ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти Бакалавр

на тему: «Проєктування мобільного додатку для пошуку тематичної інформації»

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
освітнього ступеня бакалавр
групи 126ІСТ_бд_2022[1](стн)
Устінов Є.О.
Керівник: Протас Н.М.
Рецензент: Муравльов В.В.

Полтава – 2024 року

ВСТУП

Актуальність. З розвитком Інтернету суспільство почало набувати нові навички або поліпшувати вже існуючі, зокрема у музичній сфері. Це призвело до появи та розвитку різноспрямованих подій та галузевих спільнот. На сьогоднішній день мобільні пристрої споживають значно більше трафіку, ніж будь-коли раніше. Згідно зі статистикою Cisco [1], до 2025 року планується майже 38 мільярдів пристроїв/підключень, серед яких 45% будуть мобільними. Очікується, що ця цифра лише зростатиме. Так само, як і 10 років тому, бізнес стрімко розвивався в Інтернеті, завдяки своїй популярності та актуальності, так і зараз відбувається розвиток мобільних платформ. Серед лідерів з продажу смартфонів можна відзначити пристрої на платформі Android. Також варто відзначити беззаперечний ріст ринку мобільних додатків, який у 2024 році збільшився [2].

Виходячи з усього вищезазначеного, актуальність розробки мобільних додатків є дуже високою. Ця робота направлена на створення мобільного додатка, який може допомогти людині знаходити тематичну інформацію: однодумців, місця та події у сфері музики.

Метою кваліфікаційної роботи є розробка мобільного додатка на платформі Android, призначеного для персоналізованого пошуку тематичної інформації у музичній галузі.

Відповідно до мети роботи необхідно вирішити наступні завдання:

1) провести аналіз предметної галузі;

2) виконати огляд існуючих рішень на ринку мобільних програм;

3) визначити вимоги до системи, що розробляється, і розробити варіанти її використання;

4) розробити архітектуру мобільного додатку;

5) розробити архітектуру бази даних;

6) спроєктувати інтерфейс мобільного додатку на платформі Android;

8) розробити план тестування системи.

Об'єкт дослідження – процеси проєктування програмних додатків з розподіленою архітектурою.

Предмет дослідження – проєктування та розробка додатку з використанням REST-сервісу для віддаленого доступу до сервера баз даних.

Методи досліджень – проведені в роботі дослідження базуються методах програмної інженерії, проєктування та нормалізації баз даних, розробки мобільних додатків з клієнт-серверною архітектурою.

Інформаційна база – Інтернет-ресурси та друковані видання, що місять інформацію про проєктування мобільних додатків, стандарти з розробки програмного забезпечення, довідкова інформація про використані програмні компоненти архітектури.

Практична значущість – на основі запропонованих архітектури, схеми бази даних, моделей і патернів та інтерфейсу, а також планів тестування спрощено реалізацію мобільного додатка, що виконує корисні функції пошуку тематичної інформації.

Результати роботи апробовані в рамках наукової конференції здобувачів вищої освіти за результатами науково-дослідної роботи у 2023-2024 роках.

Структура кваліфікаційної роботи логічно пов'язана з задачами досліджень і містить перелік умовних позначень, вступ, три розділи основної частини, висновки, список використаних джерел. Загальний обсяг текстової частини кваліфікаційної роботи складає 43 сторінки формату А4. Вона містить 12 рисунків і 3 таблиці.

РОЗДІЛ 1
АНАЛІЗ ВИМОГ ДО МОБІЛЬНИХ ДОДАТКІВ ДЛЯ ПРЕДМЕТНОЇ ОБЛАСТІ ПОШУКУ ТЕМАТИЧНОЇ ІНФОРМАЦІЇ

1.1 Аналіз та порівняння існуючих рішень для пошуку тематичної інформації в музичній галузі

На сьогоднішній день у Google Play та App Store існують кілька готових рішень на тему знаходження музичних однодумців та подій [3]. Ці додатки мають різний функціонал та способи реалізації основних можливостей. Серед типових функцій цих програм можна виділити налаштування профілю, можливість спілкування в чатах, а також пошук людей та груп за певними категоріями. Розглянемо деякі з них.

Додаток, «BandFriend», зображений на рис. 1.1, користується великою популярністю серед англомовних користувачів, що не дивно для продукту з Каліфорнії [4]. Зважаючи на це, можна стверджувати, що для нашого ринку цей додаток не є належним чином адаптованим та не займає значної частини. Однією з його найбільших переваг є можливість роботи в автономному режимі без підключення до Інтернету.

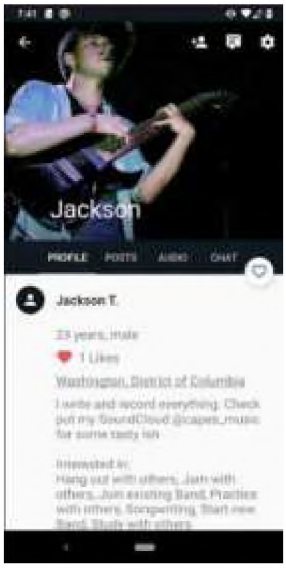


Рисунок 1.1 – Інтерфейс програми «BandFriend»

Особливістю додатка «Vampr», зображеного на рис.1.2, є його кросплатформенність, що означає його адаптованість під різні операційні системи [5]. Серед недоліків цієї програми можна відзначити відсутність україномовного користувацького співтовариства та обмеження пошуку людей у безкоштовній версії. Проте серед переваг виділяється якісний дизайн і використання зручного інтерфейсу користувача.

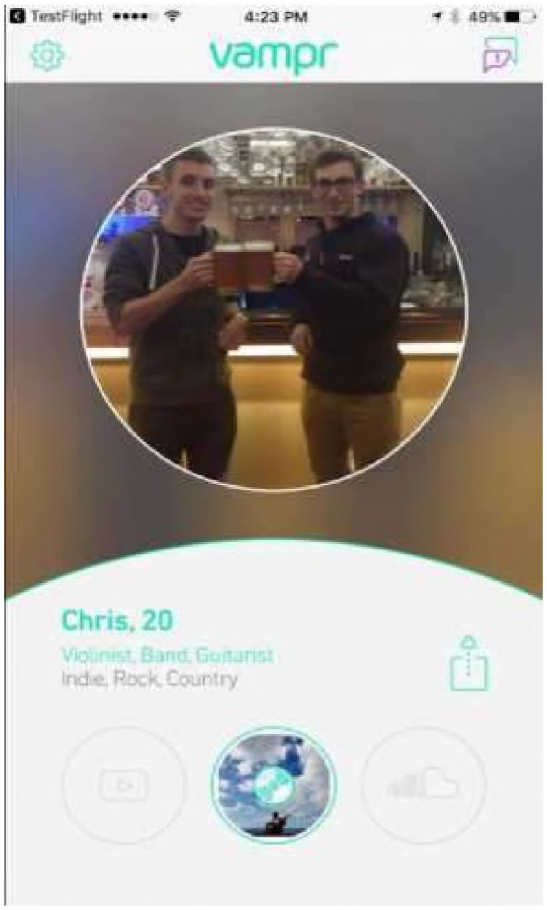


Рисунок 1.2 – Інтерфейс програми «Vampr»

TuneUp – це додаток, створений для любителів музики, які шукають однодумців для спільного музикування або відвідування концертів [6]. Він пропонує розширений пошук за музичними уподобаннями та географічним положенням. Користувачі можуть створювати профілі, в яких вони вказують інструменти, на яких вони грають, жанри, які їм подобаються, та рівень досвіду. Їм також можна приєднуватися до груп і форумів, щоб спілкуватися з іншими музикантами у своєму регіоні.

Рисунок 1.3 – Логотип додатку TuneUp

JamConnect – це додаток, який спеціалізується на знаходженні музикантів для співпраці або створення гурту [7]. Він надає можливість обмінюватися нотами, записами та організовувати репетиції. Користувачі можуть шукати музикантів за інструментом, жанром, рівнем досвіду та місцезнаходженням. Вони також можуть створювати профілі гурту та публікувати вакансії для музикантів.

Рисунок 1.4 – Логотип додатку JamConnect

Інші популярні програми для музикантів:

– BandLab: цей додаток дозволяє користувачам створювати музику разом в режимі реального часу. Він має вбудовані інструменти та ефекти, а також можливість співпрацювати з іншими користувачами над треками [8].

– Soundtrap: цей додаток є ще одним варіантом для створення музики разом. Він має просту у використанні інтерфейс і велику бібліотеку звуків та лупів [9].

– Nimbasi: цей додаток допомагає музикантам організувати свої репетиції, концерти та інші заходи. Він має календар, список завдань та інструменти для спілкування [10].

– Songkick: цей додаток допомагає музикантам знаходити концерти у своєму регіоні. Він також надає інформацію про місце проведення, склад групи та ціни на квитки [11].

