на головному екрані натискає на пошуковий рядок. Відкривається екран пошуку місць, з можливістю фільтрацією елементів за назвою.

ФІМД-5. Прокладання маршруту до обраного місця. Після обирання локації з екрану пошуку місць, на головному екрані з мапою відмічається обране місце та показується меню з інформацією щодо нього та кнопкою навігації, після натискання якої карта в масштабі відображає повний маршрут до точки. Якщо

маршрут влаштовує користувача, він має можливість перейти у спеціальний режим навігації.

ФІМД-6. Навігація до обраного місця. Додаток з мапою може використовувати GPS або інтернет для визначення поточного положення користувача на мапі. Це дозволяє показувати його точне розташування та використовувати цю інформацію для подальшої навігації, під час якої камера за замовченням слідує за рухом водія. Окрім цього, під час навігації, додаток може надавати користувачеві постійні вказівки для керування на маршруті. Це включає повороти, рух по конкретній дорозі, інформацію про відстань до наступного повороту та приблизний час прибуття до пункту призначення.

ФІМД-7. Перегляд мого профілю. Користувач на головному екрані натискає на UI-кнопку “Profile”. Відкривається екран профілю, де відображається ім'я, статус онлайн/офлайн, геолокація, що включає місто або країну актуального розташування, та кнопки для перегляду клубів користувача і виходу з облікового запису.

ФІМД-8. Перегляд моїх клубів. Користувач на екрані свого облікового запису, переходить у вкладку “My Clubs”, після чого відображається екран списку клубів, до яких належить користувач. Список розділяється на дві категорії – клуби, до яких він був запрошений як учасник, і клуби, які він створив.

ФІМД-9. Створення клубу. Користувач на екрані свого облікового запису, переходить у вкладку “My Clubs”, та натискає “Create Club”. Відкривається вікно створення клубу, в якому користувач може задати назву клубу, його опис, фотографію, перемикач відкритий/закритий клуб, який впливає на можливість додавання нових членів іншими учасниками, що не є адміністраторами. Творець клубу автоматично є адміністратором клубу, і може бути тільки один.

ФІМД-10. Додавання учасників. Після створення клубу, чи на сторінці клубу, адміністратор, або інший учасник, в залежності від типу клубу, мають можливість додавання нових учасників зі списку всіх користувачів додатку.

ФІМД-11. Редагування клубу. Адміністратор клубу у вкладці "My Clubs", натискає на контекстне меню клубу, вибирає пункт "Edit". Відкривається вікно редагування інформації про клуб, ідентичне вікну створення клубу (ФІМД-9).

ФІМД-12. Чат клубу. Перебуваючи у вкладці "My Clubs", користувач натискає на відповідний клуб, та потрапляє у його чат, що являє собою вікно для спілкування користувачів. Усі учасники клубу можуть відправляти повідомлення у чат, що окрім самого тексту відображає дату відправки.

ФІМД-13. Створення заходу клубу. Перебуваючи на сторінці клубу, адміністратор натискає на кнопку "Create Club Event". Відкривається екран карти з вибором точки. Після підтвердження точки з'являється вікно інформації про захід, в якому заповнюється назва заходу, дата/час та короткий опис. Створений захід відображається на карті лише в учасників клубу.

ФІМД-13. Створення відкритого заходу. Будь-який користувач на карті використовує жест "long press" на бажану точку, у впливаючому вікні вибирає "Create Event", після чого з'являється сторінка заповнення інформації про захід, ідентична створенню заходу клубу (ФІМД-13). Створений відкритий захід відображається усім користувачам додатку, незалежно від участі у клубах.

2.4 Моделювання об'єктів системи

Згідно з описом функціональності інтерфейсу мобільного додатку, у системі можна виділити наступні сутності: користувач, клуб, захід, чат, повідомлення.

Користувач:

- id;
- ім'я;
- фамілія;
- email;
- локація;

- isOnline;

Клуб:

- id клубу;

- назва;

- опис;

- URL зображення клубу;

- isPublic;

- учасники;

- адмін;

- id чату;

Захід:

- id заходу;

- назва;

- опис;

- час;

- широта;

- довгота;

- id клубу;

- id творця;

Чат:

- id;

- учасники;

Повідомлення:

- id повідомлення;

- id чату;

- id автора;

- текст;

- час;

- authorIsMe;

2.5 Опис архітектури системи

Розроблювана розподілена система складається з двох частин – серверної та клієнтської. У якості серверу використовується платформа для розробки мобільних та веб-додатків, розроблена компанією Google – Firebase.

Роль бази даних виконує Firebase Firestore - хмарна БД, яка забезпечує зручне зберігання та синхронізацію даних у реальному часі між різними пристроями та платформами. Firestore підтримує структуровані дані у вигляді колекцій та документів, що дозволяє розробникам легко організовувати та отримувати дані з бази даних.

Схематичне спрощене представлення архітектури розроблюваного додатку, що активно просувається компанією Google, приведено на рис. 2.1.

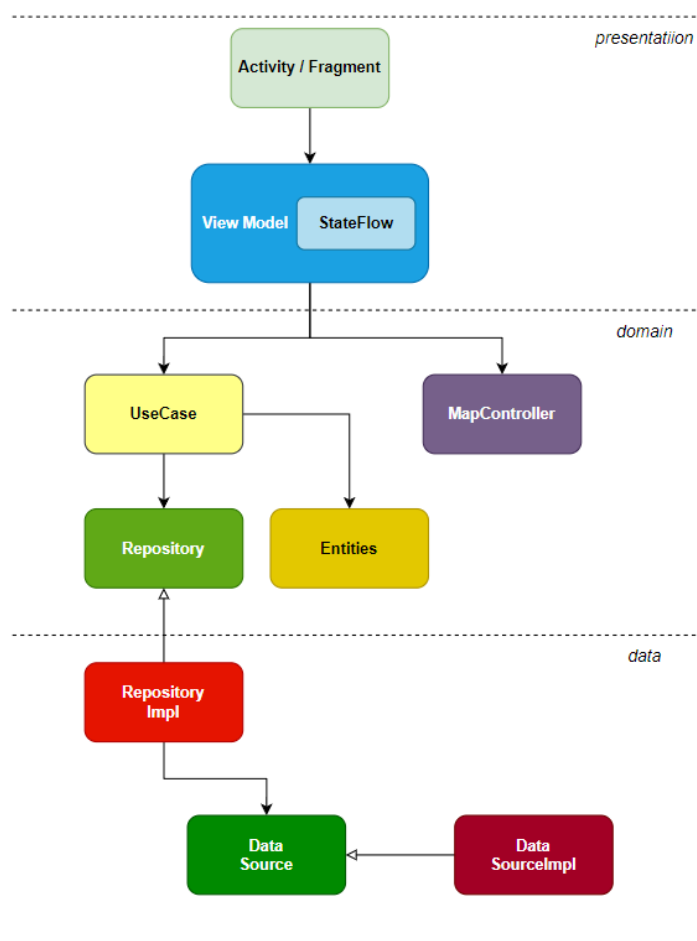


Рисунок 2.1 - Схема архітектури Android додатку

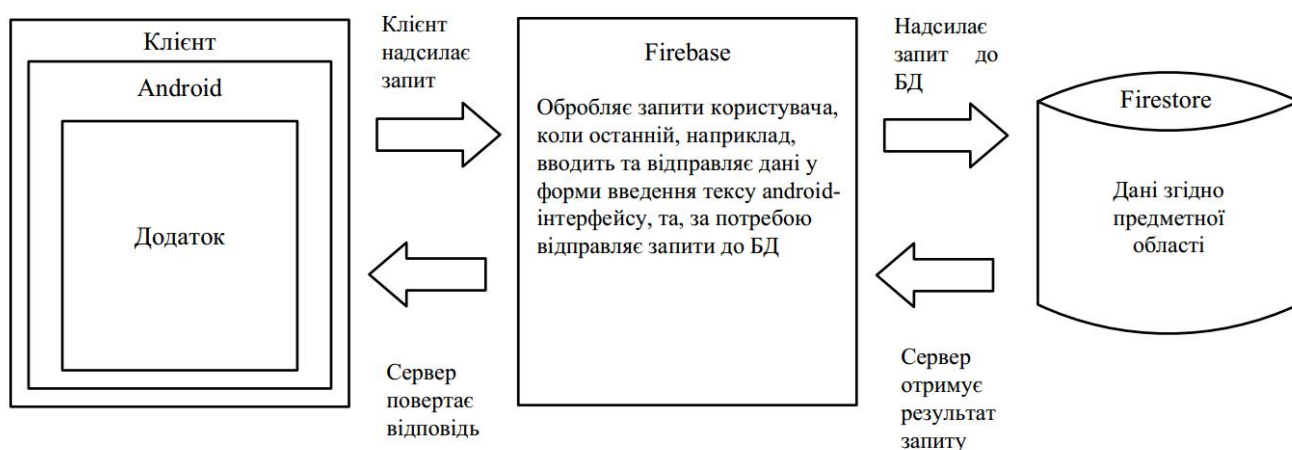


Рисунок 2.2 – Схема розподіленої системи

Архітектурні шари Android - це організація коду та відповідальностей в процесі розробки додатків. Зазвичай вони включають презентаційний (presentation), доменний (domain) та шар даних (data). Кожен з цих шарів має свої функції та обов'язки, а їх взаємодія дозволяє побудувати ефективну та масштабовану архітектуру.

Презентаційний шар (presentation): Цей шар відповідає за представлення даних користувачу та взаємодію з ним через інтерфейс користувача (UI). Він включає активності, фрагменти, адаптери, представлення та інші компоненти, які відображають дані користувачу та обробляють його дії. Презентаційний шар не повинен містити бізнес-логіки, його основна відповідальність - це перетворення та відображення даних, отриманих з доменного шару.

Доменний шар (domain): Доменний шар представляє бізнес-логіку додатку та правила його функціонування. Він включає моделі даних (Entities), інтерфейси репозиторіїв, та використовується для визначення взаємодії між презентаційним та шаром даних. Доменний шар не залежить від реалізації платформи та зовнішніх джерел даних, його завдання - забезпечити однорідну бізнес-логіку, що не залежить від інфраструктури.

Шар даних (data): Шар даних відповідає за доступ до даних, збереження та отримання даних з різних джерел, таких як бази даних, веб-сервіси, кешування

тощо. Він включає реалізацію репозиторіїв, локальну та віддалену сховища даних, API-клієнтів та інші компоненти, які забезпечують доступ до даних. Шар даних використовується доменним шаром для отримання та збереження даних, а також для виконання операцій з даними, таких як запити до бази даних або взаємодія з веб-сервісами.

Розглянемо компоненти з схематичного представлення архітектури Android додатку, що представлені на рисунку 2.2, детальніше.

2.5.1 Activity та Fragment

Activity та Fragment є частиною шару View (представлення), а це означає, що вони не повинні мати нічого спільного з бізнес-логікою та/або складними процесами в додатку. View лише відповідає за взаємодію між користувачем і додатком, аналізуючи і спостерігаючи за цим процесом.

2.5.2 ViewModel

ViewModel це один з найважливіших компонентів архітектури, тому що саме цей компонент містить у собі інформацію про стан інтерфейсу користувача, також цей компонент зберігає «цілісність» інтерфейсу при зміні в конфігурації, наприклад при повороті екрану. ViewModel виконує роль посередника між View (користувацьким інтерфейсом) та Model (даними та бізнес-логікою). Він зберігає стан візуальної частини додатку та надає методи для взаємодії з моделлю даних. ViewModel не повинен залежати від конкретних даних або джерела даних, але він може використовувати UseCase для отримання необхідних даних. На кожен фрагмент у додатку існує своя окрема ViewModel, яка зберігає стан конкретного екрана.

StateFlow є одним зі стрімів (Flow) в Kotlin та бібліотеці Kotlin Coroutines. Він використовується для представлення потоку стану або змінних, які можуть змінюватися з часом. StateFlow пропонує розробникам зручний спосіб оновлення та спостереження за станом додатку.

У контексті ViewModel, StateFlow може бути використаний для представлення стану UI та передачі змін у реактивному стилі. ViewModel може мати декілька StateFlow полів, які представляють різні аспекти стану додатку. Коли

стан змінюється, ViewModel відправляє нове значення через StateFlow, і підписані на нього спостерігачі (наприклад, фрагменти або активності) отримують це оновлення та відображають відповідні зміни в UI.

2.5.3 UseCase

UseCase представляє сценарій використання або конкретну операцію, яку виконує додаток. Він містить бізнес-логіку та робить запити до репозиторіїв або джерел даних для отримання та обробки даних. UseCase може включати такі операції, як отримання списку елементів, додавання нового елемента, оновлення даних тощо. Він повинен бути незалежним від фреймворків та зовнішніх ресурсів, і виконувати лише бізнес-логіку.

2.5.4 MapController

MapController - це спеціальний клас-обертка, який реалізує паттерн проектування “Фасад” для роботи з мапою всередині додатку. Паттерн фасад (Facade pattern) у об'єктно-орієнтованому програмуванні (ООП) є структурним патерном проектування, який надає простий уніфікований інтерфейс до складної системи класів, бібліотеки або підсистеми. Він дозволяє приховати складність та внутрішню структуру системи, надаючи більш простий та зручний інтерфейс для взаємодії з нею. Основна мета патерна фасад - спростити використання складної системи, надаючи єдину точку входу та приховуючи деталі її роботи. Фасад надає клієнтам спрощений інтерфейс, який абстрагує складність внутрішньої системи та надає лише необхідний функціонал для досягнення конкретних цілей [19].

MapController інкапсулює методи та функціональність, пов'язану з роботою з мапою, незалежно від конкретної бібліотеки, що використовується для відображення картографічних даних. MapController дозволяє виконувати різноманітні операції з мапою, такі як відображення маркерів, маршрутів, полігонів, зумування та перетягування карти, а також отримання координат місць на карті та інше. Він забезпечує зручний та єдинообразний інтерфейс для взаємодії з мапою, що дозволяє розробникам просто та зрозуміло використовувати функціональність мапи без необхідності знати деталі конкретної реалізації.

Один з ключових принципів, що реалізований у MapController, це розділення бізнес-логіки додатку від деталей реалізації мапи. Він дозволяє розробникам працювати з мапою на вищому рівні абстракції, не залежно від конкретного сервісу мапування. Це спрощує розробку та підтримку додатку, оскільки зміна або підключення нової бібліотеки для роботи з мапою вимагатиме лише змін у MapController, зберігаючи при цьому решту додатку незмінним.

Крім того, використання MapController дозволяє досягти більшої читабельності та суттєво спростити код, пов'язаний з роботою з мапою. Розробники можуть використовувати методи та функції MapController для взаємодії з мапою, замість прямого виклику методів бібліотеки мапування. Це дозволяє уникнути дублювання коду та спрощує обслуговування та зміни в майбутньому.

2.5.5 Repository

Repository є одним з ключових компонентів архітектури Clean Architecture та використовується для розділення бізнес-логіки додатку від джерела даних. У шарі domain, Repository визначає контракт (інтерфейс) для доступу до даних та визначає необхідні операції для отримання, збереження та оновлення даних.

У шарі data, реалізація Repository (Repository Implementation) забезпечує конкретну реалізацію методів, описаних в контракті Repository. Вона взаємодіє з різними джерелами даних, такими як локальна база даних, веб-сервер або зовнішні API, і виконує конкретні операції з даними, необхідні для виконання бізнес-логіки.

Використання Repository дозволяє забезпечити чистоту доменної логіки, зберігати її незалежною від конкретного джерела даних та спрощує тестування та підтримку додатку. Він створює прошарок, що відокремлює шар доменної логіки від деталей реалізації та дозволяє змінювати або додавати нові джерела даних без зміни логіки додатку.

2.5.6 DataSource

Компонент DataSource є частиною шару даних (data layer) в архітектурі додатку і відповідає за доступ до конкретного джерела даних, такого як база даних, веб-сервіс або кеш. Він визначає контракт, абстракцію або інтерфейс, який описує методи для отримання, збереження, оновлення та видалення даних.

`DataSource` визначає загальний набір операцій, які можна виконати над джерелом даних. Наприклад, це може бути метод для отримання списку користувачів, метод для додавання нового запису або метод для оновлення існуючого запису. Цей компонент може також визначати фільтри, параметри пошуку або сортування, які можна застосовувати до даних.

`DataSource Implementation` є конкретною реалізацією `DataSource` і виконує фактичну роботу з конкретним джерелом даних. Наприклад, якщо джерелом даних є база даних `SQLite`, `DataSource Implementation` буде містити код для взаємодії з базою даних, виконання `SQL`-запитів та отримання результатів.

2.6 Огляд бібліотеки для роботи з мапою

Для того щоб реалізувати роботу з мапою, пошук місць та режим навігації, до проекту були підключені бібліотеки `Navigation SDK` та `Search SDK` від компанії `Mapbox`. `Mapbox SDK` на `Android` - це набір інструментів та бібліотек, які дозволяють розробникам інтегрувати і використовувати функціонал картографії та мапових сервісів у `Android`-додатках. За допомогою `Mapbox SDK` можна створювати картографічні додатки, відтворювати та візуалізувати карту, додавати маркери, шари, маршрути та інші елементи на мапу, а також працювати з геолокацією та навігацією.

`Mapbox Navigation SDK` - це спеціальний набір інструментів і бібліотек від `Mapbox`, які надають розробникам можливості для реалізації навігаційних функцій. За допомогою `Mapbox Navigation SDK` можна створювати додатки з голосовим навігаційним інтерфейсом, побудовою маршрутів, відображенням прогнозованого часу прибуття, повідомленнями про повороти, швидкістю руху, розрахунком оптимального маршруту та багато іншого. Це дозволяє створювати навігаційні додатки з високою точністю та гарним користувацьким досвідом.

Mapbox Search SDK - це інструментарій, наданий Mapbox, який дозволяє інтегрувати можливості пошуку місць та адрес у Android-додатках. За допомогою Mapbox Search SDK розробники можуть додати функціонал пошуку місць, таких як ресторани, готелі, магазини, вулиці, а також отримувати інформацію про них, включаючи географічні координати, адреси, категорії, та інші атрибути. Mapbox Search SDK надає можливості автозаповнення пошукових запитів, фільтрації результатів пошуку, відображення списків пропозицій, підказок та рейтингів.

2.7 Опис структури системи

Згідно з представленою архітектурою на рис. 2.2, код клієнтської частини додатку, що складається з великої кількості класів, розділяється на пакети `presentation`, `domain` та `data`, а також додаткові `core`, `di` та `utils`, що містять базові абстрактні класи для фрагментів, `view`-моделей, `use-кейсів`, модулів ін'єкції залежностей, та інших допоміжних інструментів.

Схема основного функціоналу додатку приведена на рис. 2.3. Цикл взаємодії користувача з системою починається з `MainActivity.kt`. Використовуючи принцип `Single Activity`, у проекті головна активність бере на себе роль своєрідного “хоста” для фрагментів, які відображаються в залежності від певних умов. Під час запуску додатку активність підписується на зміни статусу авторизації користувача. При отриманні змін, виконується перевірка, чи користувач має авторизований статус. Якщо так, користувач потрапляє на головну сторінку з мапою. В іншому випадку відображається сторінка авторизації `LoginFragment.kt`, на якій доступна кнопка для аутентифікації через Google, після натискання на яку відображається анімація завантаження. У разі помилки під час відправлення даних користувача на сервер буде показано повідомлення з причиною помилки. Після успішної авторизації дані користувача, що беруться з облікового запису Google, зберігаються у БД на сервері, та додаток перенаправляє клієнта на `MapFragment.kt`. Якщо під час

користування додатком клієнт вийшов зі свого облікового запису, це автоматично змінить статус авторизації, на що зреагує `MainActivity`, почистить стек попередніх фрагментів, та знову відобразить сторінку авторизації.

Після того, як користувач потрапить на головну сторінку з мапою, окрім взаємодії з картою, йому буде надана можливість переглядати свій профіль, змінювати зовнішній вигляд мапи, шукати локації, прокладати до них маршрути, переходити у режим навігації до цих маршрутів, а також створювати відкриті заходи.

Зміна зовнішнього вигляду карти здійснюється за допомогою спеціальної кнопки над мапою, натискання по якій показує компонент `MapStylesBottomSheet`, реалізований за допомогою `BottomSheetDialogFragment`, з усіма можливими стилями мапи, такими як темна і світла теми, вулиці, супутник, та інші. Для наочності стилі візуально представлені у вигляді зображень, а навколо зображення обраного стилю відображається кольорова рамка. Після натискання на відповідний стиль, карта змінюється в реальному часі.

Навігація між фрагментами, зокрема перехід на попередній фрагмент, здійснюється за допомогою кнопок у тулбарі. Тулбар (Toolbar) в Android - це важливий компонент користувацького інтерфейсу, який надає зручні можливості для розміщення та управління функціями додатку. Він представляє собою панель, яка зазвичай розташовується у верхній частині екрана та містить набір інтерактивних елементів, таких як кнопки, текстові поля або іконки.

У додатку, що розроблюється в даній дипломній роботі, тулбар містить кнопку для переходу на попередній фрагмент, та додаткову кнопку дії, яка в залежності від екрану може виконувати певні дії.

Процес створення відкритого заходу починається з того, що користувач затискає палець у певному місці на карті – цей жест має назву “Long press”. Фрагмент з мапою оброблює цей жест і показує діалог, що відображає адресу обраного місця, та кнопку для створення заходу у цьому місці. Після натискання на кнопку користувачу відображається фрагмент `CreateEventFragment.kt`, на якому можна заповнити дані заходу за допомогою текстових полів. Фрагмент та його поля

мають систему валідації, яка зробить кнопку для створення заходу активною тільки тоді, коли усі відповідні поля будуть заповнені. Якщо під час заповнення якесь з полів буде пустим, під ним відобразиться відповідне повідомлення про те, що воно не допускає порожній рядок. Створений захід буде відображатися на мапі за допомогою спеціального маркера, після натискання на який відкриється діалог з усією заповненою інформацією щодо заходу, та кнопкою для побудови маршруту до нього.

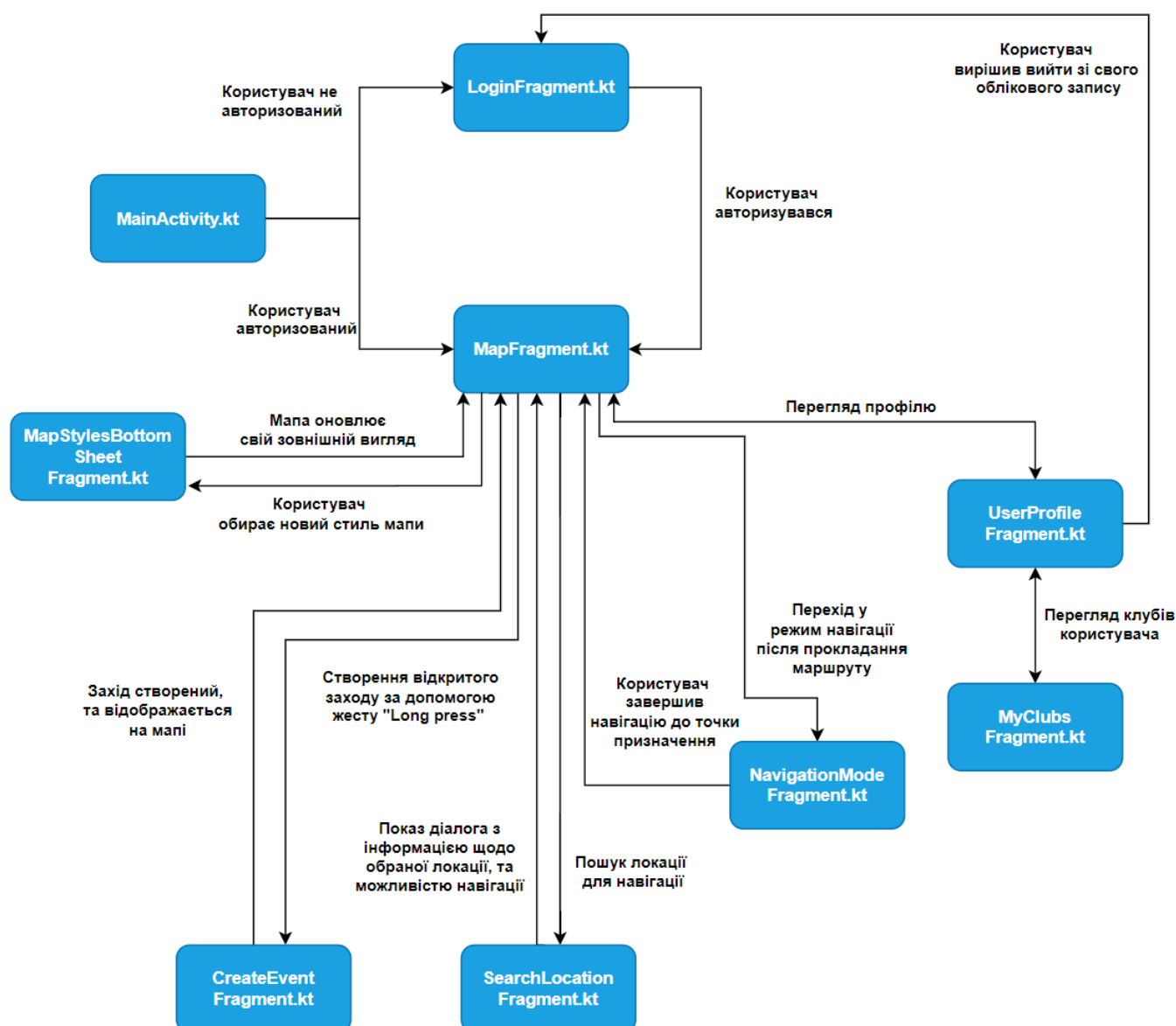


Рисунок 2.4 – Схема авторизації та взаємодії з мапою

Побудова маршруту являє собою візуальне прокладання лінії маршруту по дорозі на мапі від поточної локації до бажаної точки. Після створення відбувається зміна положення та зуму камери для того, щоб повністю відобразити маршрут на екрані пристрою, а також з'являється кнопка для переходу у режим навігації. Якщо створений маршрут не влаштовує користувача, за допомогою системної кнопки, або жесту “Назад” є можливість очистити мапу від прокладеного маршруту для подальшого використання карти. Якщо користувач спробує створити маршрут до точки, до якої неможливо доїхати з поточного місцезнаходження, додаток покаже відповідне повідомлення з помилкою.

Після переходу у режим навігації користувач потрапляє на `NavigationModeFragment.kt`, який цілком використовує навігаційні можливості з `Mapbox Navigation SDK`. Він містить наступні елементи:

Карта: вбудований режим навігації має інтерактивну карту, на якій відображаються поточне місцезнаходження користувача, маршрут, маркери і інші важливі елементи для навігації.

Дорожні знаки та підказки: візуальні підказки для користувача, такі як інформація про повороти, полоси руху, обмеження швидкості та інші дорожні знаки, що відображаються на верхній панелі.

Інформаційна панель: відображаються додаткові дані про поїздку, наприклад, залишкова відстань, час до прибуття, швидкість та інші важливі показники.

Елементи управління: кнопки паузи, відновлення, зупинки маршруту, переключення між режимами перегляду, збільшення та зменшення масштабу карти тощо.

Звукові підказки: сповіщають користувача про надходження поворотів, маневри, зміну напрямку або інші важливі події під час поїздки. Це допомагає користувачеві зосередитися на дорозі, не відволікаючись на екран пристрою.

Режими перегляду: перегляд зверху, перегляд траси, перегляд 3D тощо. Це дозволяє користувачеві вибрати найзручніший спосіб відображення маршруту на карті.

Після завершення поїздки, про яку користувача сповістить інформаційна панель, навігація до головного екрану з мапою здійснюється за допомогою кнопки виходу, що завжди доступна на інформаційній панелі.

На MapFragment.kt також передбачена можливість перейти на сторінку з інформацією щодо особистого профілю. На ній можна побачити аватар, який складається з ініціалів користувача, статус онлайн / оффлайн, ім'я, останню локацію, в якій перебував користувач, кнопку для виходу з облікового запису та кнопку для навігації на MyClubsFragment.kt.

На рис. 2.5 наведено схему використання соціальних функцій Android-додатку, які включають в себе створення та участь у клубах, спілкування у чатах, а також участь у спільних заходах клубу.

MyClubsFragment.kt містить кнопку для створення нового клубу та список клубів, у верхній частині якого відображаються клуби, де користувач – творець, а у нижній – де користувач тільки учасник. Верхня та нижня частини списку розділяються текстовими заголовками.

Після вибору елементу зі списку клубів, користувач автоматично переходить на фрагмент ChatFragment.kt. На цьому фрагменті відображається чат клубу, який представляє собою список повідомлень. Повідомлення в чаті групуються за джерелом: повідомлення від користувача та повідомлення від інших учасників. Крім того, повідомлення відокремлюються по днях за датою їх відправлення, щоб зробити чат більш структурованим та зручним у сприйнятті. У нижній частині екрану розташований рядок для введення тексту, який динамічно підлаштовується під його розмір. Якщо рядок не пустий, справа від нього з'являється кнопка для відправки повідомлення.

За допомогою кнопки у тулбарі ChatFragment.kt користувач може перейти на інформаційну сторінку обраного клубу. На ній відображається зображення клубу, тип (відкритий / закритий), назва, кількість учасників, опис, та список учасників. Якщо користувач є адміністратором клубу, для нього на сторінці відображаються кнопки редагування клубу, створення заходу клубу, та додавання нових учасників. Варто зазначити, що у випадку, коли користувач є тільки

учасником відкритого клубу, у нього також є можливість додавати нових учасників за допомогою відповідної кнопки.

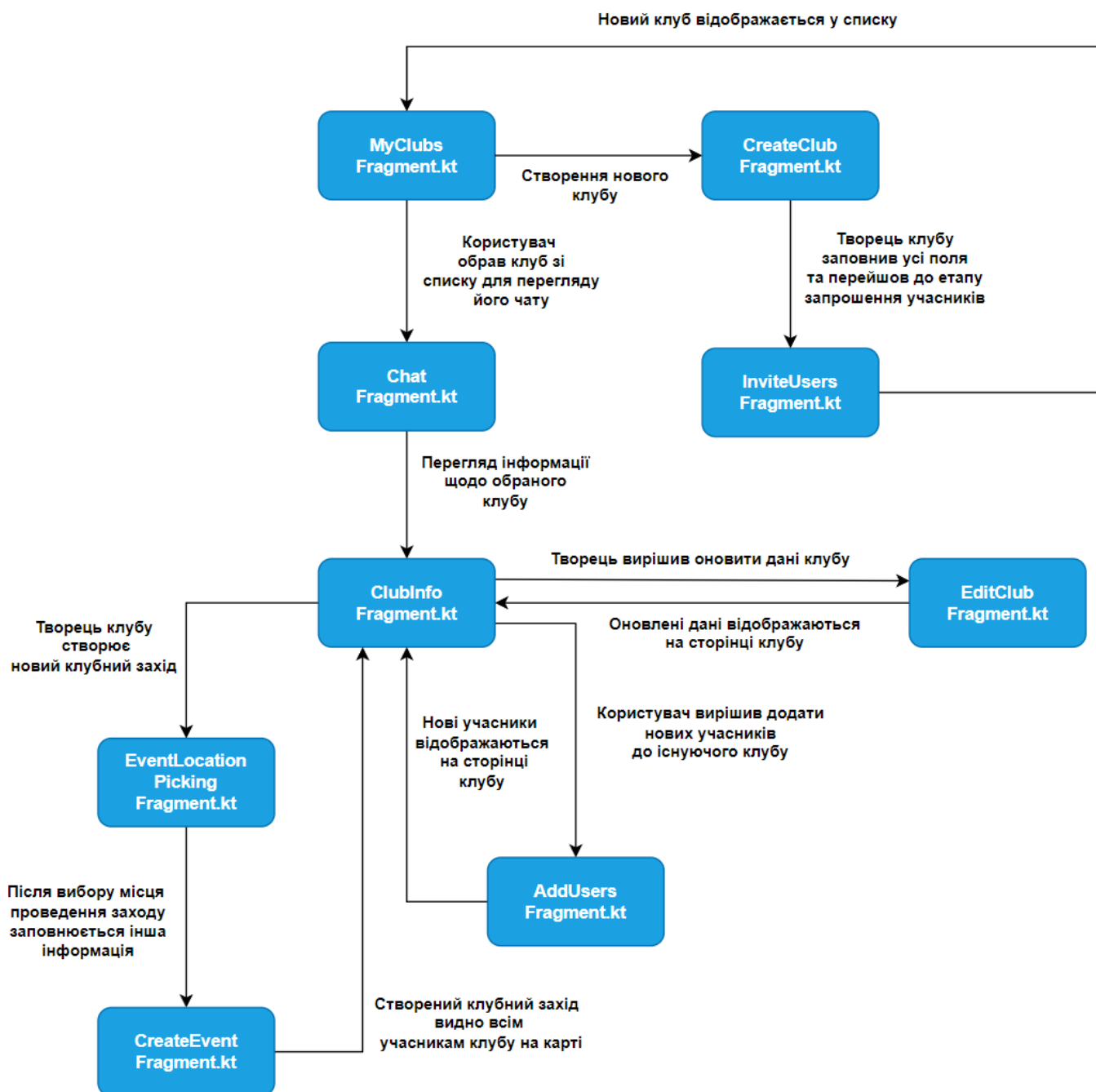


Рисунок 2.5 – Схема використання соціальних функцій додатку

Процес створення заходу клубу починається з EventLocationPickingFragment.kt, де за допомогою окремої карти, що ніяк не пов'язана з головним екраном MapFragment.kt, користувач, пересуваючи літаючий

над картою маркер, який виконує роль своєрідну роль прицілу, обирає локацію для майбутнього заходу. Для зручності використання такого способу над кнопкою для підтвердження локації знаходиться вікно з адресою місця, деталізація якої може змінюватися в залежності від розташування локації. Якщо під час конвертації координат на мапі в адресу відбудеться помилка, у вікні відобразиться повідомлення “Address not found”. Після вибору місця майбутнього заходу, користувач потрапляє на `CreateEventFragment.kt`, та заповнює усі поля аналогічно процесу створення відкритого заходу.

Користувач, як адміністратор свого клубу, має можливість перейти на `AddUsersFragment.kt` з інформаційної сторінки `ClubInfoFragment.kt`, та побачити список усіх наявних користувачів системи. Елемент списку представлений у вигляді аватара з ініціалами, ім'ям користувача, та спеціального UI-елементу “Checkbox”, за допомогою якого здійснюється вибір майбутніх учасників клубу. Після підтвердження обраних людей користувач потрапляє назад на сторінку клубу, де може побачити доданих байкерів.

Для наочності змодельємо процес створення нового клубу на `CreateClubFragment.kt` за допомогою UML-діаграми.

UML (Unified Modeling Language) – це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML-моделлю. UML був створений для визначення, візуалізації, проєктування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація [7].

Розроблена UML-діаграма створення нового клубу представлена на рис. 2.5.

Для того, щоб почати працювати з додатком, користувачу необхідно авторизуватися в системі. Після вибору облікового запису профіль користувача, що

створюється з даних Google-акаунту, відправляються на сервери Firebase, де у разі успішної валідації та аутентифікації зберігається у базі даних.

Після успішної авторизації профіль зберігається у кеші додатку для його подальшого використання під час активної сесії, та автоматично виконується навігація на головний екран, з якого через інші екрани користувач потрапляє на CreateClubFragment.kt.

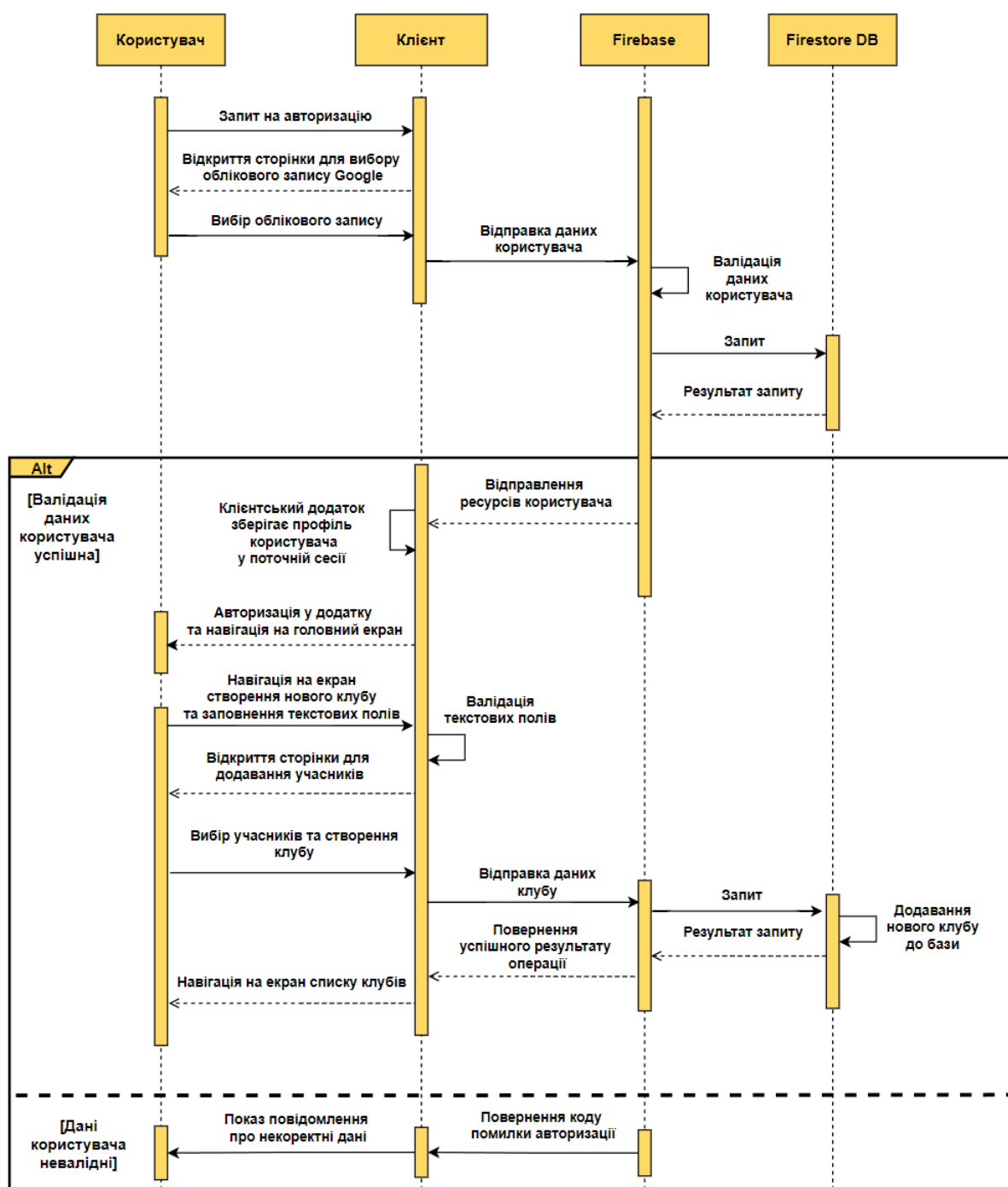


Рисунок 2.5 – UML-діаграма створення нового клубу

Як і у випадку з `CreateEventFragment.kt`, фрагмент для створення клубу має систему валідації текстових полів, що не дозволить створити клуб без їх заповнення. Окрім цього, для створення клубу передбачено опціональне додавання його зображення. По натисканню на відповідну `ImageView` показується діалог для вибору зображення з галереї або камери за допомогою вбудованих інструментів Android SDK. Обране зображення завантажується в `ImageView` та показується користувачу. За допомогою UI-елементу `Switch`, що програмно реалізований як `Custom View`, відбувається вибір типу клубу – відкритий або закритий. Після успішної валідації полів, користувач натискає на кнопку продовження, та одразу додає перших учасників майбутнього клубу на `InviteUsersFragment.kt` за аналогією з `AddUsersFragment.kt`. Після додавання дані клубу відправляються на сервер Firebase, де зберігаються у БД, а користувач автоматично повертається до `MyClubsFragment.kt`.