Оглянувши ці додатки, можна зробити висновок, що ринок музичних платформ для знаходження однодумців та подій залишається актуальним для розробки. Кожен з цих додатків привертає користувачів своїм унікальним інтерфейсом та функціональністю. Також важливо відзначити, що жоден з них не має належної підтримки україномовної спільноти, що може стати додатковим стимулом для розробки нових додатків у цьому напрямку.

1.2 Аналіз існуючих рішень для реалізації проєкту

Android Studio є однією з найпопулярніших платформ для розробки додатків під Android [12]. Це завдяки ефективній співпраці між Google і Jetbrains, які забезпечують оперативну та якісну роботу цієї інтегрованої середовища розробки. У Android Studio можна вибрати мову програмування для розробки програми, зокрема Java або Kotlin. З урахуванням швидкого зростання популярності Kotlin, використання цієї мови може бути вигідним рішенням завдяки його синтаксису та продуктивності.

Для постійної актуальності даних у додатку важливо мати доступ до сервісу, що регулярно оновлює дані. Серед багатьох доступних API, KudaGo може бути хорошим вибором завдяки необмеженості у відправленні даних та легкості використання. Крім того, сам Android Studio має вбудований механізм

для конвертації JSON у потрібну колекцію для подальшої серіалізації. Нижче подано короткий огляд корисних для роботи бібліотек.

1. Hilt: бібліотека, що реалізує патерн ін'єкції залежностей [13]. Реалізує патерн Dependency Injection (DI) для спрощення та структуризації коду (рис. 1.5). Вона автоматично надає залежності вашим класам, роблячи їх більш гнучкими та тестуємими. Полегшує знаходження та використання залежностей, зменшуючи дублювання коду.

Рисунок 1.5 – Android hilt icon

2. Moxy: бібліотека, що реалізує патерн MVP (Model-View-Presenter), це бібліотека для створення чіткої архітектури Android-додатків [14]. Розділяє логіку інтерфейсу, моделі та презентера, роблячи код більш зрозумілим та керованим. Спрощує тестування та обслуговування коду.

3. Firebase: бібліотека Google, що надає можливості автентифікації, зберігання даних в хмарі, аналітики та багато іншого [15]. Дозволяє легко інтегрувати ці функції у Android-додаток.

4. Anko: бібліотека для декларативного опису UI Android за допомогою DSL (Domain Specific Language) [16]. Спрощує та ущільнює код опису інтерфейсу користувача. Зменшує дублювання коду та робить його більш читабельним.

5. Glide: ефективна бібліотека завантаження та відображення зображень для Android [17]. Вона автоматично кешує зображення, економлячи час та трафік, підтримує різні формати зображень та розширені функції, такі як закруглення кутів та завантаження GIF-файлів.

6. Serialization: бібліотека для серіалізації та десеріалізації даних JSON, що використовуються API [18]. Вона дозволяє легко конвертувати дані JSON в об'єкти Java/Kotlin та навпаки. Взаємодіє з популярними бібліотеками GSON, Moshi, Jackson.

7. Coroutines: бібліотека для асинхронної роботи з потоками, надає структурований спосіб для асинхронної роботи з потоками в Android, забезпечує більш чіткий та лаконічний код порівняно з AsyncTask [19]. Полегшує управління станом та обробку помилок в асинхронному коді.

8. Retrofit: популярна бібліотека для надсилання мережевих запитів HTTP/HTTPS з Android, створює динамічні проксі-об'єкти для API, спрощуючи роботу з ними. Підтримує різні типи запитів (GET, POST, PUT, DELETE) та обробку JSON.

9. Preferences: бібліотеки для роботи з налаштуваннями користувача в Android. Дозволяють зберігати та отримувати прості дані, такі як значення рядків, bool, int. Взаємодіє з SharedPreferences – стандартною бібліотекою Android для налаштувань [20].

10. Groupie: бібліотека для спрощеного управління RecyclerView в Android. Дозволяє декларативно описувати адаптери RecyclerView, використовуючи DSL. Зменшує шаблонний код та робить код більш читабельним.

11. Volley: легка та гнучка бібліотека для мережевої роботи з інтернетом в Android. Вона створює черги запитів, кешує дані та обробляє відповіді, проста у використанні та добре інтегрується з іншими бібліотеками [21].

12. CardStackView: бібліотека для реалізації інтерфейсу карткового макету, подібного до Tinder. Дозволяє створювати анімовані свайпи, натискання та інші взаємодії з картками. Містить вбудовані функції відстеження подій та налаштування.

13. MapKit: офіційна бібліотека Google для використання картографічних сервісів у Android-додатках, дозволяє інтегрувати карти Google, додавати маркери, маршрути та інші функції. Проста у використанні та добре документована [22].

14. EasyPermissions: бібліотека для спрощеного запиту та отримання небезпечних дозволів в Android. Автоматично перевіряє наявність дозволів, показує запити користувачеві та обробляє результати. Зменшує шаблонний код та робить роботу з дозволами більш зручною.

15. Room: офіційна бібліотека Google для роботи з локальною базою даних SQLite на Android. Створює абстракційний шар над SQLite, спрощуючи доступ до даних. Дозволяє легко виконувати CRUD-операції (Create, Read, Update, Delete) з об'єктами Java/Kotlin [23].

Вибір бібліотек залежить від конкретних потреб та проєкту. Важливо ретельно дослідити кожну бібліотеку, перш ніж використовувати її, щоб переконатися, що вона відповідає певним вимогам та добре інтегрується з кодом.

1.3 Аналіз вимог до мобільного додатку тематичного пошуку

Функціональні вимоги визначають функції, які повинна виконувати система та послуги, які вона повинна надавати. Нижче перераховані функціональні вимоги для мобільного додатку:

1) додаток повинен здійснювати збереження даних у базі даних;

2) додаток має кешувати дані;

3) Програма повинна автоматично отримувати нові дані та оновлювати користувацький інтерфейс;

4) додаток має отримувати дані зі стороннього сервісу (API);

5) проведення валідації всіх введених даних користувача;

6) відображення індикатора під час тривалого очікування завантаження системи;

7) перевірка авторизації користувача додатком;

8) можливість зміни даних профілю.

Нефункціональні вимоги визначають обмеження та характеристики системи, що впливають на її роботу та якість. Нижче подані нефункціональні вимоги для програми:

1) використання технології, сумісної з платформою Android;

2) розробка програми в середовищі розробки Android Studio;

3) реалізація додатку мовою програмування Kotlin;

4) авторизація користувачів за допомогою Firebase;

5) збереження даних в локальній базі даних Room та віддаленій базі даних Firebase;

6) виведення повідомлення при успішній реєстрації;

7) оновлення даних та видалення застарілих при підключенні до Інтернету;

8) відповідність системи патерну проєктування MVP;

9) написання програми мовою програмування Kotlin;

10) правильне відображення графічних елементів на різних пристроях;

11) відображення анімації завантаження при отриманні даних з API.

На основі вищезазначених функціональних та нефункціональних вимог до проєктованої системи можна зробити висновок, що розробка мобільного додатку потребує уважного врахування широкого спектру вимог щодо функціональності, технічних характеристик та взаємодії зі зовнішніми сервісами. Дотримання цих вимог сприятиме створенню високоякісного та ефективного програмного продукту, що задовольнить потреби користувачів та відповість сучасним стандартам розробки мобільних додатків.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ТА АРХІТЕКТУРИ МОБІЛЬНОГО ДОДАТКА

2.1 Проєктування варіантів використання мобільного додатка

У процесі проєктування програмного продукту важливим етапом є аналіз різноманітних сценаріїв використання та їх моделювання. В даному випадку використовується UML для побудови use-case діаграми, що дозволяє систематизувати та візуалізувати варіанти взаємодії користувачів з програмним забезпеченням.

Для проєктування мобільного додатка була використана мова графічного опису для об'єктного моделювання UML [24]. На рис. 2.1 показано діаграму варіантів використання.

Основні актори, що взаємодіють із системою:

1. Користувач (User) – особа, яка авторизується в системі;
2. Менеджер даних (Data Manager) – сутність, що отримує та відправляє дані;
3. Сервер (Server) – віддалений сервер;
4. База даних (Database) – локальна база даних.

Короткий опис варіантів використання:

1. Вхід (Login) – процедура авторизації в системі;
2. Надання прав (Authorization) – делегування особі або групі осіб прав на виконання певних дій;
3. Перевірка справжності (Authentication) – перевірка ідентифікації користувача шляхом порівняння введеного пароля з паролем у базі даних;
4. Реєстрація (Registration) – процес створення нового користувача;
5. Ідентифікація (Identification) – визначення ідентифікатора суб'єкта в системі;

6. Перегляд та створення чатів (Swipe) – інтеракція з користувачами та ініціювання створення чатів;
7. Перегляд карти (Map) – відображення інформації на карті;
8. Вибір чатів (Chat) – вибір чатів для спілкування;

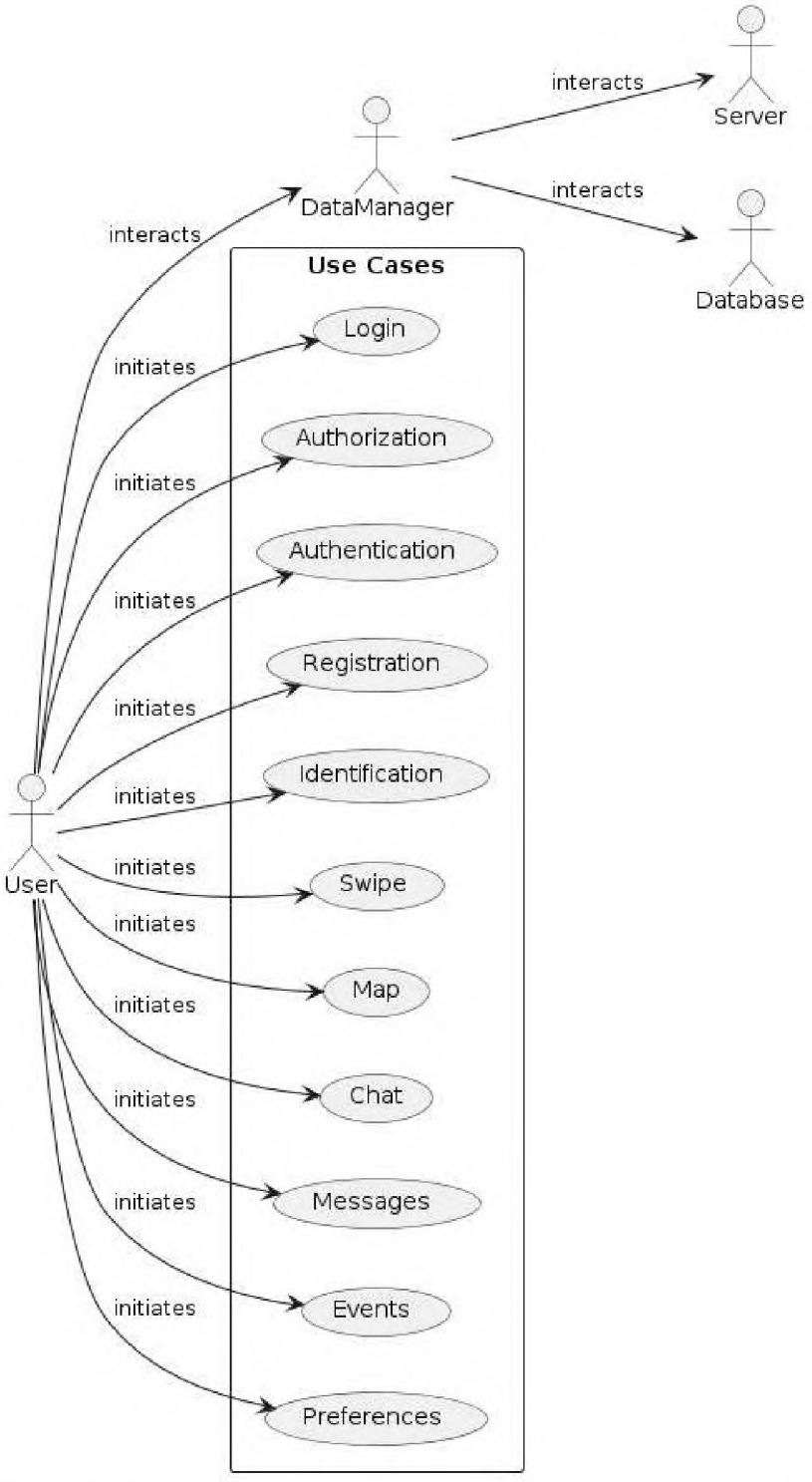


Рисунок 2.1 – Діаграма варіантів використання мобільного додатку для пошуку тематичної інформації

9. Надсилання та отримання повідомлень (Messages) – обмін повідомленнями між користувачами;

10. Перегляд майбутніх подій (Events) – перегляд запланованих подій;

11. Налаштування (Preferences) – зміна та збереження налаштувань користувача та пошуку.

Узагальнюючи вищевикладене, можна зазначити, що аналіз варіантів використання системи та їх моделювання є важливою складовою процесу проєктування програмного забезпечення. Використання UML для побудови use-case діаграми дозволяє систематизувати та зрозуміло відобразити можливі сценарії взаємодії користувачів з системою. Це сприяє розробці більш зрозумілого, ефективного та користувацько-орієнтованого програмного продукту.

2.2 Проєктування архітектури мобільного додатка

Мобільний додаток запитує дані в залежності від наявності підключення до Інтернету, доступності бази даних або API [25]. Це рішення дозволяє користуватися додатком навіть без доступу до Інтернету, при цьому забезпечуючи найбільш актуальну інформацію, якщо підключення все ж таки є. У випадку успішного запиту при наявності Інтернет-з'єднання, старі дані в базі даних автоматично оновлюються, щоб уникнути залишення застарілої інформації.

Мобільні додатки найкраще адаптуються під патерни проєктування MV*, оскільки це дозволяє ефективно розподілити обов'язки між різними складовими системи. Вибір MVP можна пояснити його простотою реалізації, особливо в порівнянні з MVC в середовищі Android Studio, тоді як VIPER може виявитися надто громіздким для невеликих додатків з обмеженим штатом розробників [26].

Користувач взаємодіє з інтерфейсом, який відображається на екрані, ініціюючи зміни, що передаються презентеру, що відповідає за управління

системою. За необхідності презентер також передає запити до моделі, де реалізована бізнес-логіка програми. Зміни в моделі автоматично відображаються у представленні системи через презентер. На рис. 2.2 цей патерн показаний більш наочно.

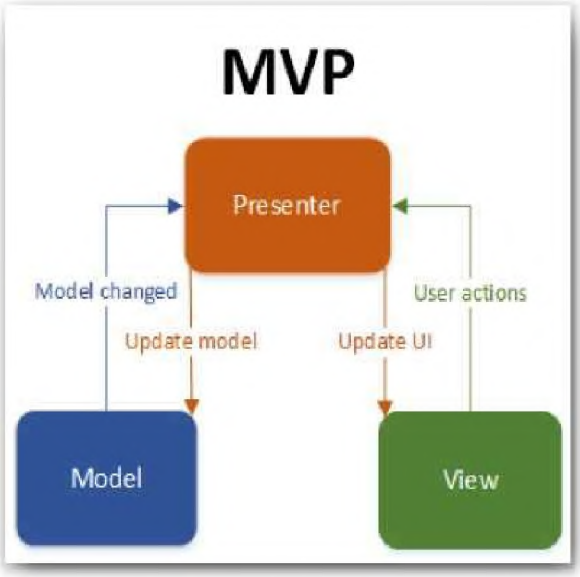


Рисунок 2.2 – Архітектура MVP

Також для підтримки чистоти коду було прийнято рішення реалізувати патерн Dependency Injection, який показаний на рис. 2.3 [27]. Суть його полягає у стилі налаштування об'єкта, при якому поля об'єкта задаються зовнішньою сутністю. Іншими словами, об'єкти налаштовуються за допомогою зовнішніх об'єктів.

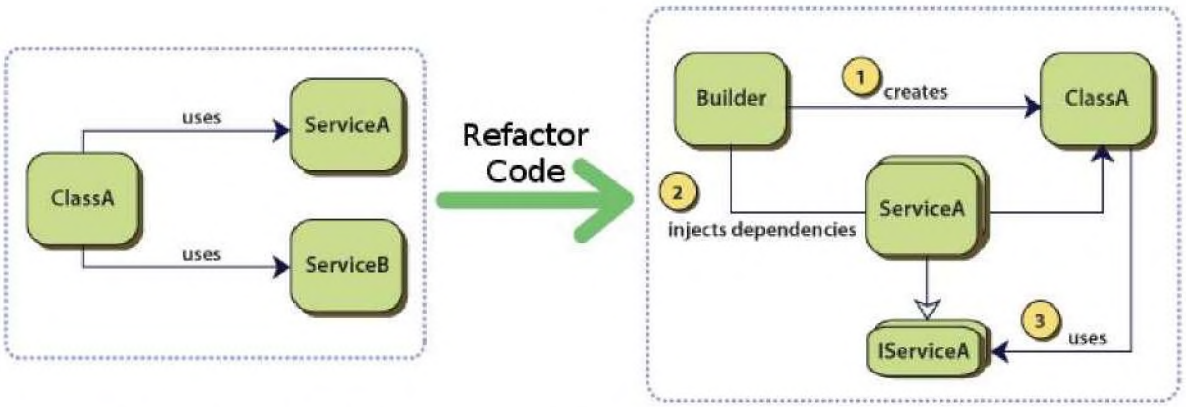


Рисунок 2.3 – Паттерн проєктування Dependency Injection

Реалізація механізму запиту даних у мобільному додатку, що залежить від умов наявності Інтернет з‘єднання або доступності бази даних, дозволяє забезпечити користувача актуальною інформацією, навіть у випадку відсутності Інтернету. Адаптація мобільних додатків під патерни проєктування MV* сприяє ефективному розподілу обов'язків та полегшує розробку, особливо у випадку використання MVP, яке є більш простою у реалізації порівняно з MVC або VIPER. Нарешті, взаємодія користувача з інтерфейсом через презентера та модель сприяє збереженню чистоти коду та ефективному управлінню системою.