2.8 Розробка модулів клієнтської частини

2.8.1 Опис основних класів системи

Робота додатку починається з класу `MainActivitiy.kt`, що приведений у лістингу 2.1. У методі `onCreate()`: викликається метод `installSplashScreen()`, який налаштовує екран-заставку або сплеш-екран для додатку, що показується під час завантаження користувальницьких даних додатку. В проекті для інкапсуляції навігаційної логіки, в незалежності від бібліотеки або інструментів, був створений інтерфейс `Router`, який за допомогою методу `attachTo(this)` здійснює прикріплення до поточної активності. Його створення, як і `MainViewModel`, оголошене у DI-модулі бібліотеки `Koin`, яка автоматично створює та надає екземпляр класу на цьому екрані за допомогою особливості мови Kotlin - делегату `by`. У методах `onStart()` та `onStop()` викликається `viewModel.updateUserProfileOnlineStatus()`, який

оновлює статус онлайн присутності користувача в залежності від стану активності. При вході у додаток стан встановлюється значення true, а при виході - false.

Лістинг 2.1 - Код MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private val viewModel: MainViewModel by viewModel()
    private val mainRouter: Router by inject()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        installSplashScreen().apply {
            setKeepOnScreenCondition {
                viewModel.isLoading.value
            }
        }
        setContentView(R.layout.activity_main)
        mainRouter.attachTo(this)
    }

    override fun onCreateView(name: String, context: Context,
        attrs: AttributeSet): View? {
        lifecycleScope.launch {
            repeatOnLifecycle(Lifecycle.State.STARTED) {
                viewModel.error.collectLatest {
                    Toast.makeText(this@MainActivity,
                        it.message, Toast.LENGTH_SHORT).show()
                }
            }
        }
        return super.onCreateView(name, context, attrs)
    }

    override fun onStart() {
        super.onStart()
        viewModel.updateUserProfileOnlineStatus(isOnline =
true)
    }

    override fun onStop() {
        super.onStop()
        viewModel.updateUserProfileOnlineStatus(isOnline =
false)
    }
}
```

Уся головна логіка активності прописана у класі `MainViewModel.kt`, що дозволяє виконувати операції авторизації, оновлення статусу користувача та навігації між екранами відповідно до бізнес-логіки додатку. Коли користувач запускає додаток, він підписується на статус авторизації за допомогою `subscribeOnUserAuthStatusUseCase`. Якщо користувач авторизований, викликається `loadUserProfileUseCase`, який завантажує профіль користувача. При успішному завершенні цього сценарію використання, відбувається навігація до головної карти за допомогою `navigateToMainMap()`. Якщо користувач неавторизований, зупиняється показ статусу завантаження та відбувається навігація до екрану входу за допомогою `navigateToLogin()`. Функція `updateUserProfileOnlineStatus()` викликає `updateUserProfileOnlineStatusUseCase`, щоб оновити статус користувача онлайн. Вона також включає обробники подій для показу та приховування завантажувача, а також обробки помилок.

Лістинг 2.2 - Код `MainViewModel.kt`

```
class MainViewModel(
    private val router: Router,
    private val subscribeOnUserAuthStatusUseCase:
        SubscribeOnUserAuthStatusUseCase,
    private val updateUserProfileOnlineStatusUseCase:
        UpdateUserProfileOnlineStatusUseCase,
    private val loadUserProfileUseCase: LoadUserProfileUseCase
) :
    BaseViewModel() {
    init {
        launch {
            subscribeOnUserAuthStatusUseCase.execute().collectLatest
            { isAuthorized -> if (isAuthorized) {
                loadUserProfileUseCase.execute {
                    onStart = {
                        startLoading()
                    }
                    onSuccess = { user ->
                        launch(Dispatchers.Main) {
                            navigateToMainMap()
                        }
                    }
                    onError = { causeError(it) }
                }
            } else {
                stopLoading()
            }
        }
    }
}
```

Продовження лістингу 2.2

```

        navigateToLogin()
    }
}
}
fun updateUserProfileOnlineStatus(isOnline: Boolean) {
    launch {
        updateUserProfileOnlineStatusUseCase.execute(isOnline)
    }
    {
        onStart = { startLoading() }
        onError = { causeError(it) }
        onTerminate = { stopLoading() }
    }
}
private fun navigateToLogin() {
    router.execute(Command.Navigate(LoginScreen()))
}

private fun navigateToMainMap() {
    router.execute(Command.Navigate(MapScreen()))
}
}

```

Для реалізації архітектурного компоненту UseCase був створений однойменний абстрактний клас, код якого приведений у лістингу 2.3. Він надає загальний шаблон для виконання дій з переданим параметром і поверненням результату. Основний метод `execute` приймає параметр `param`, який відповідає за вхідні дані для випадку використання, а також опціональний параметр `result` типу `Callback<Result>`, який може бути використаний для отримання результату виконання. У методі `execute` створюється інстанс `ResultCallback<Result>`, який відповідає за отримання результату та обробку помилок. За допомогою методу `apply(result)`, параметр `result` передається до `ResultCallback`, щоб викликати відповідні колбеки. У блоку `runCatching` виконується фактичне виконання випадку використання. Виклик функції `executeActual(param)` виконує реальну логіку сценарію використання з переданим параметром `param`.

Якщо виконання успішне (без винятків), результат передається до `resultReceiver.onSuccess?.invoke(success)`, де `success` - це отриманий результат.

Якщо виконання завершується з винятком, він обробляється за допомогою `mapException`, де `errorHandler.process(exception)` викликає обробник помилок `errorHandler`, що дозволяє мапувати винятки на відповідні помилки додатку.

Якщо виникає помилка, виконується `resultReceiver.onError?.invoke(mappedException)`, де `mappedException` - це мапована помилка.

У блоку `finally` виконується `resultReceiver.onTerminate?.invoke(isCanceled)`, де `isCanceled` - це прапорець, який вказує, чи було виконання скасоване.

Лістинг 2.3 - Абстрактний клас сценарію використання

```
abstract class UseCase<Param, Result>(  
    private val coroutineContext: CoroutineContext,  
    private val errorHandler: ErrorHandler,  
    ) {  
    protected abstract suspend fun executeActual(param: Param):  
    Result  
    @Suppress("TooGenericExceptionCaught")  
    suspend fun execute(  
        param: Param,  
        result: Callback<Result> = {},  
    ) {  
        val resultReceiver = ResultCallback<Result>()  
        .apply(result)  
        .also {  
            it.onStart?.invoke()  
        }  
        runCatching {  
            withContext(coroutineContext) {  
                executeActual(param) }  
            }.onSuccess { success ->  
                resultReceiver.onSuccess?.invoke(success)  
            }.mapException { exception ->  
                errorHandler.process(exception)  
            }.onFailed { mappedException ->  
                resultReceiver.onError?.invoke(mappedException)  
            }.finally { isCanceled ->  
                resultReceiver.onTerminate?.invoke(isCanceled)  
            }  
        }  
    }  
}
```

Розглянемо реалізацію сценарію використання на прикладі `updateUserProfileOnlineStatusUseCase` з параметрами та результатом типу `Boolean`.

пов'язаного з оновленням статусу онлайн-присутності користувача профілю. Залежності, такі як менеджер автентифікації і репозиторій користувача профілю, можуть бути надані через конструктор, що дозволяє забезпечити їх легку зміну.

У класі визначена функція `executeActual()`, яка перевизначає абстрактний метод `executeActual(param: Boolean)` в батьківському класі. У цьому методі виконується фактична логіка виконання випадку використання, яка включає оновлення статусу користувача профілю залежно від переданого параметра `param`. Спочатку перевіряється, чи автентифікований користувач шляхом перевірки значення, яке повертається з `authManager.subscribeOnUserAuthStatus()`. Якщо користувач автентифікований, викликається метод `updateUserProfileOnlineStatus(isOnline = param)` репозиторію `userProfileRepository` для оновлення статусу користувача профілю. Якщо користувач не автентифікований, повертається значення `false`.

Лістинг 2.4 - Сценарій використання `UpdateUserProfileOnlineStatusUseCase`

```
class UpdateUserProfileOnlineStatusUseCase(
    coroutineContext: CoroutineContext,
    errorHandler: ErrorHandler,
    private val authManager: AuthManager,
    private val userProfileRepository: UserProfileRepository
) : UseCase<Boolean, Boolean> {
    coroutineContext = coroutineContext,
    errorHandler = errorHandler,
} {
    override suspend fun executeActual(param: Boolean): Boolean {
        return if (authManager.subscribeOnUserAuthStatus().value) {
            userProfileRepository.updateUserProfileOnlineStatus(isOnline = param)
        } else {
            false
        }
    }
}
```

Однією з залежностей `UpdateUserProfileOnlineStatusUseCase` є інтерфейс `UserProfileRepository`, що описує контракт для репозиторію, який відповідає за роботу з профілем користувача. Цей репозиторій надає методи для завантаження, збереження та оновлення профілю користувача, а також для управління його станом. Реалізація `UserProfileRepositoryImpl` наведена у лістингу A.1 додатку A.

Лістинг 2.5 - Інтерфейс `UserProfileRepository`

```
interface UserProfileRepository {

    suspend fun loadUserProfile(): User

    suspend fun saveUserProfile(user: User): Boolean

    suspend fun updateUserProfileLocation(newLocation: Location): Boolean

    suspend fun updateUserProfileOnlineStatus(isOnline: Boolean): Boolean

    fun clearUserCache(): Boolean

    fun subscribeOnUserProfile(): StateFlow<User>
}
```

2.8.2 Опис модулю для роботи з мапою

Ключовим класом додатку, навколо якого побудовано функціональність програми, є `MapFragment.kt`. Під час найпершої авторизації у системі та навігації на цій фрагмент, додаток попросить видати `Location Permissions` - дозволи, які надаються додаткам для доступу до місцезнаходження пристрою. Вони контролюють доступ до сервісів геолокації, таких як GPS, Wi-Fi, мобільні мережі та інші, що дозволяють визначати поточну локацію користувача.

Механізм видачі дозволу на роботу з геолокацією в `MapFragment` приведений у лістингу 2.6, та полягає в перевірці наявності дозволу, відображенні діалогу запиту дозволу, обробці результату запиту та виконанні відповідних дій залежно від результату.

Лістинг 2.6 - Механізм видачі дозволів у MapFragment.kt

```

override fun onCreateView(view: View, savedInstanceState:
Bundle?) {
    super.onCreateView(view, savedInstanceState)
    ...
    checkLocationPermission()
    ...
}
private fun checkLocationPermission() {
    val permission = ContextCompat.checkSelfPermission(
        requireContext(),
        Manifest.permission.ACCESS_FINE_LOCATION)
    if (permission != PackageManager.PERMISSION_GRANTED) {
        requestPermissionLauncher.launch(Manifest.permission.ACCESS_FI
NE_LOCATION)
    } else {
        println("Permission isGranted")
        viewModel.loadMap()
        updateUserProfileLocation()
    }
}
private val requestPermissionLauncher =
registerForActivityResult(
    ActivityResultContracts.RequestPermission()
) { isGranted ->
    if (isGranted) {
        println("Permission isGranted in launcher")
        viewModel.loadMap()
        updateUserProfileLocation()
    } else {
        showMessageSnackBar(
            requireView(),
            getString(R.string.location_permission_explanation),
            isLong = true
        )
    }
}
}

```

Якщо дозволи були видані, то викликаються методи для завантаження мапи та оновлення місцезнаходження користувача. У разі відмови надати дозвіл, відображається повідомленням про необхідність надання доступів до місцезнаходження для подальшої роботи додатку.

Логіка, що відповідає за управління та взаємодією з відображенням мапи на екрані знаходиться у класі MapViewModel.kt. Його конструктор, поля, блок ініціалізації, та деякі методи наведені у лістингу А.2 додатку А.

MapViewModel відповідає за логіку та функціональність, пов'язані з відображенням та управлінням мапою на екрані. Основна мета цього класу полягає в відображенні різних елементів на мапі, таких як маркери подій, маркери місцезнаходження користувача, маршрути навігації тощо. Клас співпрацює з класом MapController, який відповідає за фізичну реалізацію мапи та його управління. Тут також використовуються різні інструменти та об'єкти, такі як Router для навігації між екранами для оновлення місцезнаходження користувача, отримання місцезнаходження за заданими координатами, підписки на події користувача, клубів, та їх заходів. Крім того, як і в інших вью-моделях, тут використовуються корутини для виконання асинхронних операцій, а саме запуску сценаріїв використання, оголошених у конструкторі, що можуть бути часово-витратними.

MapController є класом, відповідальним за управління мапою на екрані та забезпечення його відображення та функціональності. Отримуючи посилання на MapView, який є компонентом для відображення мапи у графічному інтерфейсі користувацького додатку, MapController може взаємодіяти з ним, виконуючи різні дії та налаштування:

а) ініціалізація та налаштування мапи:

- 1) встановлення стилю мапи;
- 2) ініціалізація та налаштування компонента місцезнаходження користувача;
- 3) додавання слухачів подій переміщення мапи та змін позиції та орієнтації місцезнаходження користувача;

б) управління маркерами на мапі:

- 1) відображення маркерів подій користувача та клубів;
- 2) відображення маркера вибору місця події;
- 3) очищення маркерів на мапі;

в) управління маршрутами навігації:

- 1) створення нового маршруту від поточного місцезнаходження користувача до заданого пункту призначення;

- 2) відображення маршрутної лінії на мапі;
- 3) очищення маршрутних ліній на мапі;
- г) управління камерою:
 - 1) відстеження місцезнаходження користувача на мапі та автоматичне оновлення камери;
 - 2) встановлення маркера для відображення користувача;
- д) інші допоміжні методи та функції:
 - 1) завантаження мапи;
 - 2) налаштування стилю мапи;
 - 3) відображення різних елементів мапи (маркери, маршрути);
 - 4) відображення та очищення маркера вибору місця події.

Враховуючи об'єм коду класу MapController.kt, у лістингу 2.8 як приклад наведено лише декілька методів контролеру.

Лістинг 2.7 - Методи MapController.kt

```
suspend fun createNewRoute(destination: Point, destinationName:
String?): Result<List<NavigationRoute>> {
    return suspendCoroutine { continuation ->
        val origin = userLocationCoordinates.value
        routeLinesManager.createNewRoute(
            origin = origin,
            destination = destination,
            destinationName = destinationName,
            navigationRouterCallback = object :
NavigationRouterCallback {
                override fun onRoutesReady(
                    routes: List<NavigationRoute>,
                    routerOrigin: RouterOrigin
                ) {
                    showRouteLines(routes = routes)
                }
            }
        )
        continuation.resume(Result.success(routes))
    }
    override fun onCancel(
        routeOptions: RouteOptions,
        routerOrigin: RouterOrigin
    ) {
        continuation.resume(Result.failure(Exception("Route
canceled")))
    }
    override fun onFailure(
        reasons: List<RouterFailure>,
```

Продовження лістингу 2.7

```

        routeOptions: RouteOptions
            ) {
continuation.resume(Result.failure(Exception(reasons.toString(
))))
            }
        },
    )
}

suspend fun showUsersEventsMarkers(events: List<Event>) {

    mapMarkersManager.showUsersEventsMarkers(events)

    fun showClubsEventsMarkers(events: List<Event>) {
        mapMarkersManager.showClubsEventsMarkers(events)
    }
    fun hasMarkers(): Boolean {
        return mapMarkersManager.hasMarkers
    }
    fun clearMarkers() {
        mapMarkersManager.clearMarkers()
    }
}

```

Метод `createNewRoute` використовується для створення нового маршруту від поточного місцезнаходження користувача до заданого пункту призначення. За допомогою `suspendCoroutine` створюється корутина, яка дозволяє призупинити виконання методу та продовжити його пізніше. Результат виконання методу повертається за допомогою об'єкту `Result<List<NavigationRoute>>`. Для обробки результату створення маршруту використовується `navigationRouterCallback`, який включає перевизначення трьох методів:

onRoutesReady: Викликається, коли маршрут готовий. При цьому відображаються маршрутні лінії за допомогою `showRouteLines`, а результат передається через `continuation.resume(Result.success(routes))`.

onCanceled: Викликається, якщо створення маршруту скасоване. Створюється виключення з повідомленням про скасування маршруту, і результат передається через `continuation.resume(Result.failure(Exception("Route canceled")))`.

onFailure: Викликається, якщо створення маршруту невдалий. Повідомлення про невдачу разом з причинами передається через `continuation.resume(Result.failure(Exception(reasons.toString())))`.

Метод `showUsersEventsMarkers` приймає список подій `events`, та виконується в асинхронному контексті (`suspend`), що означає, що він може бути викликаний з корутин. Це призначено для відображення маркерів відкритих подій, що створили користувачі на мапі.

Функція `showClubsEventsMarkers` необхідна для відображення маркерів подій, пов'язаних з клубами, на мапі.

Метод `hasMarkers` перевіряє, чи є на мапі присутні маркери. Він повертає значення `true`, якщо на мапі є маркери, або `false`, якщо маркерів немає. Це допомагає визначити наявність маркерів на мапі для подальшої обробки.

Метод `clearMarkers` видаляє всіх маркерів за допомогою функції `clearMarkers` об'єкта `mapMarkersManager`, що призводить до очищення всіх маркерів, які були розміщені на мапі.

2.8.3 Опис модулю для роботи з чатом

Гарним прикладом демонстрації архітектури розроблюваного додатку є модуль з чатом. Візуально чат являє собою перш за все список з повідомлень. В Android SDK список реалізується за допомогою `RecyclerView` - контейнеру, який відображає список елементів на екрані. Він управляє відображенням та перерендерингом елементів списку за допомогою повторного використання "віджетів" (`ViewHolder`). `RecyclerView` також відповідає за керування прокруткою та лаяутом елементів.

Реалізація списку на `RecyclerView` на Android складається з декількох основних компонентів:

`Adapter` - клас, який забезпечує зв'язок між даними та `RecyclerView`. Він відповідає за створення та управління віджетами (`ViewHolder`) для кожного елемента списку, а також за заповнення даними цих віджетів. `Adapter` також відслідковує зміни в даних та повідомляє `RecyclerView` про необхідність оновлення.

ViewHolder - клас, який представляє віджет одного елемента списку. Він містить посилання на різні віджети, такі як **TextView**, **ImageView** тощо, які відображають дані цього елемента. **ViewHolder** також може мати додаткові методи для взаємодії з віджетами, наприклад, обробник подій кліку. Реалізація цього класу забезпечує ефективне використання ресурсів шляхом повторного використання віджетів при прокрутці списку. Коли елемент списку стає видимим, **RecyclerView** передає йому **ViewHolder** зі збереженими посиланнями на віджети, що дозволяє уникнути зайвих операцій зі створенням та знаходженням віджетів на кожному скролі. Це забезпечує плавну прокрутку та швидке відображення великих списків даних.

LayoutManager - це клас, який визначає логіку розміщення елементів списку на екрані. Він відповідає за розташування та прокрутку елементів **RecyclerView**. Android постачає різні реалізації **LayoutManager**, такі як **LinearLayoutManager**, **GridLayoutManager** або **StaggeredGridLayoutManager**, які надають різні способи відображення списку.

Дані - набір даних, які будуть відображені в **RecyclerView**. Це може бути список об'єктів або колекція, яка містить дані для кожного елемента списку. У лістингу 2.8 приведена модель даних повідомлення.

Лістинг 2.8 - Модель даних повідомлення