2.3 Опис компонентів, що становлять систему

Компонентне проєктування системи полягає в розбитті програмного забезпечення на окремі компоненти, які можуть функціонувати незалежно один від одного. Це дозволяє досягти більшої модульності, зручності в розробці та тестуванні програми. Реалізовані класи для функціонування системи описані нижче.

Model, View, Presenter – сутності, що описують патерн MVP. Activity – це окремий екран Android. MainActivity – екран, з яким взаємодіє користувач. Fragment – це подібність Activity, яку ми можемо підключати в різні частини програми. Але одна Activity може містити декілька Fragment [28].

CalendarFragment, ChatFragment, SwipeFragment, MessageFragment, MapFragment, ProfileFragment, SignInFragment, SignUpFragment – це Fragment, що реалізує календар, чат, картки, повідомлення, карту, налаштування профілю, реєстрацію та авторизацію відповідно.

CalendarView, ChatView, MessageView, ProfileView, SignInView, SignUpView, SwipeView – інтерфейси, що описують календар, чат, повідомлення, налаштування профілю, реєстрацію та авторизацію відповідно.

SignInPresenter, SignUpPresenter, CalendarPresenter, ProfilePresenter, ChatPresenter, SwipePresenter – це Presenter, що описують реєстрацію, авторизацію та календар відповідно.

Chat – клас даних, що зберігає поля чатів. Message – інтерфейс для реалізації повідомлень. TextMessage – клас даних, що зберігає текстове повідомлення. ImageMessage – клас даних, що зберігає графічне повідомлення. Event – клас даних, що зберігає подію. User – клас даних, що зберігає поля користувача.

У підсумку, компонентне проєктування системи дозволяє створювати більш масштабовані та модульні додатки, що полегшує розробку, підтримку та розширення програмного забезпечення.

2.3 Проєктування бази даних мобільного додатку

Для роботи програми було розроблено бази даних, схема яких представлена на рис. 2.5. База даних системи складається із 8 таблиць. Розглянемо докладніше кожну з них.

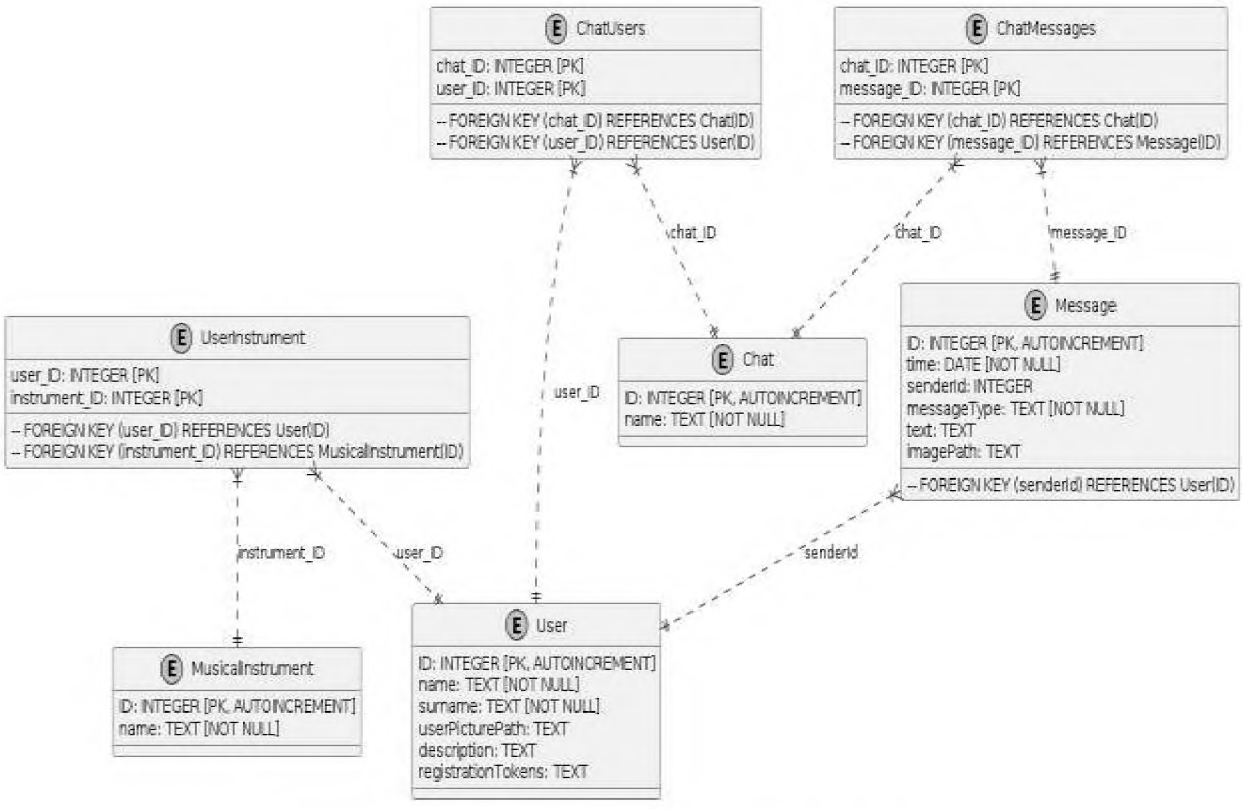


Рисунок 2.5 – Діаграма сутностей бази даних

Модель User зберігає інформацію про користувача. Її атрибути:

- ID – ідентифікатор користувача
- name – ім'я користувача
- surname – прізвище користувача
- userPicturePath – шлях до зображення профілю користувача
- description – опис користувача
- registrationTokens – токени користувача

Модель MusicalInstrument зберігає інформацію про музичні інструменти. Її атрибути:

- ID – ідентифікатор інструменту
- name – назва інструменту

Модель UserInstrument зберігає інформацію про музичні інструменти, якими володіє користувач. Її атрибути:

- user_ID – ідентифікатор користувача
- instrument_ID – ідентифікатор інструменту

Модель Chat зберігає інформацію про чати. Її атрибути:

- ID – ідентифікатор чату
- name – назва чату

Модель ChatUsers зберігає інформацію про чати, в яких бере участь користувач. Її атрибути:

- chat_ID – ідентифікатор чату
- user_ID – ідентифікатор користувача

Модель Message зберігає інформацію про повідомлення. Її атрибути:

- ID – ідентифікатор повідомлення
- time – час відправлення повідомлення
- senderId – ідентифікатор користувача, який надіслав повідомлення
- messageType – тип повідомлення
- text – текст повідомлення
- imagePath – шлях до зображення

Модель ChatMessages зберігає інформацію про повідомлення в чатах. Її атрибути:

- chat_ID – ідентифікатор чату
- message_ID – ідентифікатор повідомлення

Модель Event зберігає інформацію про події (на рис.2.5 не показана). Її атрибути:

- url – URL події
- name – назва події
- date – дата події
- participantCount – кількість учасників
- eventImageView – URL зображення події
- description – опис події

Розроблена база даних забезпечує збереження та організацію важливих даних для функціонування програми, включаючи інформацію про користувачів, чати, повідомлення, події та їх взаємозв'язки. Це дозволяє ефективно управляти даними та забезпечувати надійну роботу програми.

2.4 Опис діаграми діяльності для процесу входу і реєстрації в додатку

Діаграма діяльності відображає процес входу і реєстрації користувача в додатку із використанням сервісу Firebase для аутентифікації [29]. На діаграмі показані взаємодії між користувачем, додатком та Firebase, а також можливі шляхи і результати кожного кроку. UML діаграма наведена у додатку А.

Основні етапи:

1. Запуск додатку:
- Користувач запускає додаток для тематичного пошуку інформації.
- Додаток створює основну активність (MainActivity).
2. Перевірка авторизації:
- Якщо користувач вже авторизований, додаток продовжує свою роботу і завершує процес.

- Якщо користувач не авторизований, створюється фрагмент входу (SignInFragment).

3. Процес входу/реєстрації:

- Користувач перевіряє, чи має він обліковий запис.
- Якщо обліковий запис є, користувач вводить свої дані для входу.
- Якщо облікового запису немає, користувач вводить свої дані для реєстрації.
- Якщо користувач бажає створити обліковий запис, він вводить нові дані.

4. Введення даних:

- Додаток створює фрагмент реєстрації (SignUpFragment) та користувач вводить свої дані.

5. Перевірка введених даних:

- Додаток перевіряє, чи є введені дані валідними.
- Якщо дані не валідні, виводиться повідомлення про помилку.

6. Перевірка доступу до інтернету:

- Якщо є доступ до інтернету, дані передаються до Firebase для аутентифікації.
- Якщо доступу до інтернету немає, виводиться повідомлення про помилку відсутності інтернету.

7. Реєстрація/авторизація:

- Якщо реєстрація або авторизація успішна, користувач може продовжити роботу з додатком.
- Якщо реєстрація або авторизація неуспішна, виводиться повідомлення про помилку.

Взаємодії між компонентами:

- Користувач: взаємодіє з додатком, запускає його, вводить свої дані для входу або реєстрації.

- Додаток для тематичного пошуку інформації: обробляє введені дані користувача, створює необхідні активності та фрагменти, перевіряє валідність даних і доступ до інтернету.
- Firebase: використовується для перевірки даних користувача і забезпечення процесу аутентифікації.

Діаграма діяльності детально показує, як додаток для тематичного пошуку інформації взаємодіє з користувачем і Firebase для забезпечення безпечного входу або реєстрації, а також які кроки і перевірки здійснюються в процесі.

Таким чином, на основі виявлених вимог було спроєктовано архітектуру системи. Були виявлені компоненти системи та її основні елементи. Було спроєктовано архітектуру бази даних та визначено принцип взаємодії компонентів системи. Подальша реалізація системи спиратиметься на розроблену архітектуру.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ТЕМАТИЧНОГО ПОШУКУ ІНФОРМАЦІЇ

3.1 Опис реалізації моделей системи

У розробленому мобільному додатку було спроєктовано дев'ять моделей, які використовуються для зберігання та передачі даних, а також для забезпечення взаємодії з користувачами. Окрім того, для реалізації залежностей (DI) та роботи з API було створено п'ять додаткових моделей.

Діаграма класів, що описує спроєктовані моделі показана на рис. 3.1.

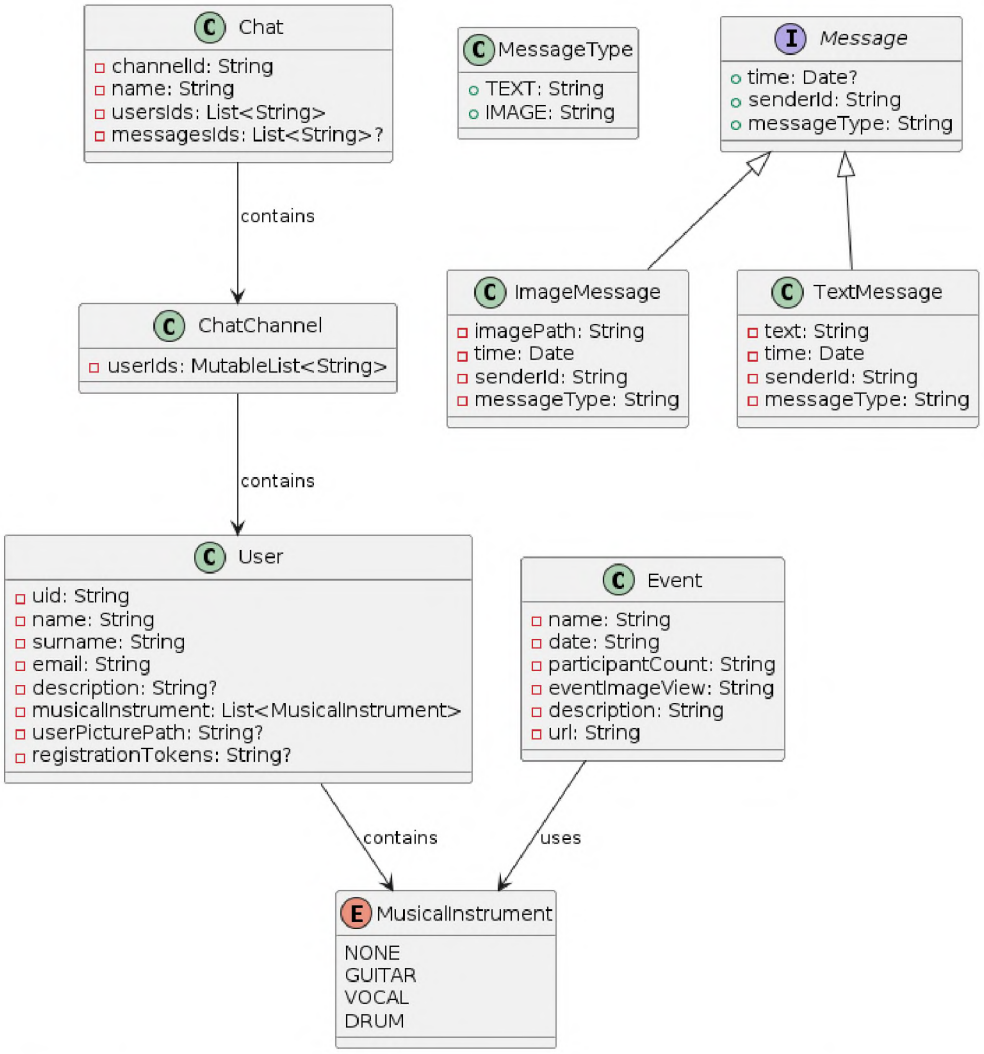


Рисунок 3.1 – Діаграма класів, що реалізує моделі взаємодії з користувачами

Моделі для реалізація класів даних наступні.

1. Chat (Чат): зберігає інформацію про конкретний чат, включаючи ідентифікатор каналу, назву, список ідентифікаторів користувачів та повідомлень. Має поля:

– channelId: ідентифікатор каналу (рядок).

– name: назва чату (рядок).

– usersIds: список ідентифікаторів користувачів, які беруть участь у чаті (список рядків).

– messagesIds: список ідентифікаторів повідомлень у чаті, може бути null (список рядків або null).

2. Event (Подія): описує подію з її основними характеристиками, такими як назва, дата, кількість учасників тощо. Цей клас також може бути серіалізованим та передаваним між компонентами (Parcelable). Має поля:

– name: Назва події (рядок).

– date: Дата події (рядок).

– participantCount: Кількість учасників (рядок).

– eventImageView: URL зображення події (рядок).

– description: Опис події (рядок).

– url: Ідентифікатор події, також використовується як первинний ключ у базі даних (рядок).

3. MessageType (Тип Повідомлення): об'єкт, що містить константи для визначення типу повідомлень. Має константи:

– TEXT: Текстове повідомлення.

– IMAGE: Зображення.

– 4. Message (Повідомлення): задає структуру для повідомлень, включаючи час, ідентифікатор відправника та тип повідомлення. Має методи:

– val time: Час відправлення повідомлення (об'єкт Date або null).

– val senderId: Ідентифікатор відправника (рядок).

– val messageType: Тип повідомлення (рядок).

5. ImageMessage (Повідомлення із Зображенням): реалізує повідомлення, що містить зображення. Має поля:

– imagePath: Шлях до зображення (рядок).

– time: Час відправлення (об'єкт Date).

– senderId: Ідентифікатор відправника (рядок).

– messageType: Тип повідомлення, за замовчуванням MessageType.IMAGE (рядок).

6. TextMessage (Текстове Повідомлення): реалізує текстове повідомлення. Має поля:

– text: Текст повідомлення (рядок).

– time: Час відправлення (об'єкт Date).

– senderId: Ідентифікатор відправника (рядок).

– messageType: Тип повідомлення, за замовчуванням MessageType.TEXT (рядок).

7. MusicalInstrument (Музичний Інструмент): перерахування можливих музичних інструментів, якими може володіти користувач. Має константи:

– NONE: Без інструменту.

– GUITAR: Гітара.

– VOCAL: Вокал.

– DRUM: Барабан.

8. User (Користувач): описує профіль користувача з основною інформацією про нього, включаючи ідентифікатор, ім'я, прізвище, email, опис, музичні інструменти тощо. Має поля:

– uid: Ідентифікатор користувача (рядок).

– name: Ім'я (рядок).

– surname: Прізвище (рядок).

– email: Email (рядок).

– description: Опис користувача (рядок або null).

– musicalInstrument: Список музичних інструментів, якими володіє користувач (список MusicalInstrument).