```
data class MessageItem(
    val id: String,
    val chatId: String,
    val authorId: String,
    val authorName: String,
    val authorPhoto: String?,
    val shortAuthorName: String,
    val text: String,
    val time: Long,
    val authorIsMe: Boolean
)
```

За допомогою прапора **authorIsMe** в адаптері списку є можливість ідентифікувати, чи прийшло повідомлення від користувача або від інших учасників

чатом, та в залежності від цього використовувати різну xml-верстку для зовнішнього вигляду повідомлення.

У лістингу А.3 додатку А наведений код класів ViewHolder, що допомагають забезпечити відображення різних типів повідомлень чату на екрані RecyclerView. Вони обробляють зв'язування даних з віджетами та реагують на події кліку на елементи списку.

Клас UserMessageViewHolder є підкласом MessageViewHolder і використовується для відображення повідомлень інших користувачів. Він має додаткові поля і методи, такі як onItemClick, який викликається при кліку на повідомлення, і case, який визначає тип розміщення елементів у списку (з роздільником дати або без нього). В методі onBind здійснюється заповнення віджетів даними повідомлення, такими як текст, інформація про автора та час повідомлення.

Клас MyMessageViewHolder також є підкласом MessageViewHolder і використовується для відображення повідомлення користувача. Він має схожу структуру з UserMessageViewHolder і має ті ж самі поля і методи, такі як onItemClick і case. В методі onBind також здійснюється заповнення віджетів даними повідомлення.

У класі адаптеру списку чата є метод метод onCreateViewHolder, який наведений у лістингу 2.9. Він створює і повертає об'єкт MessageViewHolder для кожного типу елемента списку. Залежно від типу, використовуються різні макети ListItemMyMessageBinding або ListItemUserMessageBinding, а також передаються необхідні зворотні виклики та параметри.

Лістинг 2.9 - Код методу адаптера списку onCreateViewHolder

```
override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): MessageViewHolder {
    return when (viewType) {
        MY_MESSAGE -> MyMessageViewHolder(
            ListItemMyMessageBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
```

Продовження лістингу 2.9

```

        ), onMessageScroll, onItemClick,
        MyMessageViewHolder.WITHOUT_DATE_DIVIDER
    )
    MY_MESSAGE_WITH_DATE_DIVIDER ->
    MyMessageViewHolder(
        ListItemMyMessageBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        ), onMessageScroll, onItemClick,
        MyMessageViewHolder.WITH_DATE_DIVIDER
    )
    USER_MESSAGE -> UserMessageViewHolder(
        ListItemUserMessageBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        ), onMessageScroll, onItemClick,
        UserMessageViewHolder.WITHOUT_DATE_DIVIDER
    )
    USER_MESSAGE_WITH_DATE_DIVIDER ->
    UserMessageViewHolder(
        ListItemUserMessageBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        ), onMessageScroll, onItemClick,
        UserMessageViewHolder.WITH_DATE_DIVIDER
    )
    else -> throw RuntimeException("Such viewType
    doesn't exist")
    }
    }

```

За відображення і взаємодію з елементами інтерфейсу фрагмента чату відповідає клас ChatFragment.kt. У лістингу 2.10 наведено налаштування для RecyclerView, що включають встановлення менеджера макету (LinearLayoutManager), налаштування параметрів прокрутки і присвоєння адаптера.

При отриманні списку повідомлень з ViewModel, список оновлюється в адаптері chatAdapter, і прокручується вниз для відображення нових елементів. При зміні стану завантаження, відображається або приховується індикатор завантаження loadingLayout.

Окрім цього, на ChatFragment.kt є кнопка btnSendMessage і поле введення etMessage, які налаштовуються для надсилання повідомлень, залежно від вмісту тексту.

Механізм collectOnLifecycle на цьому екрані використовується для підписки на дані з Flow у ViewModel і оновлення відповідних елементів інтерфейсу фрагмента в реальному часі.

Лістинг 2.10 - Налаштування чату на екрані ChatFragment.kt

```

override fun onCreateView(view: View, savedInstanceState:
Bundle?) {
    super.onCreateView(view, savedInstanceState)
    viewModel.club.collectOnLifecycle { club ->
        binding.header.tvHeaderTitle.text = club.name
    }
    with(binding.header) {

btnCustomAction.setIconResource(R.drawable.ic_info)
        btnCustomAction.setOnClickListener {
            viewModel.navigateToClubInfo()
        }
        btnBack.setOnClickListener {
            viewModel.navigateBack()
        }
    }
    with(binding) {
        rvChat.layoutManager =
LinearLayoutManager(requireContext()).apply {
            stackFromEnd = true
            reverseLayout = false
        }
        rvChat.adapter = chatAdapter
        btnSendMessage.isVisible =
!etMessage.text.isNullOrEmpty()
        btnSendMessage.setOnClickListener {

viewModel.sendMessage(etMessage.text.toString())
            etMessage.text = null
        }
        etMessage.doOnTextChanged { text, _, _, _ ->
            btnSendMessage.isVisible =
!text.isNullOrEmpty()
        }
        viewModel.messages.collectOnLifecycle {
            chatAdapter.submitList(it) {

```

Продовження лістингу 2.10

```
rvChat.smoothScrollToPosition(chatAdapter.currentList.lastIndex + 1)
    }
}
viewModel.isLoading.collectOnLifecycle { loading ->
    loadingLayout.root.visibility = if (loading)
View.VISIBLE else View.GONE
}
}
```

Тепер розглянемо як влаштовані `domain` та `data` шари модулю з чатом. Для реалізації механізму оновлення повідомлень у реальному часі був розроблений додатковий тип сценарію використання, що повертає зазначений тип даних у `Flow`.

`Flow` - тип `Kotlin`, який може емітувати кілька значень послідовно, на відміну від `suspend` функцій, які повертають тільки одне значення. `Flow` здатний представляти будь-яку послідовність значень, які протягом часу надходять асинхронно. За допомогою `Flow` можна обробляти дані в міру їх надходження, а не чекати на завершення всієї операції. Це робить його ідеальним для роботи з даними в реальному часі, такими як потоки з мережі, баз даних або введення користувача.

Лістинг 2.11 - Сценарій використання `SubscribeOnMessagesUseCase`

```
class SubscribeOnMessagesUseCase(
    coroutineContext: CoroutineContext,
    errorHandler: ErrorHandler,
    private val chatRepository: ChatRepository,
    private val userIdCache: UserIdCache
) : FlowUseCase<SubscribeOnMessagesUseCase.Param,
List<Message>>> {
    coroutineContext = coroutineContext,
    errorHandler = errorHandler) {
    override fun executeActual(param: Param):
Flow<List<Message>> {
        return chatRepository.subscribeOnMessages(
            chatId = param.chatId,
            userId = userIdCache.get()
        )
    }
    data class Param(val chatId: String)
}
```

Клас `SubscribeOnMessagesUseCase` представляє сценарій використання у випадку підписки на повідомлення чату. Він успадковується від `FlowUseCase`, який є загальним класом базового використання для потокових операцій.

Серед залежностей цього класу можна побачити інтерфейс `chatRepository`, в якому визначені методи для створення чату, відправки та підписки на повідомлення. Мета цього інтерфейсу - визначити загальний контракт для роботи з чатами, щоб його можна було використати в різних частинах додатку, не залежно від конкретної реалізації чату, реалізація якого у даному проекті належить архітектурному шару `data`, та називається `ChatRepositoryImpl`, код якого можна побачити у лістингу A.4 додатку A.

Лістинг 2.12 - Інтерфейс `ChatRepository`

```
interface ChatRepository {
    suspend fun addChat(members: List<String>): String
    suspend fun updateChatMembers(chatId: String,
    chatMembersIdsList: List<String>): Boolean
    suspend fun sendMessage(chatId: String, userId: String,
    text: String): Boolean
    fun subscribeOnMessages(chatId: String, userId: String):
    Flow<List<Message>>
}
```

`ChatRepositoryImpl` є реалізацією інтерфейсу `ChatRepository` та відповідає за взаємодію з чатами та повідомленнями у `Firebase Firestore`. Основні методи цього класу включають `addChat`, `updateChatMembers`, `sendMessage` та `subscribeOnMessages`.

- `addChat` створює новий чат з вказаними учасниками. Він повертає ідентифікатор створеного чату.
- `updateChatMembers` оновлює список учасників заданого чату.
- `sendMessage` відправляє повідомлення від користувача в чат. Він повертає `true`, коли повідомлення успішно додано.
- `subscribeOnMessages` створює потік повідомлень з заданого чату. Коли нове повідомлення додається в чат, воно автоматично включається в цей потік.

Реалізація цього репозиторію використовує Firebase Firestore для зберігання та отримання даних, екземпляри якого виступають у ролі архітектурного компоненту Data Source. Він має дві колекції в Firestore: одну для чатів `chatsCollection` та одну для повідомлень - `messagesCollection`.

Окрім цього, цей клас містить додатковий приватний метод `mapToMessage`, який перетворює (формат повідомлення, що використовується Firebase) в `Message` (формат повідомлення, що використовується у додатку). Компаніон-об'єкт цього класу містить дві константи, які визначають назви колекцій чатів та повідомлень у Firestore.

2.9 Розробка бази даних додатку

Для роботи системи у консолі Firebase було створено 5 колекцій, що відображають сутності розроблюваної БД, а саме `chats`, `clubs`, `events`, `messages`, та `users`. На відміну від реляційних СУБД, таких як MySQL, Firebase має наступні типи даних:

- **String**: Рядковий тип даних, який представляє послідовність символів.
- **Number**: Числовий тип даних, який включає цілі числа та числа з плаваючою точкою.
- **Boolean**: Логічний тип даних, який представляє значення `true` або `false`.
- **Object**: Об'єктний тип даних, який дозволяє зберігати колекції ключ-значення. Об'єкти можуть вкладатися один в одного, створюючи структуру даних з багаторівневою вкладеністю.
- **Array**: Масивний тип даних, який дозволяє зберігати послідовність значень. Масиви можуть містити елементи різних типів даних.
- **Null**: Спеціальне значення, що позначає відсутність даних або невизначеність.
- **Timestamp**: Тип даних, що представляє момент в часі. В Firestore використовується об'єкт `Timestamp`, який містить дату та час.

- Geopoint: Тип даних, що представляє географічну точку з координатами широти та довготи.

- Reference: Тип даних, який вказує на інший документ у базі даних Firestore. Це дозволяє створювати зв'язки між документами.

В Firestore, для кожного документа, що зберігається у колекції, генерується унікальний ідентифікатор (ID), механізм роботи якого дуже схожий на первинний ключ з автоінкрементацією у SQL-подібних СУБД. При генерації ID Firestore використовує алгоритм, що забезпечує унікальність ID на рівні всієї бази даних, незалежно від колекції. Це гарантує, що нові документи будуть мати унікальний ID та можуть бути однозначно ідентифіковані у Firestore, одже його можна використовувати для ідентифікації та отримання доступу до конкретного документа. Також усі поля документів в Firebase за замовченням можуть мати значення Null.

Окрім цього, у Firestore, на відміну від традиційних реляційних баз даних, немає поняття "первинного ключа" або "зовнішнього ключа". Це документ-орієнтована БД, яка зберігає дані у вигляді колекцій документів, а не в таблицях зі зв'язками між ними. Замість цього, Firestore надає можливість використовувати поля в документах для створення зв'язків між колекціями. Користувач сам вирішує, яке поле використовувати для зв'язку між сутностями, і це поле може бути будь-якого типу даних, такого як рядок, число або навіть посилання на інший документ. Ця особливість Firestore відкриває безліч можливостей для гнучкості та адаптації системи до конкретних потреб проекту. Використання полів для зв'язків між документами дозволяє легко встановлювати та змінювати зв'язки між різними колекціями даних. Крім того, такий підхід дає змогу ефективно оптимізувати запити та зменшувати навантаження на базу даних.

Схему розробленої бази наведено на рис. 2.6. Поля, за допомогою яких створюється зв'язок між документами різних колекцій, на схемі виділено жирним шрифтом. Самі зв'язки позначені за допомогою кольорових ліній. Тип зв'язку “багато до багатьох” позначений за допомогою пунктиру.

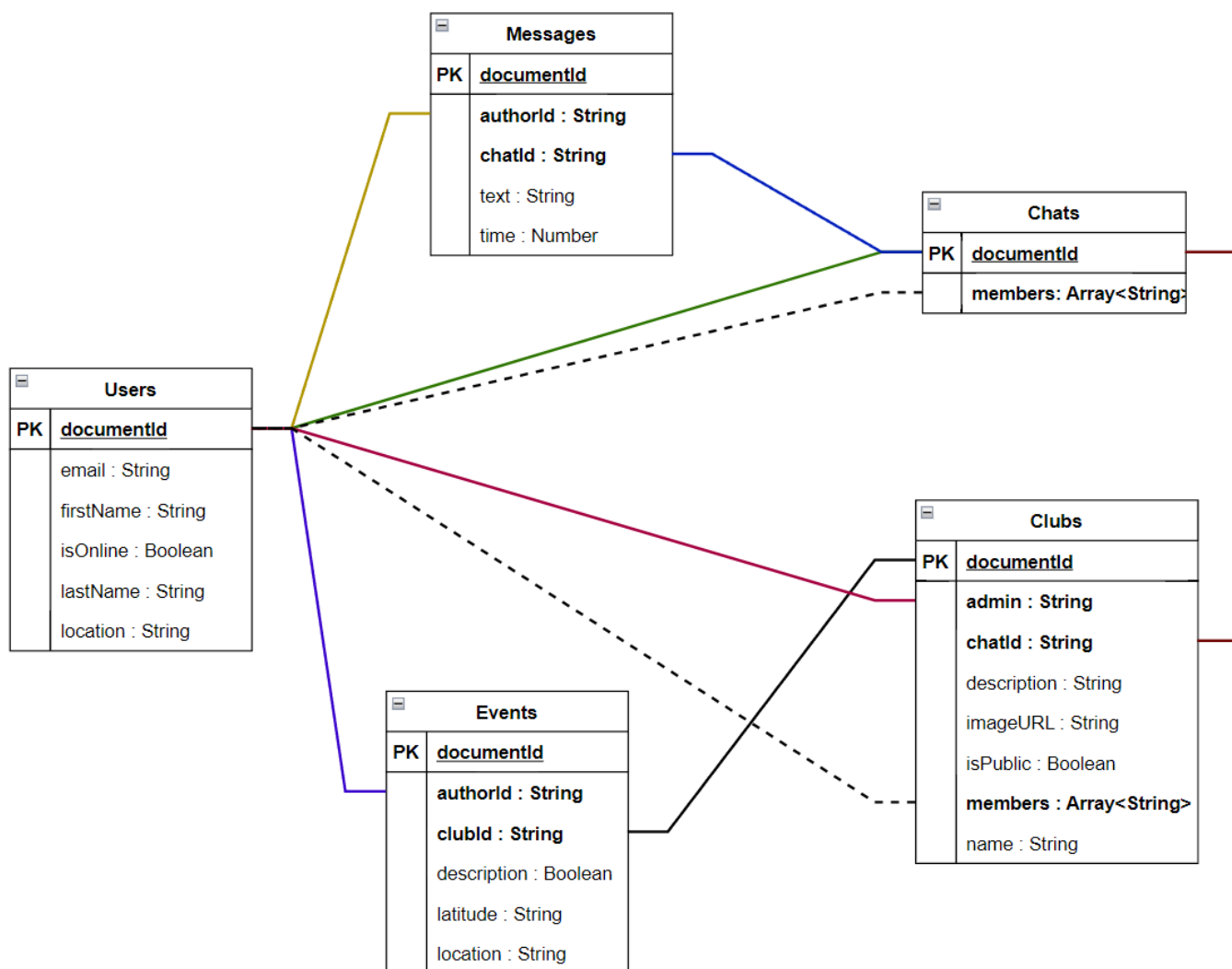


Рисунок 2.6 – Схема розробленої бази даних

На схемі представлено наступні сутності:

User. Містить основну інформацію про користувача, таку як ім'я, прізвище, електронна пошта, місцезнаходження та статус онлайн.

Event. Містить назву, опис, час заходу, широту та довготу місця проведення, ідентифікатор клубу, та ідентифікатор автора заходу. Користувач може бути автором заходу (зв'язок один до багатьох: Users до Events). Id клубу може мати значення null, якщо захід не пов'язаний з клубом.

Club. Містить назву, опис, URL-адресу зображення (якщо є), прапор, що позначає, чи є клуб публічним, список учасників клубу, ідентифікатор адміністратора клубу та ідентифікатор чату, пов'язаного з клубом. Клуб може мати багатьох учасників, а також одного адміністратора (зв'язок багато до багатьох:

Users до Clubs). Також клуб має пов'язаний чат (зв'язок один до одного: Clubs до Chats).

Chat. Містить список учасників чату. Зв'язок між чатом (Chats) та користувачами (Users) у базі має тип "багато до багатьох". Це означає, що в одному чаті можуть бути багато учасників, і кожен користувач може бути учасником багатьох чатів.

Message. Містить ідентифікатор чату, ідентифікатор автора повідомлення, текст повідомлення та час надсилання повідомлення. Повідомлення пов'язане з конкретним чатом та автором (зв'язок багато до одного: Chats та Users до Messages).

2.10 Висновки щодо розробки розподіленої системи

У процесі розробки розподіленої системи було здійснено широкий спектр дій і визначено різні аспекти, що сприяють успішній реалізації проекту.

Аналізуючи основні етапи розробки додатку, було виділено такі ключові складові: Backend, Frontend, Випробування ПЗ та Розміщення у магазині програм. Були ретельно визначені технічні вимоги до системи. Зокрема, були встановлені вимоги до робочих характеристик серверної та клієнтської частини. Це дозволило підібрати апаратні компоненти, що відповідають вимогам продуктивності та надійності системи. Також були описані вимоги до безпеки даних, логічного устрою БД та інтерфейсу програми, що поліпшує ефективність та зручність використання.

Особлива увага була приділена розробці функціональності інтерфейсу додатку. На її основі було проведено моделювання сутностей, таких як користувач, клуб, захід, чат та повідомлення, що дозволяють забезпечити багатофункціональний та зручний для користувача інтерфейс.

Архітектура системи була ретельно пророблена з метою забезпечення її стабільності та масштабованості. Проведений огляд бібліотеки для роботи з мапою

Марбох дозволив визначити найкращий варіант для використання в системі з урахуванням функціональних потреб.

Було розроблено опис та схеми структури системи, що дозволяють зрозуміти загальну організацію та взаємозв'язки компонентів.

Важливим етапом розробки була реалізація модулів клієнтської частини. Для цього були використані різноманітні технології та програмні інструменти, які дозволили ефективно створити необхідний функціонал. Лістинги основних класів додатку були розроблені з урахуванням кращих практик програмування, що сприяло створенню структурованого та легкозрозумілого коду.

Окрім того, була розроблена база даних додатку, яка відповідає вимогам логічного устрою та безпеки даних. Для забезпечення надійності та продуктивності були використані відповідні технології та методи, а також платформа Firebase. Важливим аспектом було забезпечення цілісності даних, що дозволило зберігати та обробляти інформацію ефективно.

Таким чином, завдяки врахуванню вимог та етапів розробки, ретельному проектуванню архітектури та використанню відповідних технологій, вдалося створити функціональну та надійну систему. Цей розділ є ключовим для подальшої успішної реалізації та функціонування додатку, адже він забезпечує основу для наступних етапів проекту.

3 ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Опис сценаріїв роботи з додатком

Після успішної розробки розподіленої системи, наступним кроком є проведення випробувань та перевірка її працездатності. Випробування допоможуть виявити можливі помилки та недоліки в системі, а також переконатися в її відповідності вимогам та очікуванням користувачів.

Для цього необхідно провести моделювання сценаріїв використання функціональних можливостей додатку. Розглянемо наступні ситуації: створення клубу, створення заходу та навігації до заходу. Для фіксації кожного кроку будуть використані скриншоти з мобільного додатку, а також консолі від Firebase для моніторингу змін даних у реальному часі.

3.1.1 Створення клубу

Першим сценарієм є ситуація, у якій користувач бажає створити новий клуб для любителів певних мотоциклів, заповнюючи текстові поля, завантажуючи зображення, та додаючи нових учасників.

Майбутній клуб буде присвячений фанатам продукції фірми Harley, що є американським виробником мотоциклів. Назвемо цей клуб “Harley Fans”, коротко заповнемо поля з описом, та додамо нових байкерів. Перш за все, необхідно відкрити сторінку усіх клубів, з якої можна перейти до сценарію створення клубу. Для цього на головному екрані перейдемо у профіль користувача, а далі потрапимо на сторінку клубів за допомогою кнопки “My Clubs”. У правому верхньому куті екрану побачимо кнопку для створення клубу.

Відкриємо сторінку створення нового клубу. На цьому екрані бачимо вікно для завантаження зображення, пусті поля та елемент для вибору типу клубу. Зображення клубу є опціональним параметром. У разі його відсутності на сервері відповідне поле буде мати значення null, яке оброблюється додатком, та в залежності від його наявності показується фото або перша літера назви. Для демонстрації можливостей додатку завантажимо з галереї відповідне фото. Далі заповнимо текстові поля з назвою та описом, та оберемо тип клубу. У разі успішної валідації, кнопка для збереження уведених даних та перехід на екран додавання користувачів стане активною. Натискаємо на кнопку “Invite users”, та потрапляємо на сторінку зі списком усіх користувачів системи. За бажанням обираємо відповідних майбутніх учасників, натискаючи на елемент списку, та завершуємо створення клубу. Створений клуб побачимо на сторінці зі списком усіх клубів, на яку автоматично здійснюється навігація після завершення цього сценарію. Діаграма сценарію використання для створення клубу приведена на рис. 3.1.

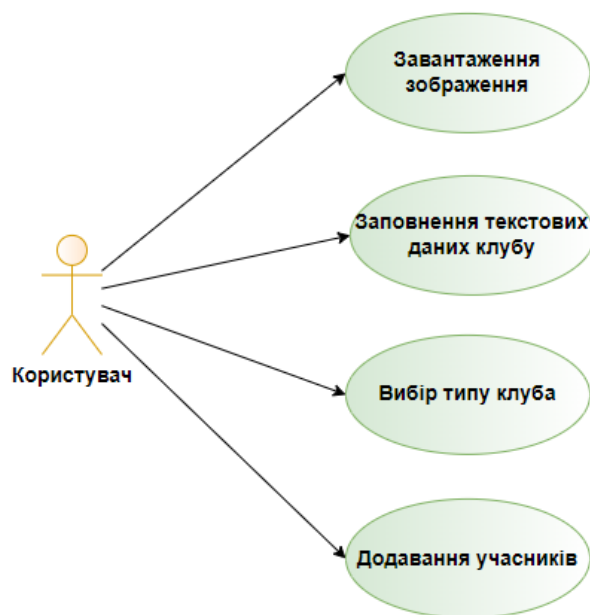


Рисунок 3.1 – Діаграма сценарію використання для створення клубу

3.1.2 Створення заходу

Змоделюємо другий сценарій, у якому адміністратор клубу “Harley Fans” побажав створити рок-фестиваль “Harley Rocks” на Жданівському пляжі міста Запоріжжя на майбутні вихідні, до якого зможуть долучитися усі учасники клубу. Для цього перш за все необхідно визначитись із місцем проведення заходу.

Для початку, відкриємо сторінку клубу, чий захід буде створено. Для цього у списку оберемо клуб “Harley Fans”, потрапимо на чат, з якого за допомогою кнопки у тулбарі виконаємо навігацію на інформаційну сторінку. У нижній частині екрану натискаємо на кнопку “Create club event”, після чого бачимо вікно з мапою, та маркером посередені, який вказує на поточне місцезнаходження адміністратора клубу. За допомогою жестів пересуваємо мапу так, щоб маркер вказував на Жданівський пляж. Якщо все виконано правильно, у вікні, що відображає адресу, на яку вказує маркер, буде вказано “Tbilis’ka Street, 25, Vosnesenivksyi district, Zaporizhzhia, Zaporiz’ka oblast, Ukraine”.

Після підтвердження адреси за допомогою кнопки “Next” потрапляємо на екран с заповненням інформації щодо заходу. Заповнюємо поля з назвою та описом, а також за допомогою діалогів для вибору календарного дня та часу

задаємо дату проведення заходу. Після відправки даних на сервер, пересвідчимося в успішному створенні заходу, відкривши головну мапу на Жданівському пляжі.

На Рисунку 3.2 зображена діаграма використання для створення клубного заходу. До можливостей користувача входить вибір місця, заповнення текстових даних та вибір дати проведення.

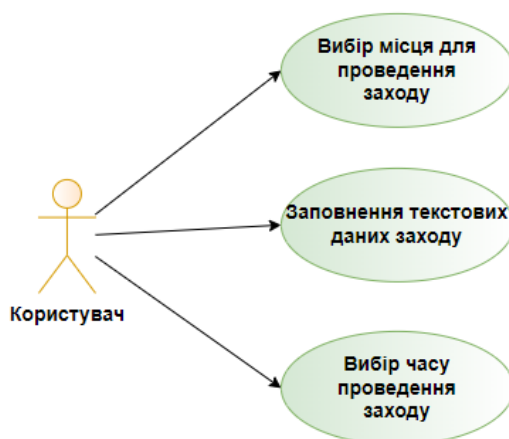


Рисунок 3.2 – Діаграма сценарію створення заходу клубу

3.1.3 Навігація до заходу

Третій сценарій являє собою ситуацію, коли учаснику або адміну клубу необхідно дістатися на своєму байку до місця проведення заходу.

Для тестів використаємо попередньо-створений захід “Harley Rocks”.

Спочатку треба відкрити головний екран додатку з мапою, на якій необхідно знайти кольоровий маркер заходу на Жданівському пляжі у місті Запоріжжя. Натискаємо на маркер, та у діалозі з інформацією, що з’явився над мапою, натискаємо на кнопку “Navigate”. Якщо зона розташування користувача дозволяє доїхати до пляжу, після натискання на кнопку ми побачимо як камера віддаляється для того, щоб охопити весь побудований маршрут. Маршрут буде відображатися у вигляді синьої лінії на мапі, вказуючи шлях до об’єкту. Окрім цього, під кнопкою для показу поточного розташування з’явиться нова кнопка для переходу у режим навігації, якщо побудований маршрут нас влаштовує.

Переходимо у режим навігації натисканням на відповідний UI-елемент, та бачимо новий екран з мапою, де відображаються маршрут, користувач, кінцева

точка прибуття, інформаційні дані про поїздку та візуальні дорожні підказки. Під час руху на реальному транспорті спостерігаємо як мапа змінюється в реальному часі, відображаючи поточне положення користувача та його переміщення по маршруту. Навігаційні підказки будуть включати напрямки поворотів, розташування маневрів, прогнозований час прибуття та іншу корисну інформацію. В процесі подорожі ми будемо отримувати голосові команди та візуальні підказки для керування по маршруту без відволікання уваги від дороги.