– userPicturePath: Шлях до фотографії користувача (рядок або null).

– registrationTokens: Токени для реєстрації користувача (рядок або null).

9. ChatChannel (Канал Чату): зберігає список ідентифікаторів користувачів, які беруть участь у каналі чату. Має поле:

– userIds: Список ідентифікаторів користувачів (список рядків).

Діаграма класів, що описує спроєктовані моделі API та DI показана на рис. 3.2.

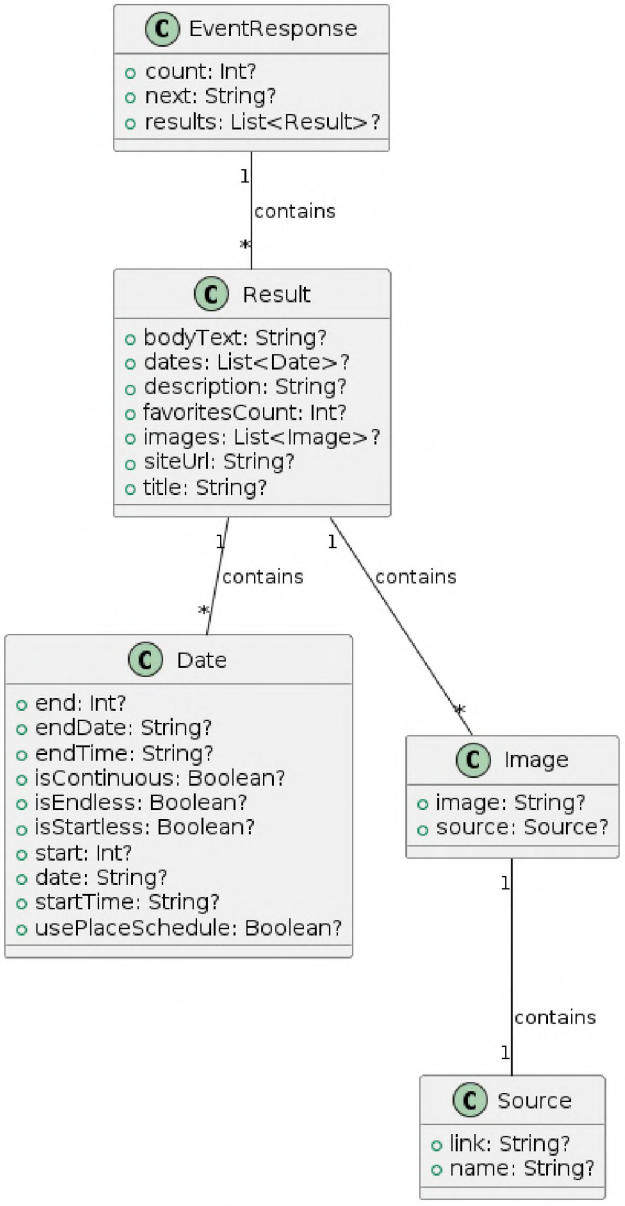


Рисунок 3.2 – Діаграма класів, що реалізує моделі API та DI

Моделі API та DI наступні:

1. Date (Дата): описує структуру даних для дати, яка містить інформацію про початок, кінець, та інші характеристики події. Має поля:

– end: Час закінчення події (ціле число або null).

– endDate: Дата закінчення події (рядок або null).

– endTime: Час закінчення події (рядок або null).

– isContinuous: Показує, чи є подія безперервною (булеве значення або null).

– isEndless: Показує, чи подія нескінченна (булеве значення або null).

– isStartless: Показує, чи подія без початку (булеве значення або null).

– start: Час початку події (ціле число або null).

– date: Дата початку події (рядок або null).

– startTime: Час початку події (рядок або null).

– usePlaceSchedule: Показує, чи використовується розклад місця (булеве значення або null).

2. EventResponse (Відповідь на Запит Події): структура даних для відповіді на запит про події, що містить інформацію про кількість подій, наступну сторінку результатів та список результатів. Має поля:

– count: Кількість подій (ціле число або null).

– next: URL наступної сторінки з результатами (рядок або null).

– results: Список результатів (список Result або null).

3. Image (Зображення): Описує структуру зображення, що включає шлях до зображення та джерело зображення. Має поля:

– image: Шлях до зображення (рядок або null).

– source: Джерело зображення (об'єкт Source або null).

4. Result (Результат): містить інформацію про результати події, такі як текст опису, дати, зображення, URL сайту тощо. Має поля:

– bodyText: Текст опису події (рядок або null).

– dates: Список дат події (список Date або null).

– description: Опис події (рядок або null).

– favoritesCount: Кількість збережень в обране (ціле число або null).

– images: Список зображень, пов'язаних з подією (список Image або null).

– siteUrl: URL сайту події (рядок або null).

– title: Назва події (рядок або null).

5. Source (Джерело): описує джерело даних або зображень, яке включає посилання та назву джерела. Має поля:

– link: Посилання на джерело (рядок або null).

– name: Назва джерела (рядок або null).

Запропоноване проєктне рішення охоплює структуру та призначення кожної моделі, забезпечуючи чітке розуміння їхньої ролі та функціоналу в додатку.

3.2 Проєктування інтерфейсу мобільного додатку

Спроєктовані інтерфейси та фрагменти для кількох ключових функціональних елементів мобільного додатку використовують бібліотеку Moxy для управління інтерфейсом користувача (UI). Moxy забезпечує гнучке управління життєвим циклом компонентів UI та дозволяє зручно управляти станом і поведінкою екранних елементів через спеціально створені інтерфейси та фрагменти [30].

Основні компоненти та їх функції спроєктованого інтерфейсу додатку наступні.

1. Інтерфейс календаря та його реалізація: інтерфейс CalendarView успадковує MvpView і визначає кілька методів, які визначають поведінку календаря:

– showLoading(isShow: Boolean) – показує або ховає індикатор завантаження.

– setEvents(events: List<Event>) – встановлює список подій для відображення.

– setTimestamp(timestamp: Long) – зберігає таймстемп для подальшого використання.

Фрагмент CalendarFragment реалізує CalendarView. Він використовує анотацію @AndroidEntryPoint, що дозволяє використовувати залежності через DI (Dependency Injection), здійснює ініціалізацію адаптера CalendarAdapter для відображення подій у RecyclerView та реалізує методи setTimestamp, showLoading і setEvents для роботи з подіями та станом завантаження. Метод isInternetAvailable перевіряє наявність інтернет-з'єднання.

2. Адаптер для календаря: клас CalendarAdapter використовується для відображення списку подій у вигляді елементів RecyclerView, реалізує ListAdapter для роботи з елементами типу Event і включає методи onCreateViewHolder і onBindViewHolder для створення та заповнення елементів списку. Клас CalendarAdapter використовує Glide для завантаження та відображення зображень подій.

3. Інтерфейс та реалізація чатів: інтерфейс ChatView визначає методи для роботи з інтерфейсом чату:

– updateRecyclerView(items: List<Item>) – оновлює список елементів у RecyclerView.

– toMessageFragment(secondUser: User, item: ChatItem) – переходить до фрагмента повідомлень для конкретного чату.

Фрагмент ChatFragment реалізує ChatView та включає ініціалізацію RecyclerView для відображення списку чатів, використання FirestoreUtil для отримання даних чату та слухача для оновлення списку повідомлень та реалізацію методу toMessageFragment для переходу до фрагмента повідомлень.

4. Клас ChatItem для елементів чату представляє окремий елемент чату та містить метод bind для прив'язки даних до елементів ViewHolder, завантаження зображення користувача за допомогою GlideApp та методи equals, hashCode, getLayout, isSameAs для визначення унікальності елементів та їх відображення.

5. Інтерфейс та реалізація карти: інтерфейс MapView успадковує MvpView і використовується для роботи з картою. Фрагмент MapFragment реалізує MapView і включає ініціалізацію MapKitFactory для роботи з картами, використання EasyPermissions для запиту дозволів на доступ до геолокації та реалізацію методів onStart та onStop для управління станом карти при відкритті та закритті фрагмента.

6. Інтерфейс повідомлень та їх реалізація: інтерфейс MessageView визначає методи для роботи з інтерфейсом повідомлень:

– setSettingsFragment() – встановлює фрагмент налаштувань.

– updateRecyclerView(items: List<Item>) – оновлює список повідомлень.

Фрагмент MessageFragment реалізує MessageView та включає реалізацію методу updateRecyclerView для оновлення списку повідомлень, методи для відправки повідомлень та зображень, використовуючи FirestoreUtil та StorageUtil, метод sendNotification для надсилання повідомлень користувачам.

7. Клас MessageItem – це абстрактний клас, що слугує базовим класом для різних типів повідомлень. Він містить спільну функціональність для роботи з текстовими та графічними повідомленнями. Абстрактний клас MessageItem не може бути створений напряму, і призначений для наслідування іншими класами.