Під час руху до точки призначення у режимі навігації, додаток Mapbox забезпечує постійне відстеження місцезнаходження користувача, враховуючи зміну напрямку та швидкості руху. Він також автоматично коригує маршрут у разі зміни напрямку або пропуску повороту. Якщо ми відхиляємося від маршруту, додаток надсилає попередження та пропонує оптимальний шлях для повернення на маршрут. Завершивши подорож, байкер отримує повідомлення те що маршрут закінчений. Вихід з режиму навігації здійснюємо за допомогою спеціальної кнопки на інформаційній панелі, та потрапляємо на головний екран з мапою.

На Рисунку 3.3 зображена діаграма використання для навігації до клубного заходу. У ній користувач обирає захід на мапі, переглядає інформацію про нього, прокладає маршрут та виконує подорож за допомогою режиму навігації.

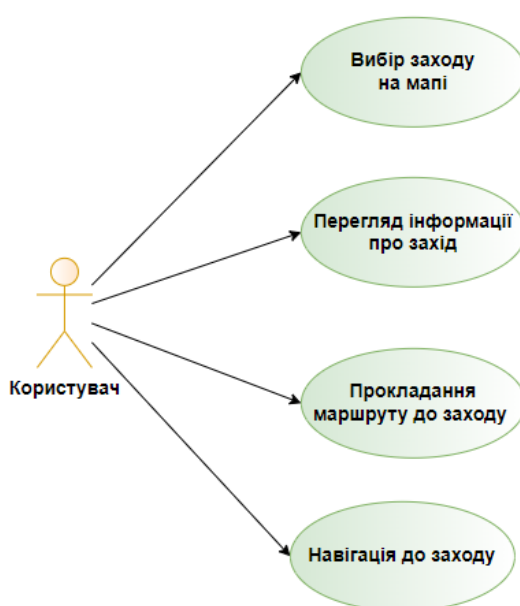


Рисунок 3.3 – Діаграма сценарію навігації до заходу

3.2 Демонстрація виконання описаних сценаріїв роботи з додатком

3.2.1 Сценарій створення клубу

Для того, щоб створити новий клуб, необхідно відкрити відповідний екран додатку. На ньому ми бачимо вікно для додавання зображення та пусті поля.

Почнемо з вибору зображення. Для цього натискаємо на іконку пейзажу. Після цього відкривається діалог, який запропонує нам обрати зображення з галереї, або з камери. Обираємо, галерею, та шукаємо необхідну картинку.

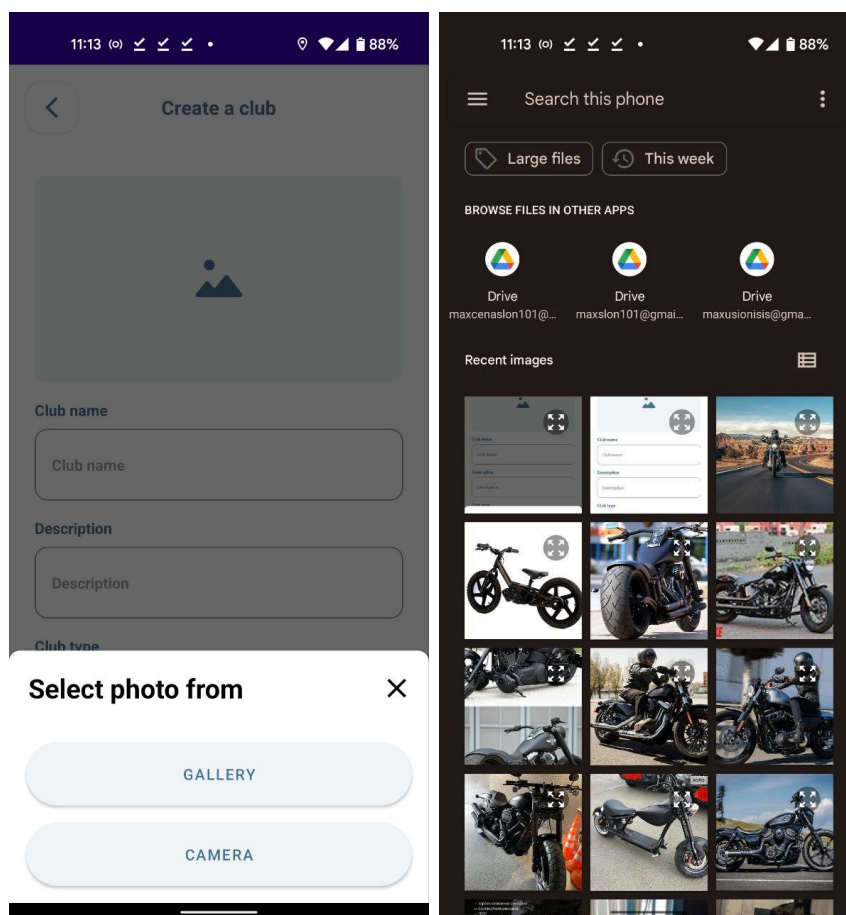


Рисунок 3.4 – Діалог та системне вікно для вибору зображення

Далі необхідно заповнити усі пусті поля, адже при наявності хоча б одного пустого поля локальна валідація не дозволить продовжити створення і кнопка для збереження даних буде неактивною.

Заповнення полів почнемо з назви клубу “Harley Fans”, яку вводимо у поле з підписом Club name. Потім додамо невеликий опис клубу “This is a club for Harley Funs”. Для демонстрації валідації на рис 3.5 наведено скриншот стану поля, якщо з нього видалити увесь текст.

За замовченням перемикач типу клубу встановлений в public. Ми бажаємо, що інші користувачі могли додавати своїх друзів та для росту та популяризації клубу серед байкерської, одже залишаємо його незмінним.

Після успішної валідації натискаємо на кнопку “Invite users”, та потрапляємо на екран додавання нових користувачів.

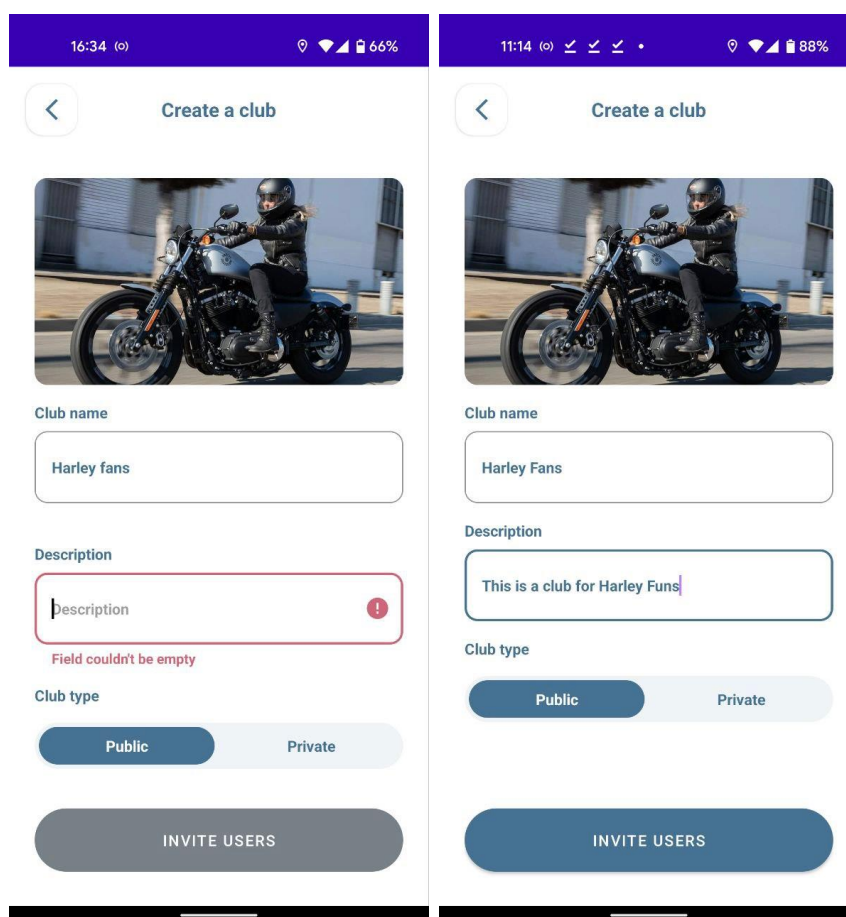


Рисунок 3.5 – Заповнення текстових полів та демонстрація валідації

На екрані додавання нових учасників клубу ми бачимо байкерів, що на поточний момент вже зареєстровані та використовують додаток. Натискаючи на елемент списку, ми обираємо майбутнього учасника. Варто зазначити, що додаток

не забороняє нам створити клуб без додавання нових людей. У цьому випадку єдиним учасником клубу буде адміністратор, тобто ми самі. Додамо усіх наявних байкерів в системі, та завершимо створення клубу.

На рис. 3.6 приведений список користувачів та статус екрана під час відправки даних. Після натискання на кнопку “Create a club”, дані, що були заповнені на попередньому екрані, та список користувачів, відправляється єдиною моделлю ClubInput, та зберігаються на серверах Firebase. Під час відправки даних на сервер, що є довготривалою операцією, ми бачимо індикатор завантаження, що не дасть взаємодіяти з елементами інтерфейсу додатку доки операція не буде виконана.

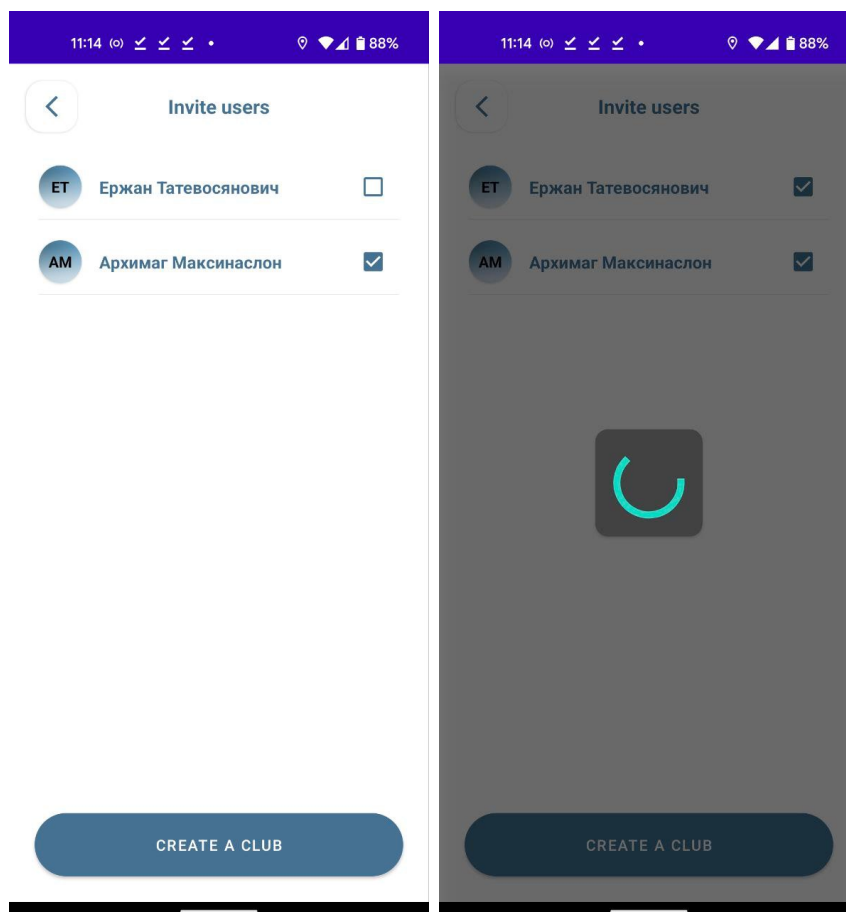


Рисунок 3.6 – Додавання користувачів та завершення створення клубу

Тепер зайдемо у консоль Firebase, та переглянемо, чи з'явився новий документ із даними тільки що створеного клубу “Harley Funs”.

На рис. 3.7 можна побачити, що до колекції clubs з'явився новий документ з полями назви та опису, що ми заповнювали на клієнтському додатку. Перейдемо на сторінку зі списком усіх клубів, та побачимо “Harley Fans” поряд з іншими клубами. На рис. 3.8 показано, що “Harley Fans” відображається у категорії “My Clubs”, що означає те, що даний клуб був створений нами. У підзаголовку елемента клубу бачимо, що його учасниками є три користувача, включаючи нас.

З наведених скриншотів робимо висновок, що сценарій створення нового клубу працює, та відповідає вимогам користувачів системи.

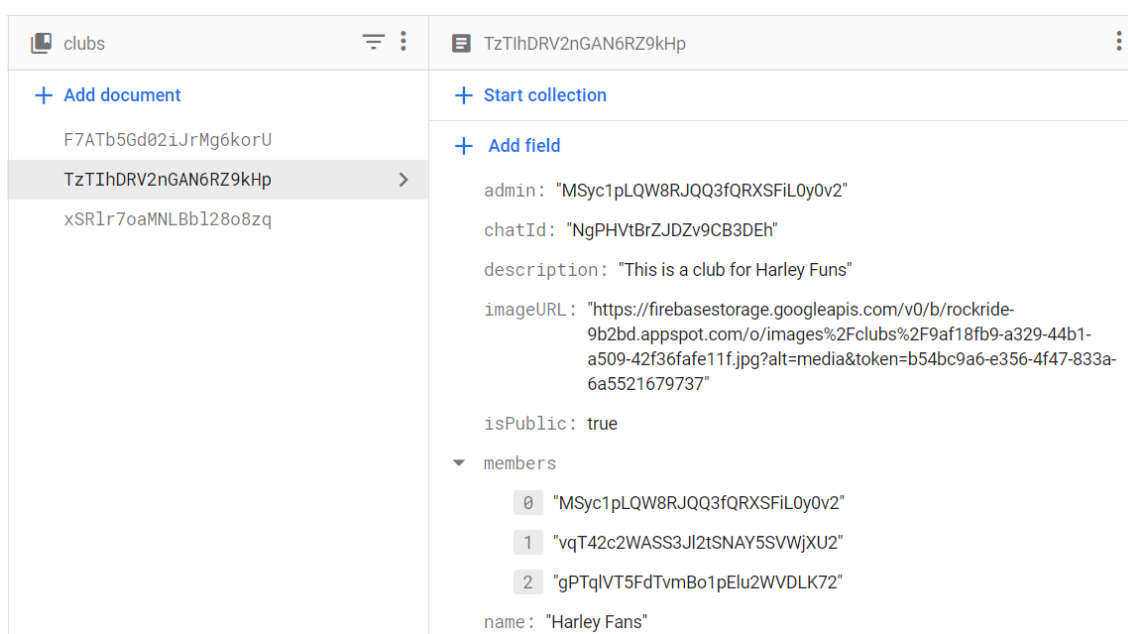


Рисунок 3.7 – Перегляд створеного клубу у Firebase Console

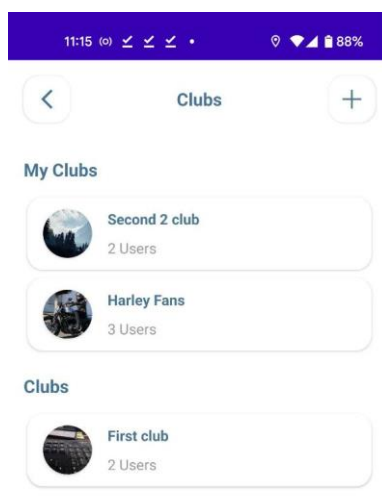


Рисунок 3.8 – Перегляд створеного клубу у на сторінці зі списком

3.2.2 Сценарій створення заходу

Для створення нового клубного заходу, необхідно відкрити сторінку клубу. У даному випадку ми створюємо захід, що пов'язаний з рок фестивалем клубу “Harley Fans”, одже обираємо його зі списку усіх доступних нам клубів, пропускаємо екран з чатом, та одразу переходимо на сторінку клубу.

На сторінці клубу ми, як адміністратор, бачимо кнопку з олівцем для редагування клубу, та кнопку для створення заходу “Create club event”. Натискаємо на останню, та потрапляємо до екрану з вибором місця. За замовченням додаток бере точку, що знаходиться над нашим поточним місцерозташуванням. Пересуваємо разом з мапою маркер так, щоб він опинився за адресою Тбіліська вулиця, 25. Саме тут буде точка збору байкерів для проведення рок-фестивалю.

Скриншот сторінки клубу та вибору місця проведення на карті наведені на рис. 3.9.

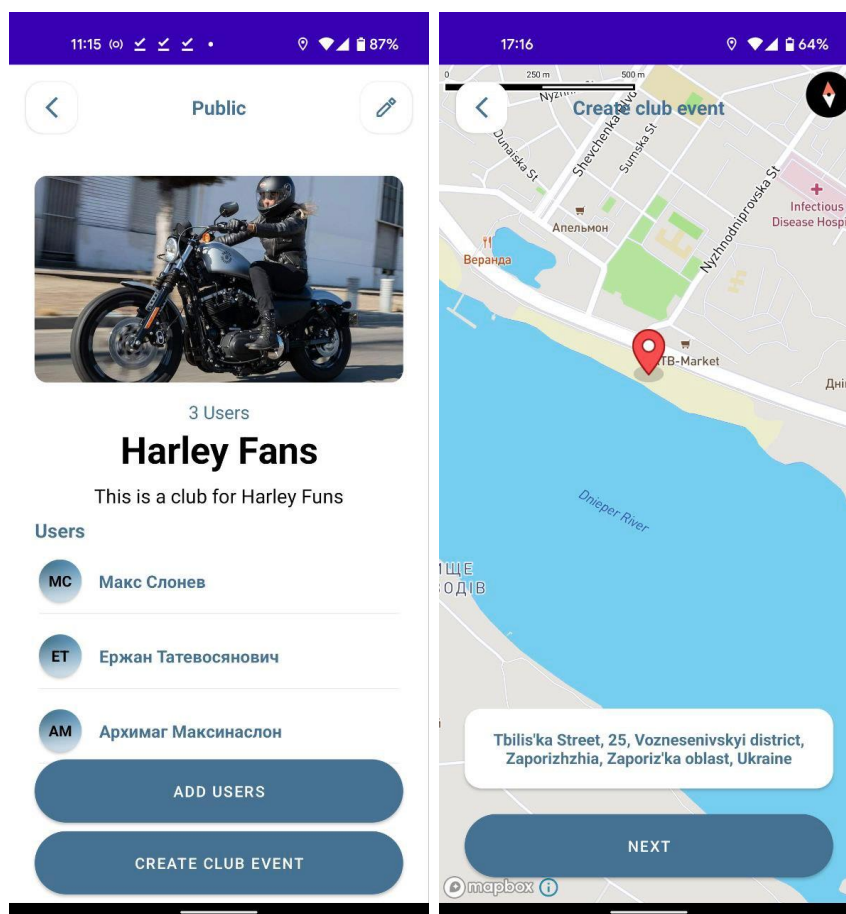


Рисунок 3.9 – Сторінки клубу та вибору місця проведення на карті

Натискаємо “Next”, і бачимо екран с заповненням інформації щодо заходу з пустими полями. У поле з підписом Event name вводимо назву заходу “Harley Rocks”, а у поле Description – “This is an event of Harley Club”. Далі необхідно задати дату та час заходу. Натискаємо на елементи з відповідним підписом і бачимо системні діалоги з календарем та годинником для вибору дати та часу. Після підтвердження обраних чисел бачимо, що додаток оновив ці поля обраними даними, одже ми можемо завершити створення заходу, натиснувши на кнопку “Create event”.

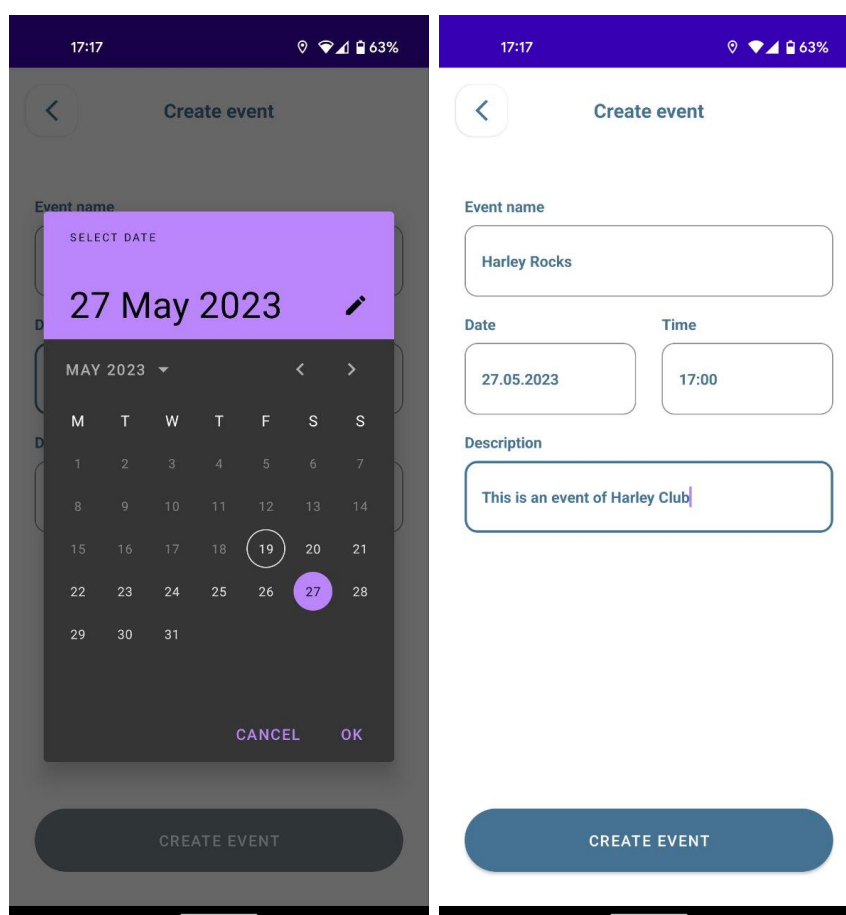


Рисунок 3.10 – Сторінка створення заходу та вибір бажаної дати з календаря

Для того, щоб переконатися, що захід був створений, зайдемо у консоль Firebase, та побачимо новий документ з полями, що ми задали у додатку. Скриншот БД Firestore, в якій зберігаються дані зі створеним заходом “Harley Rocks”, приведений на рис. 3.11.

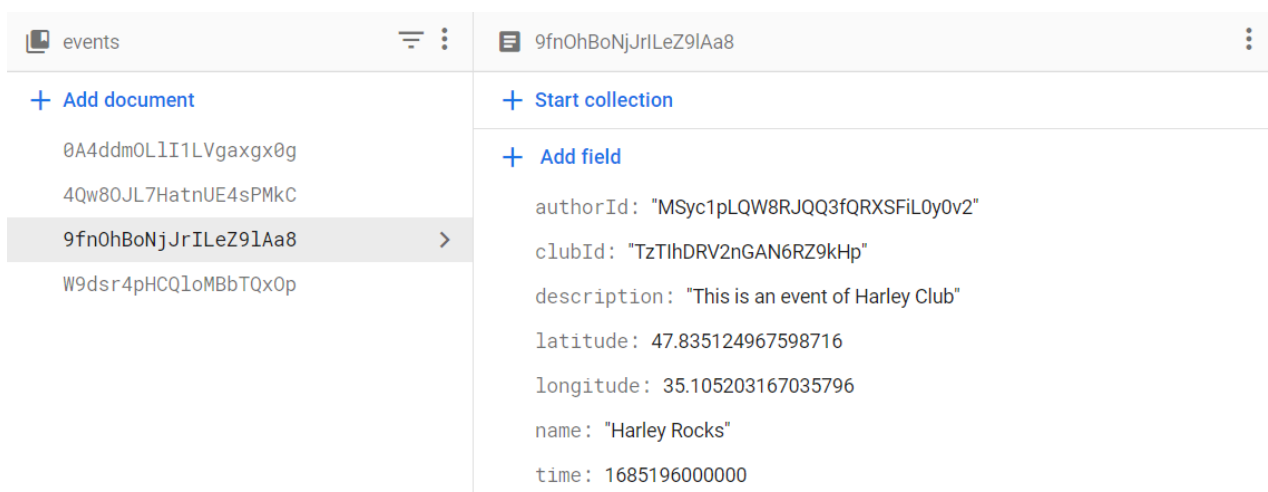


Рисунок 3.11 – Перегляд створеного заходу у Firebase Console

3.2.2 Сценарій навігації до заходу

Знайдемо на мапі тільки що створений захід “Harley Rocks”. Відкриємо головний екран, та знайдемо на мапі Жданівський пляж у місті Запоріжжя. На рисунку 3.12 можна побачити, що на пляжі відображається синій маркер, який символізує захід клубу. Заходи, що не належать до клубів, та створюються будь-якими користувачами, на мапі позначені зеленим кольором.

Натиснемо на маркер. Перед нами відкриється вікно, де ми можемо побачити уведену нами інформацію під час створення цього заходу. За допомогою хрестика у правому верхньому куті вікна, або свайпа по екрану, ми можемо закрити його, та продовжити працювати з мапою.

Натискаємо на кнопку “Navigate” у діалоговому вікні та бачимо, як камера карти віддаляється від маркеру для того, щоб охопити весь побудований маршрут до пляжу від нашого поточного місцезнаходження. Актуальний шлях виділено синім кольором на мапі. Дана дорога нас влаштовує, одже натискаємо на кнопку зі стрілкою повороту, що з’явилася після генерації маршруту у нижній частині екрану, та переходимо до режиму навігації, скриншот якої приведений на рис. 3.13. Під час реальної їзди, окрім крім підказок, ми бачимо як стрілка, що відображає користувача, пересувається згідно заданого маршруту. Після досягнення заданої точки, натискаємо на синю кнопку з хрестиком в нижній інформаційній панелі, та завершуємо режим навігації.

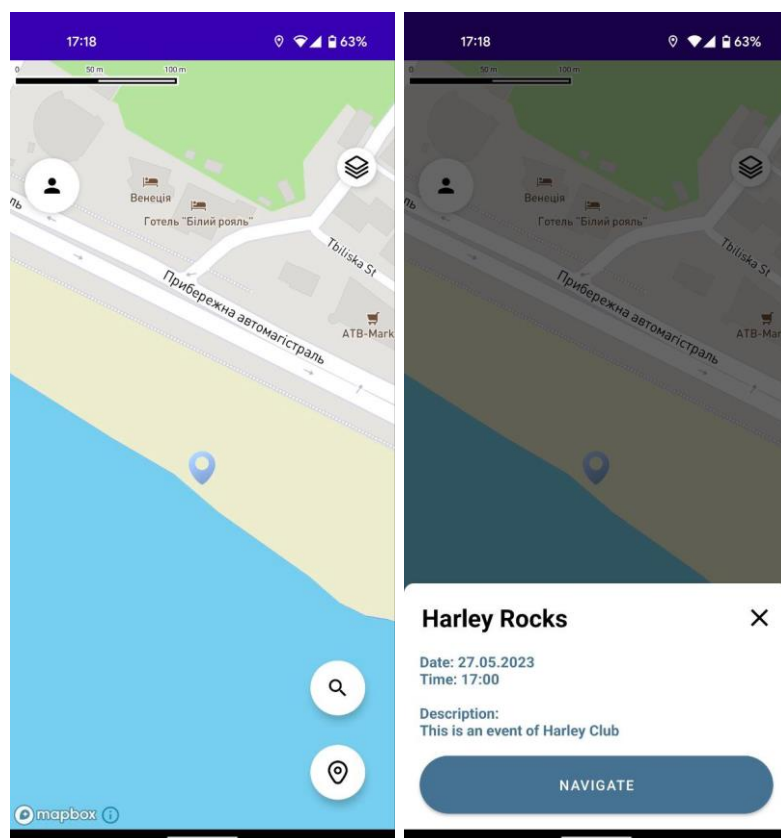


Рисунок 3.12 – Перегляд створеного заходу на мапі

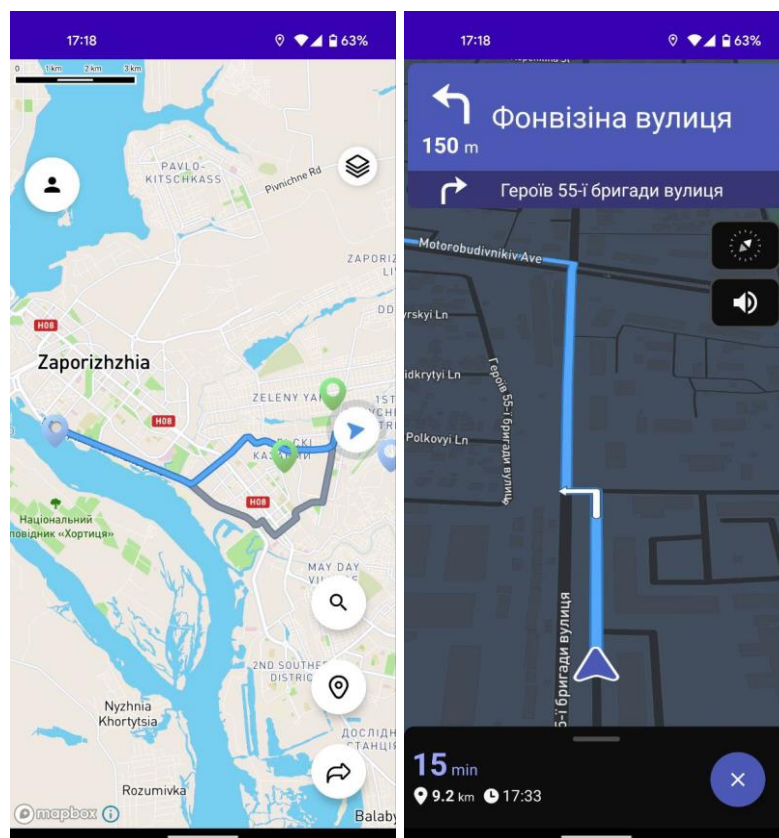


Рисунок 3.13 – Побудова маршруту до фестивалю та режим навігації

3.3 Висновки щодо випробувань розподіленої системи

Під час випробувань розподіленої системи були проведені різноманітні дії для перевірки її функціональності та надійності. Було описано та виконано сценарії роботи з додатком, такі як створення клубу, створення заходу та навігація до заходу.

Детально змодельовано сценарій створення клубу, щоб переконатися в його коректності та правильному функціонуванні. Відтворено усі необхідні кроки, від введення необхідних даних до підтвердження створення клубу. Результати підтвердили, що процес створення клубу працює належним чином.

Також було проведено моделювання сценарію створення заходу, щоб перевірити правильність його виконання. У рамках випробувань було введено відповідні дані, такі як назва заходу, дата, місце проведення та інші важливі деталі. В результаті було підтверджено, що створення заходу відбувається без помилок та відповідає очікуваному функціоналу.

Крім того, було перевірено сценарій навігації до заходу, щоб переконатися в правильності його роботи. У процесі було перевірено можливість користувача знайти та відобразити деталі конкретного заходу, а також прокласти маршрут до точки. Результати випробувань підтвердили, що навігація до заходу відбувається успішно та коректно.

Одже, можна зробити висновок, що проведені випробування підтвердили функціональність та правильну роботу сценаріїв роботи з додатком. Це свідчить про те, що розподілена система пройшла успішні випробування та виконує свої функції відповідно до очікувань. Описані сценарії були відтворені без помилок, а необхідні дії та операції виконувалися належним чином. Це підтверджує, що розроблене програмне забезпечення працює стабільно та може виконувати потрібні завдання.

ВИСНОВКИ

Метою даної дипломної роботи була розробка додатку для байкерів на базі операційної системи Android, який дозволить створювати та організовувати тематичні заходи, відмічати їх на карті, а також полегшити спілкування серед байкерської спільноти.

У ході розробки системи було проаналізовано предметну область, визначені вимоги до системи, на основі яких складено технічне завдання. Для розробки проекту була обрана мова програмування Kotlin, бібліотека з картою Mapbox, та вивчені технології Firebase, Kotlin Coroutines, архітектурний паттерн MVVM і бібліотека Koін для впровадження залежностей.

Систему було спроектовано відносно загальноприйнятих стандартів проектування програмного забезпечення, відповідно до усіх вимог які були описані, а також з можливістю легкої подальшої підтримки та її масштабування. Розроблена схема архітектури клієнтської частини та проведено проектування, в результаті якого були визначені базові архітектурні компоненти додатку та зв'язки між ними. За допомогою платформи Firebase була створена серверна частина з базою даних.

Проведено програмно-апаратна реалізація та випробування системи за допомогою обраних технологій. Розроблений додаток було встановлено та протестовано на пристроях різного цінового сегменту та потужності.

Розроблений додаток повністю відповідає вимогам технічного завдання. Дизайн інтерфейсу задовольняє потребам кінцевого користувача, зручний у використанні на пристроях з різними розмірами екрану.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 Introduction to Monolithic Architecture and MicroServices Architecture.
URL: <https://medium.com/koderlabs/introduction-to-monolithic-architecture-andmicroservices-architecture-b211a5955c63> (дата звернення: 09.05.2023).
- 2 Monolithic Architecture pattern. URL: <https://microservices.io/patterns/monolithic.html> (дата звернення: 09.05.2023).
- 3 MVVM – Model View ViewModel. URL: <https://medium.com/nuances-of-programming/mvvm-na-android-c-компонентами-архитектуры-библиотека-koine2e77b77950e> (дата звернення: 09.05.2023).
- 4 Guide to app architecture - Android Developers. URL: <https://developer.android.com/topic/architecture> (дата звернення: 09.05.2023).
- 5 Horstmann C. S., Cornell G. Core Java Volume I – Fundamentals. 11th ed. — Upper Saddle River, NJ: Pearson Education, Inc., 2018. — 928 p.: ill.
- 6 Jemerov D., Isakova S. Kotlin in Action. / Translated by Kiselev A. N. - M.: DMK Press, 2018. - 402 p.: ill.
- 7 Freeman E., Robson E., Sierra K., Bates B. Head First Design Patterns. 2nd ed. — Kharkiv: Vydavnytstvo "O'Reilly", 2022. — 640 p.: ill. — (Head First O'Reilly Series).
- 8 Griffiths D., Griffiths D. Head First Android Development. 2nd ed. — St. Kharkiv: Vydavnytstvo "O'Reilly", 2018. 912 p.: ill. — (Head First O'Reilly Series).
- 9 Doug Lowe. Android Phones and Tablets for Dummies. 5th ed. — Wiley Publishing, Inc., 2015. — 384 p. ISBN 978-1-118-96917-5.
- 10 Matthew Dolan. Dependency Injection in Android with Dagger 2 and Kotlin. — Manning Publications Co., 2019. — 616 p. ISBN 978-1-6172-9128-5.
- 11 Tomasz Nurkiewicz, Ben Christensen. Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications. 1st ed. — O'Reilly Media, Inc., 2016. — 372 p. ISBN 978-1-491-95159-0.

- 12 Sean McQuillan. Kotlin Coroutines by Tutorials. 1st ed. — Razeware LLC, 2020. — 296 p. ISBN 978-1-950491-15-2.
- 13 David Sklar, Adam Trachtenberg. PHP Cookbook: Solutions and Examples for PHP Programmers. 3rd ed. — O'Reilly Media, Inc., 2014. — 820 p. ISBN 978-1-491-90335-9.
- 14 Cantelon M., Harter T., Holowaychuk T. Node.js in Action. 2nd ed. — Shelter Island, NY: Manning Publications Co., 2017. — 432 p.: ill.
- 15 Jake Wharton. Android App Development with Kotlin. 1st ed. — Addison-Wesley Professional, 2019. — 760 p. ISBN 978-0-13-526180-3.
- 16 Firebase for Android Developers Documentation. URL: <https://firebase.google.com/docs/android> (дата звернення: 09.05.2023).
- 17 ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary. URL: <https://www.iso.org/standard/71952.html> (дата звернення: 09.05.2023).
- 18 ISO/IEC/IEEE 2382:2015 Information technology — Vocabulary. URL: <https://www.iso.org/standard/63598.html> (дата звернення: 09.05.2023).
- 19 Патерни проектування. URL: <https://refactoring.guru/uk/design-patterns> (дата звернення: 09.05.2023).

ДОДАТОК А

Лістинг А.1 - Код класу UserProfileRepositoryImpl

```

class UserProfileRepositoryImpl(
    private val userService: UserService,
    private val userIdCache: UserIdCache
) : UserProfileRepository {

    private val dummyUser = User(UserIdCache.DEFAULT_USER_ID,
    "", "", "", "", false)
    private val userProfile = MutableStateFlow<User>(dummyUser)

    override suspend fun loadUserProfile(): User {
        val fetchedUser =
userService.fetchUser(getUserProfileIdFromCache())
        userProfile.value = fetchedUser
        return fetchedUser
    }

    override suspend fun saveUserProfile(user: User): Boolean {
        userService.saveUser(user)
        userIdCache.save(userId = user.id)
        userProfile.value = user
        return true
    }

    override suspend fun updateUserProfileLocation(newLocation:
Location): Boolean {
        userService.updateUserLocation(
            newLocation = newLocation,
            userId = getUserProfileIdFromCache()
        )
        userProfile.value = userProfile.value.copy(location =
newLocation.toShortOneLineFormat())
        return true
    }

    override suspend fun
updateUserProfileOnlineStatus(isOnline: Boolean): Boolean {
        userService.updateUserOnlineStatus(
            isOnline = isOnline,
            userId = getUserProfileIdFromCache()
        )
        userProfile.value = userProfile.value.copy(isOnline =
isOnline)
        return true
    }

    override fun clearUserCache(): Boolean {
        return userIdCache.clear()
    }
}

```

Продовження лістингу A.1

```

    }
    override fun subscribeOnUserProfile(): StateFlow<User> {
        return userProfile.asStateFlow()
    }
    private fun getUserIdFromCache(): String {
        if (userIdCache.get() != UserIdCache.DEFAULT_USER_ID) {
            return userIdCache.get()
        } else {
            throw
NetworkException.UserNotAuthorizedException("There is no saved
user in cache")
        }
    }
}
}

```

Лістинг A.2 - Головна логіка взаємодії з мапою у MapViewModel.kt

```

class MapViewModel(
    private val mapController: MapController,
    private val router: Router,
    private val updateUserProfileLocationUseCase:
UpdateUserProfileLocationUseCase,
    private val getLocationUseCase: GetLocationUseCase,
    private val subscribeOnUsersEventsUseCase:
SubscribeOnUsersEventsUseCase,
    private val subscribeOnClubsEventsUseCase:
SubscribeOnClubsEventsUseCase,
    private val startListenClubsUpdatesUseCase:
StartListenClubsUpdatesUseCase,
    subscribeOnMemberClubsUseCase:
SubscribeOnMemberClubsUseCase,
    subscribeOnAdminClubsUseCase: SubscribeOnAdminClubsUseCase,
) : BaseViewModel() {
    val isCameraTracking = mapController.isCameraTracking
    val navigationRoutes = mapController.navigationRoutes
    val userLocationCoordinates =
mapController.userLocationCoordinates
    private val userEvents =
subscribeOnUsersEventsUseCase.execute()
    private val _clubsEvents =
MutableStateFlow(listOf<Event>())
    private val clubsEvents = _clubsEvents.asStateFlow()
    private val adminClubs: Flow<List<Club>> =
subscribeOnAdminClubsUseCase.execute()
    private val memberClubs: Flow<List<Club>> =
subscribeOnMemberClubsUseCase.execute()
    private val clubsIds = adminClubs.combine(memberClubs) {
list1, list2 -> list1 + list2
    }.map { clubsList ->

```

Продовження лістингу A.2

```

        clubsList.map { club ->
            club.id
        }
    }.distinctUntilChanged()
    private var eventPickedLocation: Point? = null

    init {
        launch {
            startListenClubsUpdatesUseCase.execute() {
                onError = {causeError(it)}
            }
        }
        launch {
            userEvents.collectLatest { eventsList ->
                withContext(Dispatchers.Main) {
                    mapController.showUsersEventsMarkers(eventsList)
                }
            }
        }
        launch {
            clubsIds.filter { it.isNotEmpty() }
                .flatMapLatest { clubsIds ->
                    subscribeOnClubsEventsUseCase.execute(
                        param
                        =
                        SubscribeOnClubsEventsUseCase.Param(
                            clubsIds = clubsIds
                        )
                    )
                }.collectLatest { clubsEvents ->
                    _clubsEvents.value = clubsEvents
                }
        }
        launch {
            clubsEvents.collectLatest { eventsList ->
                withContext(Dispatchers.Main) {
                    mapController.showClubsEventsMarkers(eventsList)
                }
            }
        }
    }

    fun attachMapController(mapControllerDeclaration:
        MapControllerDeclaration) {

        mapControllerDeclaration.invoke(MapControllerAttachHelper(mapC
        ontroller = mapController))
    }

    fun detachMapController(mapControllerDeclaration:
        MapControllerDeclaration) {

```

Продовження лістингу A.2

```

mapControllerDeclaration.invoke(MapControllerAttachHelper(mapC
ontroller = mapController))
    }

    fun setCameraTracking() {
        mapController.setCameraTracking()
    }

    fun      setOnEventMarkerClick(onEventMarkerClickListener:
(event: Event) -> Unit) {

mapController.setOnEventMarkerClick(onEventMarkerClickListener
)
    }

    fun
setOnEventPositionPickerMarkerShow(onEventPositionPickerMarker
Show: (location: String) -> Unit) {
        mapController.setOnEventPositionPickerMarkerShow      {
point ->
            eventPickedLocation = point
            launch {
                getLocationUseCase.execute(
                    param = GetLocationUseCase.Param(
                        longitude = point.longitude(),
                        latitude = point.latitude()
                    )
                ) {
                    onSuccess = { location ->

onEventPositionPickerMarkerShow.invoke(location.toFullOneLineF
ormat())
                        }
                        onError = {
                            causeError(it)
                            launch(Dispatchers.Main) {
                                clearEventPositionPickerMarker()
                            }
                        }
                    }
                }
            }
        }
    }

fun navigateToCreateEvent() {
    eventPickedLocation?.let {
        router.execute(
            Command.Navigate(
                CreateEventScreen(
                    longitude = it.longitude(),
                    latitude = it.latitude(),
                    clubId = null

```

Продовження лістингу A.2

```

        )
    )
}
eventPickedLocation = null
}

fun clearEventPositionPickerMarker() {
    mapController.clearEventPositionPickerMarker()
}

fun clearMapMarkersAndRouteLines() {
    mapController.clearMarkers()
    mapController.clearRouteLines()
}

fun mapHasRouteLines(): Boolean {
    return navigationRoutes.value.isNotEmpty()
}

fun createNewRoute(destination: Point, destinationName:
String?) {
    launch {
        mapController.createNewRoute(
            destination = destination,
            destinationName = destinationName
        ).fold(
            onSuccess = { routes ->
                mapController.showRouteLines(routes)
            },
            onFailure = {
                _error.emit(it)
            }
        )
    }
}
}

```

Лістинг A.3 – Код класів ViewHolder

```

abstract class MessageViewHolder(
    view: View,
    private val onMessageScroll: (MessageItem) -> Unit
) : RecyclerView.ViewHolder(view) {

    open fun onBind(message: MessageItem) {
        onMessageScroll.invoke(message)
    }
}

```

Продовження лістингу А.3

```

class UserMessageViewHolder(
    private val binding: ListItemUserMessageBinding,
    private val onMessageScroll: (MessageItem) -> Unit,
    private val onItemClick: (MessageItem, View) -> Unit,
    private val case: Int
) : MessageViewHolder(binding.root, onMessageScroll) {

    override fun onBind(message: MessageItem) {
        super.onBind(message)

        if (case == MyMessageViewHolder.WITH_DATE_DIVIDER) {
            binding.dateDivider.dateDividerLayout.isVisible =
true
            binding.dateDivider.tvDate.text =
message.time.toddMMMyyyy()
        }

        binding.tvMessage.text = message.text
        binding.shortName.text = message.shortAuthorName
        binding.tvChatName.text = message.authorName
        binding.tvMessageTime.text = message.time.toHHmm()
        binding.layoutMessage.setOnClickListener {
            onItemClick.invoke(message, binding.layoutMessage)
        }
    }
    companion object {
        const val WITH_DATE_DIVIDER = 1
        const val WITHOUT_DATE_DIVIDER = 2
    }
}

class MyMessageViewHolder(
    private val binding: ListItemMyMessageBinding,
    private val onMessageScroll: (MessageItem) -> Unit,
    private val onItemClick: (MessageItem, View) -> Unit,
    private val case: Int
) : MessageViewHolder(binding.root, onMessageScroll) {

    override fun onBind(message: MessageItem) {
        super.onBind(message)

        if (case == WITH_DATE_DIVIDER) {
            binding.dateDivider.dateDividerLayout.isVisible =
true
            binding.dateDivider.tvDate.text =
message.time.toddMMMyyyy()
        }

        binding.tvMessage.text = message.text
        binding.shortName.text = message.shortAuthorName
        binding.tvMessageTime.text = message.time.toHHmm()
    }
}

```

Продовження лістингу А.3

```

        binding.layoutMessage.setOnClickListener {
            onItemClick.invoke(message, binding.layoutMessage)
        }
    }

    companion object {
        const val WITH_DATE_DIVIDER = 1
        const val WITHOUT_DATE_DIVIDER = 2
    }
}

```

Лістинг А.4 - Клас ChatRepositoryImpl

```

class ChatRepositoryImpl : ChatRepository {
    private val chatsCollection =
        Firebase.firestore.collection(CHATS_COLLECTION)
    private val messagesCollection =
        Firebase.firestore.collection(MESSAGES_COLLECTION)
    override suspend fun addChat(members: List<String>): String
    {
        val createdChat = ChatFb(
            members = members
        )
        val chatQuery =
            chatsCollection.add(createdChat).await()
        return chatQuery.id
    }
    override suspend fun updateChatMembers(
        chatId: String,
        chatMembersIdsList: List<String>
    ): Boolean {
        chatsCollection.document(chatId).update("members",
            chatMembersIdsList).await()
        return true
    }

    override suspend fun sendMessage(chatId: String, userId:
        String, text: String): Boolean {
        val message = MessageFb(
            chatId = chatId,
            authorId = userId,
            text = text,
            time = Date().time
        )
        messagesCollection.add(message).await()
        return true
    }
    override fun subscribeOnMessages(chatId: String, userId:
        String): Flow<List<Message>> {

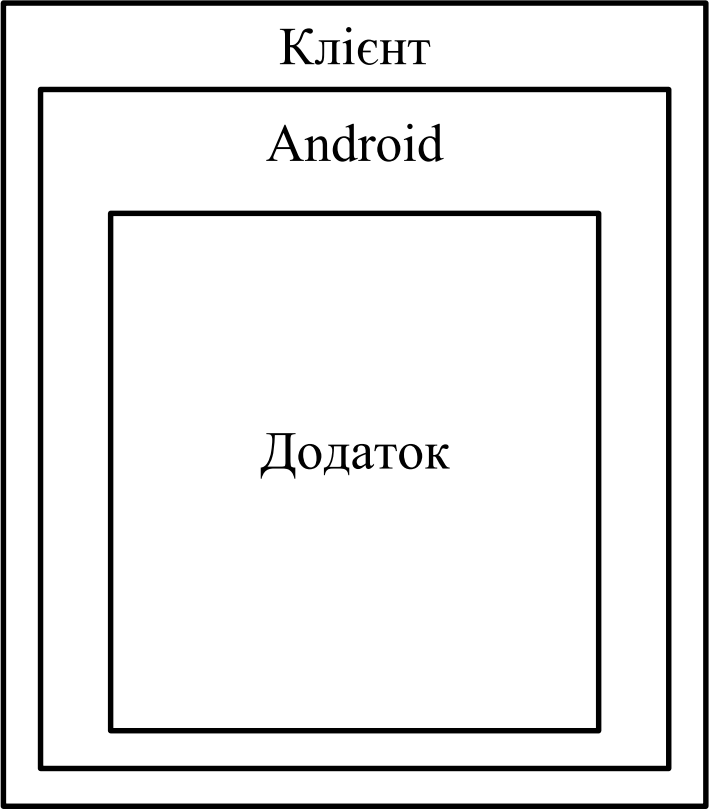
```

Продовження лістингу А.4

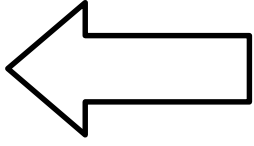
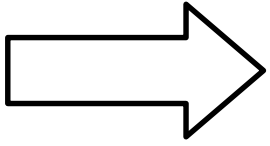
```

        return messagesCollection.whereEqualTo("chatId",
chatId).orderBy("time")
            .asFlow(closeOnError = true) {
                it.documents.map { document ->
                    val messageFb =
document.toObject<MessageFb>()
                    ?: throw FirebaseException("Can't map
club from backend")
                    messageFb.mapToMessage(messageId =
document.id, userId = userId)
                }
            }
    }
    private fun MessageFb.mapToMessage(messageId: String,
userId: String): Message {
        return Message(
            id = messageId,
            chatId = this.chatId,
            authorId = this.authorId,
            text = this.text,
            time = this.time,
            authorIsMe = this.authorId == userId
        )
    }
    companion object {
        const val CHATS_COLLECTION = "chats"
        const val MESSAGES_COLLECTION = "messages"
    }
}