– Метод bind: Використовується для заповнення даних повідомлення у ViewHolder.

– Методи setTimeText та setMessageRootGravity встановлюють текст часу та налаштовують розташування кореневого елементу повідомлення в залежності від відправника.

8. Клас для текстового повідомлення TextMessageItem – це клас, який наслідує MessageItem і реалізує логіку для текстових повідомлень. Крім базового контенту, визначає текст повідомлення.

– Метод `bind: встановлює текст повідомлення у відповідний ViewHolder.

– Методи equals та hashCode: використовуються для порівняння об’єктів та генерації хеш-кодів.

9. Клас для графічного повідомлення ImageMessageItem – це клас, який наслідує MessageItem і реалізує логіку для графічних повідомлень. Завантажує та відображає зображення повідомлення.

– Метод `bind: Використовує Glide для завантаження зображення у ViewHolder.

– Методи equals та hashCode: Подібні до реалізації у TextMessageItem.

10. Інтерфейс налаштувань ProfileView – це інтерфейс, який визначає методи для роботи з профілем користувача. Його метод setSettingsFragment викликається для відображення фрагмента налаштувань профілю.

11. Фрагмент профілю ProfileFragment – це реалізація ProfileView, яка відповідає за роботу з профілем користувача. Забезпечує зміну зображення профілю, збереження даних користувача, перехід до фрагменту налаштувань. Використовує Glide для завантаження зображень у профіль користувача.

12. Інтерфейси реєстрації та авторизації SignInView і `SignUpView. Ці інтерфейси визначають методи для реєстрації та авторизації користувачів. Вони включають методи для навігації між фрагментами реєстрації та авторизації. Також визначають методи для відображення повідомлень про помилки.

13. Адаптер та інтерфейс для карток користувачів SwipeView і клас CardStackAdapter реалізують функціонал для відображення та управління картками користувачів. SwipeView визначає метод setData для встановлення даних користувачів. CardStackAdapter відповідає за відображення карток користувачів у RecyclerView.

14. Фрагмент карток користувачів клас SwipeFragment – це фрагмент для роботи з картками користувачів. Відповідає за відображення карток, управління жестами, обробку подій swipe (перегортання). Включає логіку для відображення повідомлень про відсутність інтернету.

15. Інтерфейс головного екрана MainView визначає методи для управління головним екраном додатка. Також включає методи для відображення панелі навігації, кнопок, прогрес-бару.

16. Головна активність MainActivity – це реалізація MainView, яка управляє головним екраном додатка. Відповідає за налаштування стартового фрагмента, обробка подій навігації, показ прогрес-бару та інших елементів. Включає логіку для перемикання між різними фрагментами, такими як чат, карта, календар, профіль.

Цей проєкт описує структуру мобільного додатка, де використані різні патерни проєктування, такі як шаблон MVP (Model-View-Presenter), для організації коду та управління інтерфейсом користувача. Використання абстрактних класів і інтерфейсів забезпечує гнучкість і зручність у підтримці та розширенні функціоналу додатка.

3.3 Проєктування презентерів системи

У проєкті запланована реалізація декількох презентерів, які відповідають за різні функціональні частини додатка. Презентери виконують роль посередників між моделлю даних та інтерфейсом користувача (View), забезпечуючи логіку обробки даних та їх відображення на екрані.

Усі презентери успадковуються від класу MvpPresenter, до якого передається інтерфейс виду (MvpView), що визначає методи для взаємодії з інтерфейсом користувача. Це забезпечує чітке розділення логіки обробки даних (Presenter) та їх відображення (View). Багато класів використовують анотацію @Inject, що вказує на ін'єкцію залежностей за допомогою фреймворку Dagger або подібного інструмента. Більшість презентерів використовують CoroutineExceptionHandler та presenterScope.launch для асинхронної обробки даних та управління потоками. Кожен презентер взаємодіє зі своїм інтерфейсом виду (CalendarView, ChatView, тощо), що визначає методи для оновлення інтерфейсу користувача. Опис реалізацій презентерів наведено нижче.

Клас CalendarPresenter реалізує презентер для календаря. Він використовує GetEventsUseCase та GetDatabaseUseCase для отримання даних подій. Метод

класу onFirstViewAttach викликається при першому приєднанні виду, завантажує події та оновлює інтерфейс користувача. Метод selectDatasource вибирає джерело даних залежно від наявності інтернет-з'єднання та часу останнього оновлення.

Клас ChatPresenter реалізує презентер для чату. Містить метод onItemClick, який обробляє клік по елементу чату, витягує дані про іншого користувача та переходить до фрагменту повідомлень.

Класи MapPresenter і MessagePresenter реалізують презентери для карти та повідомлень. MapPresenter це простий презентер для взаємодії з інтерфейсом карти (MapView). MessagePresenter це презентер для управління повідомленнями (MessageView).

Реалізація презентера для налаштувань профілю користувача виконана у класі ProfilePresenter. Він містить метод saveData, який зберігає дані профілю та налаштування пошуку користувачів у Firestore.

SignInPresenter відповідає за реалізацію презентера для авторизації. Містить наступні методи:

– метод signIn: Виконує авторизацію користувача через Firebase, перевіряє верифікацію email та оновлює токени для повідомлень.

– метод isEmailValid: Перевіряє правильність введеного email.

– метод onBtnRegistrationClicked: Перехід до фрагмента реєстрації.

Реалізація презентера для реєстрації користувачів SignUpPresenter містить наступні методи:

– метод isEmailValid: Перевіряє правильність введеного email.

– метод isPasswordValid: Перевіряє правильність введеного пароля.

– метод checkData: Перевіряє введені дані користувача, створює новий обліковий запис та зберігає його у Firestore.

Проєкт мобільного додатка демонструє реалізацію презентерів для різних функціональних частин мобільного додатка, використовуючи архітектурний патерн MVP. Це забезпечує чітке розділення відповідальностей між логікою

обробки даних та відображенням інтерфейсу, що полегшує підтримку та розширення додатка.

3.4 Проєктування Dependency Injection та Use Case, взаємодії з Firebase та сервісу push up повідомлень

Для реалізації DI був реалізований один інтерфейс, три модулі, один use case та один клас. Також для коректної роботи кожен Activity і Fragment повинен починатися з @AndroidEntryPoint і створювати презентер через @Inject. У самому презентері також повинен бути конструктор @Inject. Код для впровадження залежностей Retrofit та Room вимагає створення окремих класів. Firebase має багато сервісів. При проєктуванні використано Firestore, Authentication, Storage і Cloud Messaging. Для отримання push up повідомлень потрібно створити сервіс, щоб процес працював незалежно від роботи додатку. Щоб надіслати повідомлення, потрібно звернутися до Firebase Cloud Messaging.

Впровадження Залежностей (DI) реалізовано через один інтерфейс та три модулі для впровадження залежностей. Один з модулів – це GlideModule, який забезпечує роботу з зображеннями, використовуючи бібліотеку Glide. Реалізовано один use case та один клас. Це частина архітектурного підходу, де бізнес-логіка винесена в окремі компоненти. Для коректної роботи кожен Activity і Fragment повинен починатися з @AndroidEntryPoint, а презентери створюються через @Inject. У презентерах конструктор теж має бути відзначений анотацією @Inject. Для роботи з Retrofit та Room використовуються окремі класи. Використовуються сервіси Firestore, Authentication, Storage і Cloud Messaging.

Для отримання push-сповіщень створюється сервіс, який працює незалежно від стану додатка. Для надсилання повідомлень використовується Firebase Cloud Messaging.

Інтерфейс для роботи з API EventApi використовує Retrofit для HTTP-запитів. Модуль NetworkModule надає інстанси для роботи з мережею, зокрема, налаштовує Retrofit для роботи з JSON. Модуль DatabaseModule використовується для надання інстансів бази даних, реалізованої за допомогою Room. Клас FireMessageGlideModule застосовується для налаштування Glide, що працює з Firebase Storage. Клас GetEventsUseCase використовується для отримання подій з API та перетворення їх у формат, зручний для використання в додатку. Утилітарний клас StorageUtil призначений для роботи з Firebase Storage, зокрема, завантаження файлів. Утилітарний клас FirestoreUtil призначений для роботи з Firestore, зокрема, збереження та отримання даних про користувачів і чати. Клас MyFirebaseMessagingService забезпечує отримання і обробку push-сповіщень за допомогою Firebase Cloud Messaging.

Така структура коду забезпечує модульність і розширюваність додатка, полегшуючи впровадження нових функцій та підтримку існуючих.