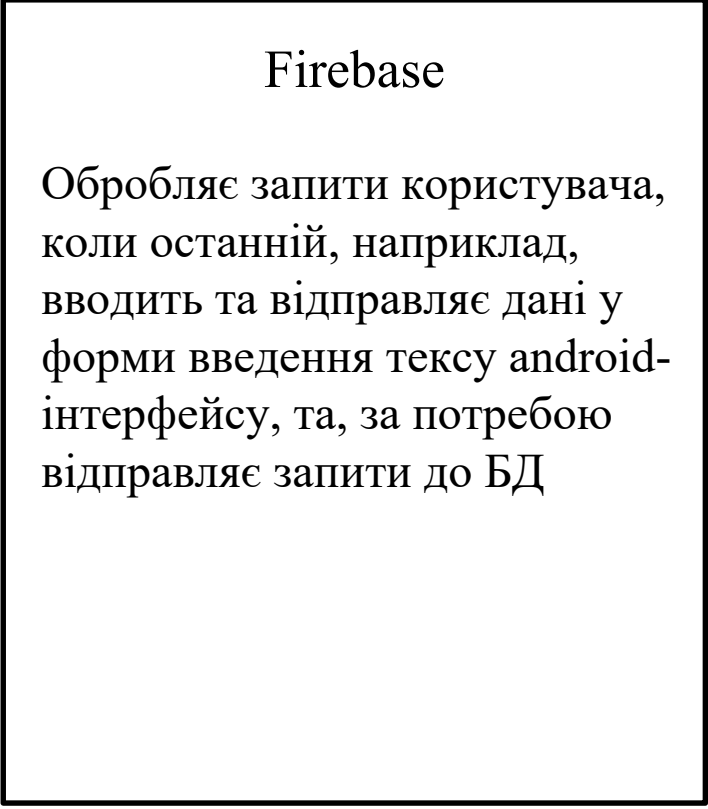
```



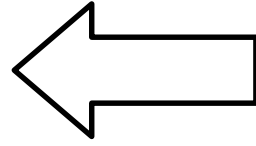
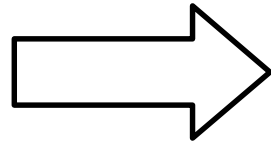
Клієнт
надсилає
запит



Сервер
повертає
відповідь



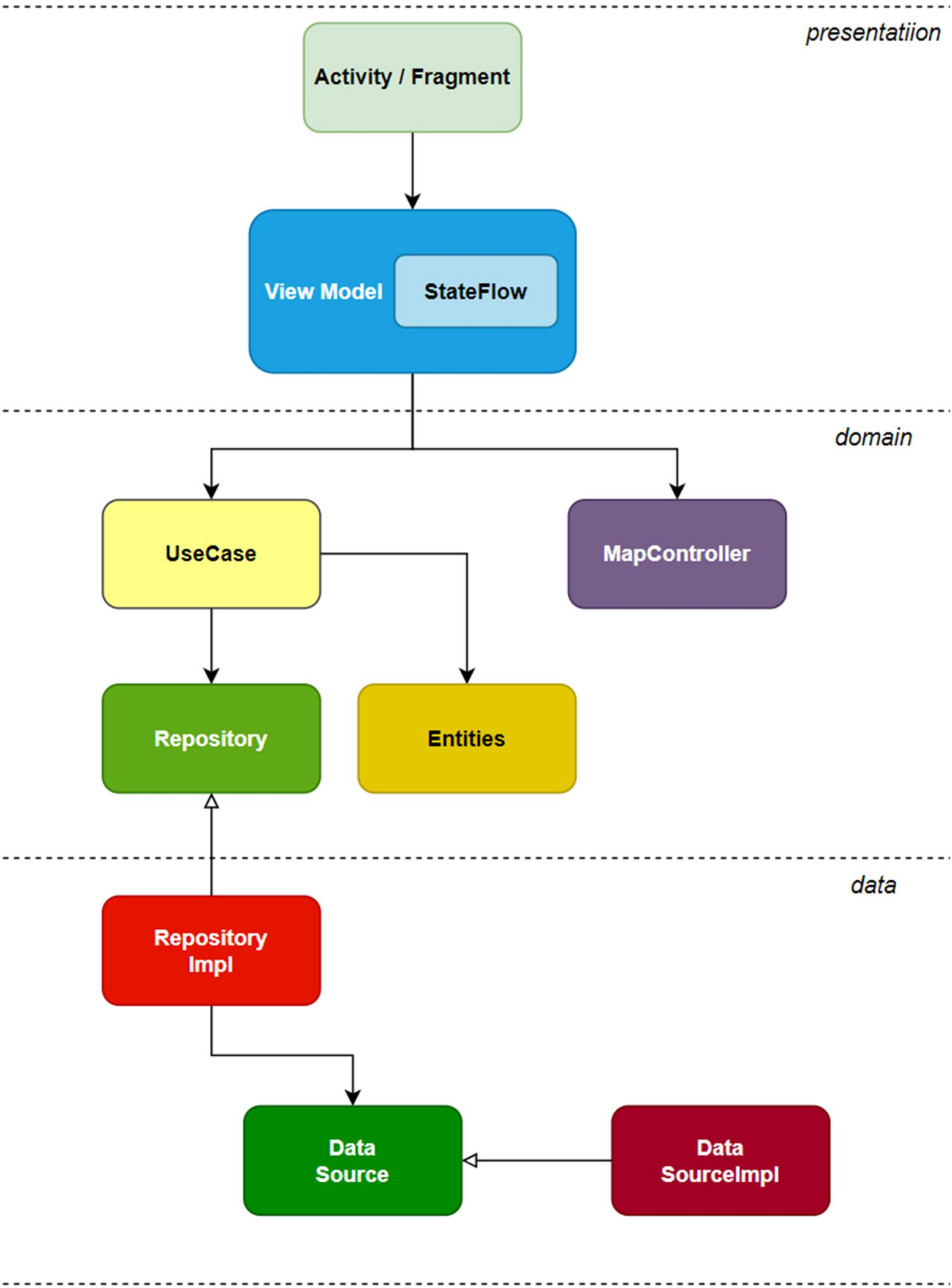
Надсилає
запит до
БД



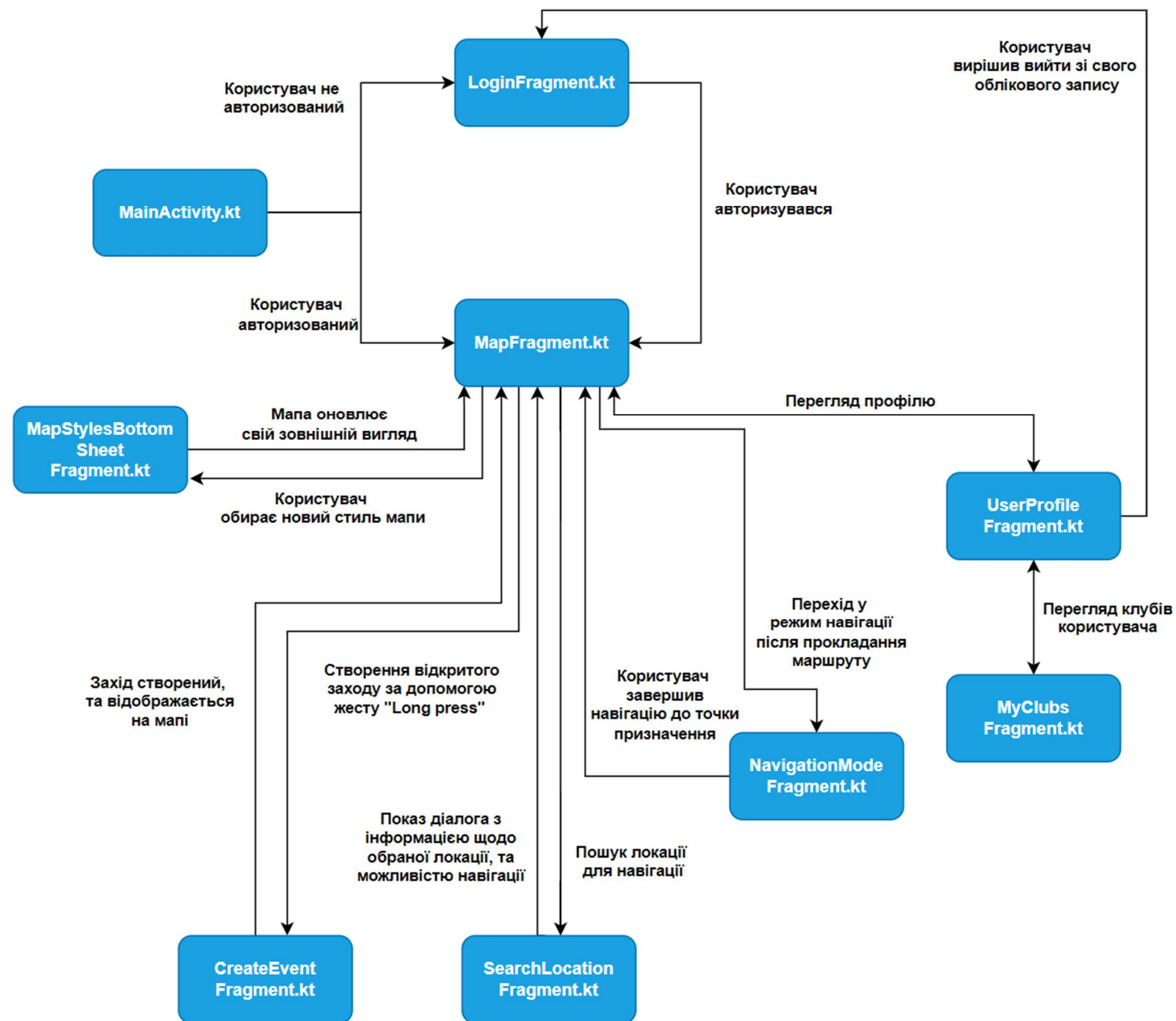
Сервер
отримує
результат
запиту



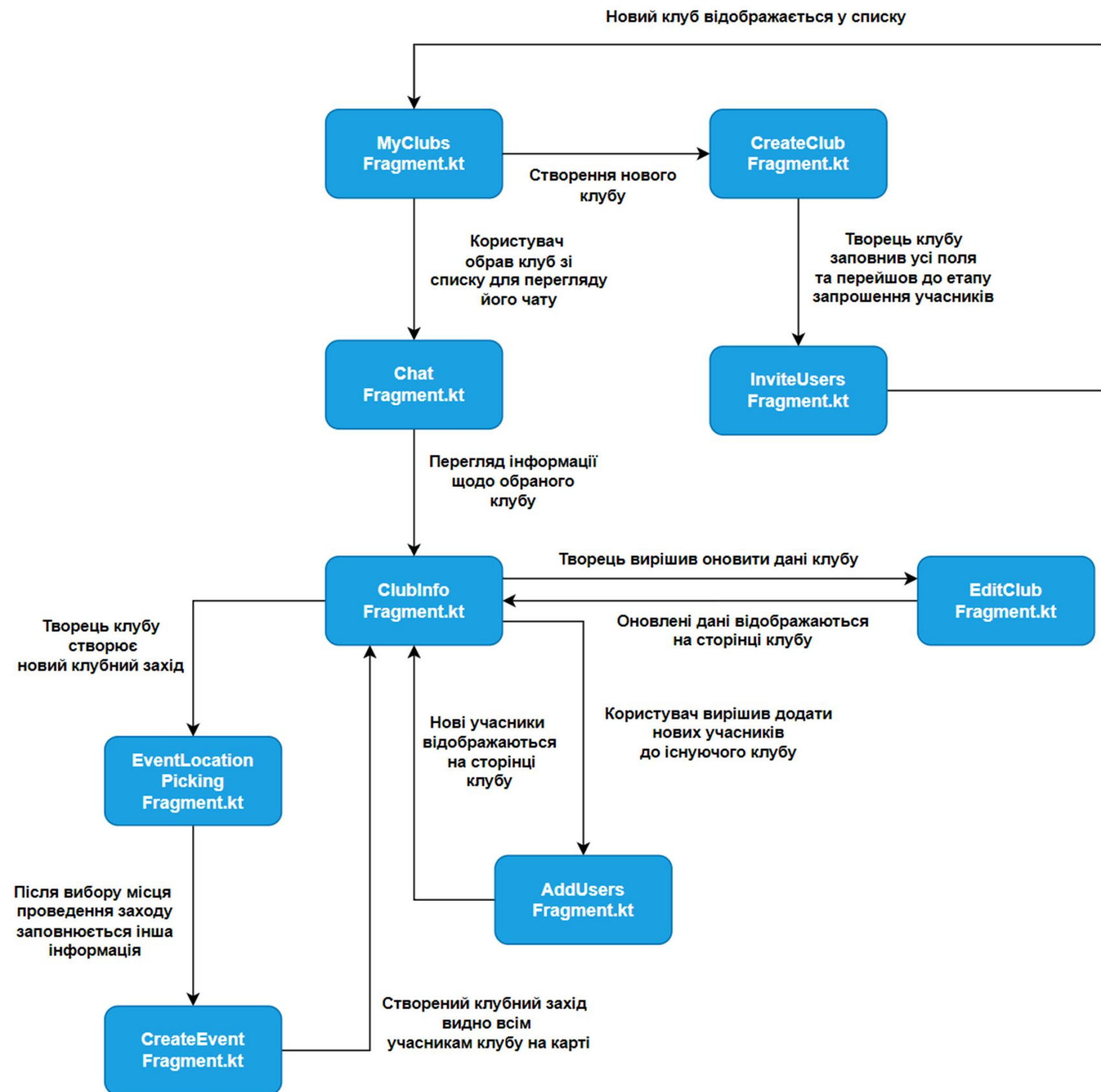
					13.02070849.51917 ПЗ								
					Розробка комп'ютерної системи на базі ОС Android для вуличних гонщиків Схема розподіленої системи				Лім.		Маса	Мірило	
Змін.	Арк.	№ док.	Підпис	Дата									
Розробив		Слонєв М.А.											
Перевірів		Скрупський С.Ю.											
Т.контр.									Аркуш 1		Аркушів 1		
									НУ "Запорізька політехніка" КНТ-519				
Н.контр.		Щербак Н.В.											
Затв.		Кудерметов Р.К.											



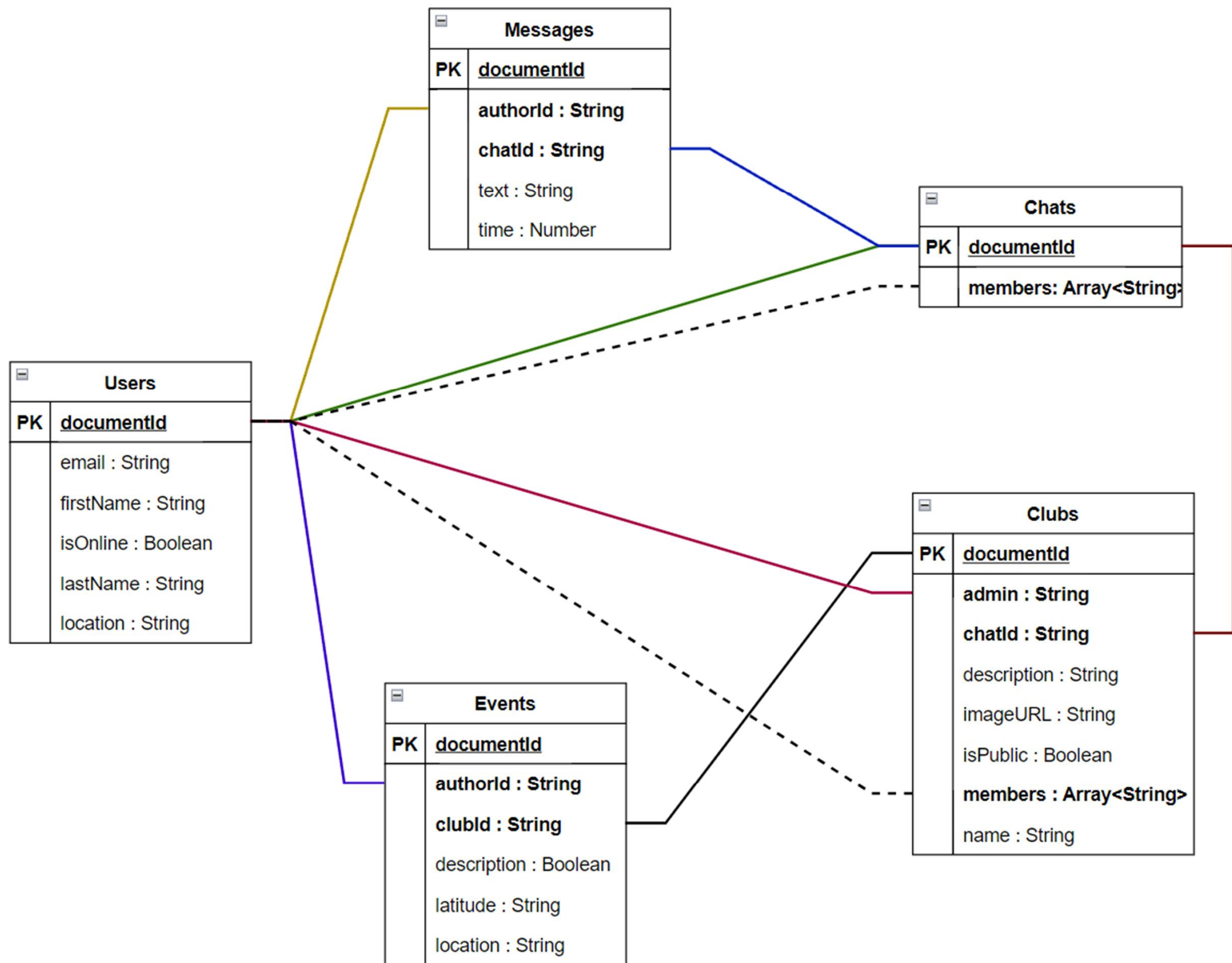
						13.02070849.51917 ПЗ			
						Розробка комп'ютерної системи на базі ОС Android для вуличних гонщиків Схема архітектури Android-додатку	Літ.	Маса	Мірило
Змін.	Арк.	№ док.	Підпис	Дата					
Розробив		Слонєв М.А.							
Перевірів		Скрупський С.Ю.							
Т.контр.							Аркуш 1	Аркушів 1	
Н.контр.		Щербак Н.В.					НУ "Запорізька політехніка" КНТ-519		
Затв.		Кудерметов Р.К.							



					13.02070849.51917 ПЗ				
					Розробка комп'ютерної системи на базі ОС Android для вуличних гонщиків Схема авторизації та взаємодії з мапою	Лім.		Маса	Мірило
Змін.	Арк.	№ док.	Підпис	Дата					
Розробив		Слонєв М.А.							
Перевірів		Скрупський С.Ю.							
Т.контр.									
						Аркуш 1		Аркушів 1	
Н.контр.		Щербак Н.В.				НУ "Запорізька політехніка" КНТ-519			
Замв.		Кудерметов Р.К.							



					13.02070849.51917 ПЗ			
					Розробка комп'ютерної системи на базі ОС Android для вуличних гонщиків	Лім.	Маса	Мірило
Змін.	Арк.	№ док.	Підпис	Дата				
Розробив		Слонєв М.А.			Схема використання соціальних функцій додатку			
Перевірів		Скрупський С.Ю.						
Т.контр.						Аркуш 1	Аркушів 1	
Н.контр.		Щербак Н.В.			НУ "Запорізька політехніка" КНТ-519			
Замв.		Кудерметов Р.К.						



					13.02070849.51917 ПЗ							
					Розробка комп'ютерної системи на базі ОС Android для вуличних гонщиків	Лім.			Маса		Мірило	
Змін.	Арк.	№ док.	Підпис	Дата								
Розробив		Слонєв М.А.										
Перевірів		Скрупський С.Ю.			Схема розробленої бази даних	Аркуш 1			Аркушів 1			
Т.контр.												
						НУ "Запорізька політехніка" КНТ-519						
Н.контр.		Щербак Н.В.										
Затв.		Кудерметов Р.К.										