3.5 Розрахунок витрат забезпечення функціонування мобільного додатку

Для забезпечення функціонування мобільного додатку необхідно врахувати різні витрати, такі як: витрати на електроенергію, витрати на розміщення серверної частини в інтернеті (хостинг), заробітну плату програмістів, а також витрати на канцелярські товари, витратні матеріали для комп'ютерів, оренду приміщення, амортизацію комп'ютерів та оргтехніки, і інші операційні витрати. Вартість електроенергії розраховується за тарифом для підприємств, який становить 1 кВт/г = 4,32 грн. Таким чином, вартість споживання електроенергії на місяць складе 1365 грн.

Планується розміщення серверної частини мобільного додатку на ресурсах провайдера з міста Львів, що забезпечить ефективне обслуговування. Розрахунок щомісячних витрат на підтримку серверу наведено в таблиці 3.1.

Таблиця 3.1 – Розрахунок щомісячних витрат на підтримку серверної частини мобільного додатку

Найменування	Сума, грн	ЄСВ, грн
Зарплата програміста	15000	460,80
Зарплата адміністратора	2500	460,80
Транспортні витрати адміністратора	400	×
Електроенергія	1365	×
Хостинг	250	×
Інтернет	120	×
Інші витрати	350	×
Разом:	19885	921,60
Всього витрат:	22091,6	

Таблиця 3.2 – Розрахунок щомісячних матеріальних витрат

Найменування	Сума, грн/міс.
Електроенергія	1365
Хостинг	250
Інтернет	120
Інші витрати	350
Разом:	2085

Витрати на період розробки програмного продукту складають: Зпр = 923,5 грн.

Розрахуємо собівартість програмного продукту за формулою:

$$С_{ст} = З_{пр} + ЗП_{пр} + ЄСВ + А. \quad (3.1)$$

Тоді $С_{ст}$ – собівартість розробки мобільного додатку складе:

Тоді собівартість розробки мобільного додатку складе:

Сст=923,5+1900+460,80+120=3404,3 грн.

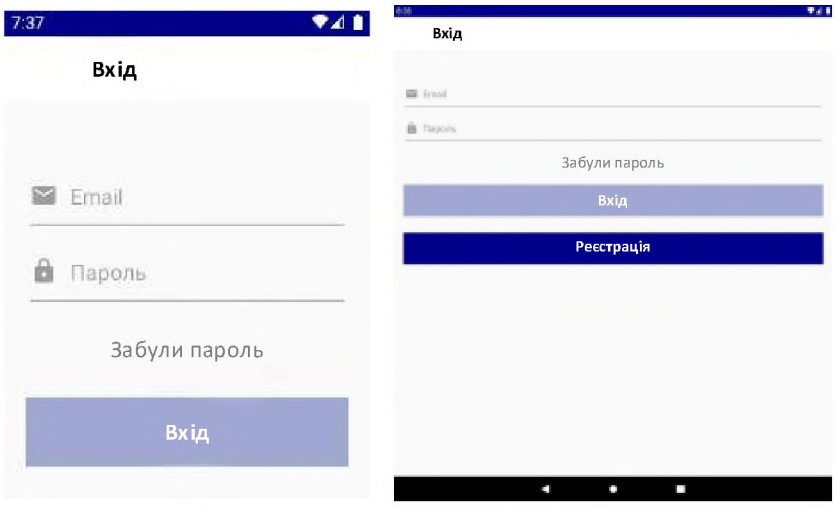
Після проведення розрахунків видно, що основна частина витрат припадає на заробітну плату програмістів та адміністратора, що свідчить про високу вартість людських ресурсів у розробці програмного забезпечення. Також варто звернути увагу на вартість електроенергії, яка займає значну частину матеріальних витрат, що важливо врахувати при плануванні бюджету на подальше функціонування додатку.

3.6 Розробка планів тестування мобільного додатку для тематичного пошуку інформації

Функціональне тестування – це процес перевірки програмного продукту на відповідність встановленим функціональним вимогам. План тестів для перевірки мобільного додатку на функціональність наведено в таблиці 3.3.

Таблиця 3.3 – Функціональні тести

№	Назва тесту	Кроки	Очікуваний результат
1	Перевірка підключення телефону до інтернету	Запуск мобільної програми	Додаток визначає наявність підключення до інтернету
2	Перевірка роботи бази даних	Запуск мобільного додатка без інтернету під авторизованим користувачем	Додаток відобразить останні отримані дані
3	Отримання та парсинг даних при підключенні до інтернету	Запуск мобільної програми з доступом до мобільного інтернету та/або Wi-Fi	Додаток запитує дані з API, отримує коректні дані, парсує їх, записує в базу даних і відображає
4	Перевірка реєстрації існуючого користувача	Спроба реєстрації під вже існуючим користувачем	Додаток видає помилку та повідомляє користувача
5	Перевірка валідності вхідних даних у реєстраційних полях	Заповнення реєстраційних полів некоректними даними	Додаток показує повідомлення про невалідність даних при спробі реєстрації
6	Збереження стану даних у додатку	Зміна орієнтації пристрою	Орієнтація змінюється, відображаючи ті ж дані



а) б)

Рисунок 3.3 – Інтерфейс програми: а) на QVGA; б) Foldable

Якість адаптивності інтерфейсу програми потрібно перевірити шляхом коректного відображення на різних пристроях. Додаток має правильно відображатися на смартфонах з різними діагоналями екранів. Для тестування рекомендується використати віртуальні машини Android Studio. Приклад відображення інтерфейсу на екранах з роздільною здатністю QVGA та на складних екранах показаний на рис. 3.3.

Розробка мобільного додатку для тематичного пошуку інформації включає проєктування дев'яти моделей для зберігання даних та п'яти додаткових моделей для роботи з API та залежностями. Використання бібліотеки Моху забезпечує гнучке управління інтерфейсом користувача та дозволяє зручно управляти екранними елементами. Презентери відповідають за посередництво між даними та інтерфейсом, забезпечуючи чітке розділення логіки та відображення. Реалізація ін'єкції залежностей через Dagger спрощує роботу з компонентами системи, включаючи інтеграцію з Firebase для аутентифікації, зберігання даних та надсилання повідомлень. Проєкт використовує архітектурний патерн MVP, що полегшує підтримку та розширення додатка.

ВИСНОВКИ

В результаті виконання роботи була досягнута її початкова мета та вирішені поставлені завдання. Було проаналізовано основну наукову, методичну та нормативну літературу з теми програмування мобільних додатків з клієнт-серверною архітектурою.

Розглянуто існуючі мобільні додатки для пошуку музичних однодумців та подій, такі як BandFriend, Vampr, TuneUp, та JamConnect. Вони мають різний функціонал, зокрема, налаштування профілю, чат, пошук за категоріями, і є актуальними для розробки подібних рішень на українському ринку. Аналіз інструментів для розробки показав переваги використання Android Studio, Kotlin та різних бібліотек, таких як Hilt, Moxy, та Firebase, для створення надійного та зручного додатку. Визначено функціональні та нефункціональні вимоги, серед яких важливими є збереження і кешування даних, автоматичне оновлення інтерфейсу, валідація введених даних та авторизація користувачів.

Розглянуто варіанти використання мобільного додатка за допомогою UML діаграм, що допомогло систематизувати сценарії взаємодії користувачів з системою. Виконано детальний аналіз архітектури додатка, зокрема обґрунтовано вибір патерну MVP, що сприяє розподілу обов'язків між компонентами і забезпечує простоту коду. Спроєктовано компоненти системи, зокрема описано сутності, які реалізують функціональність додатка, що покращує модульність та зручність в розробці. Запропоновано базу даних, яка забезпечує надійне зберігання та управління інформацією про користувачів, чати, повідомлення та події. Діаграма діяльності для процесу входу і реєстрації користувача описує кроки аутентифікації за допомогою Firebase, що забезпечує безпеку та зручність користувачів. Висновки вказують на важливість комплексного підходу до проєктування мобільного додатка, з акцентом на користувацький досвід та гнучкість системи.

Описано розробку та планування тестування мобільного додатку для тематичного пошуку інформації.

Спроєктовано дев'ять моделей для зберігання даних та п'яти додаткових моделей для роботи з API та залежностями. Використання бібліотеки Moxy забезпечує гнучке управління інтерфейсом користувача та дозволяє зручно управляти екранними елементами. Презентери відповідають за посередництво між даними та інтерфейсом, забезпечуючи чітке розділення логіки та відображення. Реалізація ін'єкції залежностей через Dagger спрощує роботу з компонентами системи, включаючи інтеграцію з Firebase для аутентифікації, зберігання даних та надсилання повідомлень. Проєкт використовує архітектурний патерн MVP, що полегшує підтримку та розширення додатка.

Таким чином, поставлені задачі розв‘язано у повному обсязі. Напрямком подальших досліджень є реалізація запропонованого проєкту в код, розміщення додатку в магазині Apps та впровадження нових методів аутентифікації та зберігання даних з використанням хмарних сервісів для підвищення безпеки користувачів.