МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
« »травня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система організації процесу тренувань на мобільній платформі Android. Серверна частина»
здобувача групи ІН – 02 Попельнуха Назара Павловича

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

_____ Назар ПОПЕЛЬНУХ
(підпис)

Керівник,
к.т.н., доцент кафедри ком'ютерних наук. В'ячеслав МОСКАЛЕНКО _____
(підпис)

Суми – 2024

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-02 Попельнуха Назара Павловича

1. Тема роботи: «Інформаційа система організації процесу тренувань на мобільній платформі Android. Серверна частина»
затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до «29» травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій щодо серверних частин для Android застосунків. 3) Розробка серверної частини інформаційної системи тренувань. 4) Аналіз результатів.
5. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій щодо серверних частин для Android застосунків*		
3	*Розробка серверної частини інформаційної системи тренувань.*		
4	*Аналіз результатів.*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 65 стр., 35 рис., 1 додаток, 16 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є необхідною та актуальною, оскільки присвячується інформаційній системі організації тренувань, які є необхідністю в поточних реаліях дистанційної роботи. В рамках цієї роботи особливої уваги було надано саме серверної частини.

Об’єкт дослідження — серверна частина Android-застосунків.

Мета роботи — створення серверної частини для інформаційної системи організації процесу тренувань.

Методи дослідження — сучасні архітектури та тестування Android-застосункуів.

Результати — розроблено серверну частину системи організації тренувань на платформі Android, яка дозволяє користувачам планувати свої заняття, переглядати та сортувати відео. Проведено тестування розробленої системи на витоки пам’яті та загального навантаження на смартфон.

ІНФОРМАЦІЙНА СИСТЕМА, СЕРВЕРНА ЧАСТИНА, ANDROID, FIREBASE, JAVA.

ЗМІСТ

ВСТУП

Розвиток інформаційних технологій відіграє ключову роль у сучасному світі, адаптуючи численні аспекти повсякденного життя до нових реалій, включаючи здоров'я та фізичну підготовку. Значення та необхідність інформаційних систем для тренувань виникли як відповідь на зміну життєвих обставин, зокрема, підвищення частки сидячої роботи та збільшення рівня віддаленої роботи, що стало особливо актуальним під час та після світової пандемії COVID-19. Ця робота присвячена розробці інформаційної системи організації процесу тренувань на мобільній платформі Android, з акцентом на серверну частину, яка відіграє важливу роль у забезпеченні ефективності та доступності тренувальних програм.

Передусім, ця система відкриває нові можливості для людей, які здійснюють свою професійну діяльність у сидячому положенні та не мають змоги відвідувати спортивні зали. Вона пропонує зручний інтерфейс для взаємодії між тренером та клієнтом, де тренер може завантажувати індивідуальні відео-інструкції та моніторити прогрес своїх клієнтів, тим самим забезпечуючи персоналізований підхід до тренувань.

Актуальність створення такої системи беззаперечно визначається сучасними тенденціями ринку праці та змінами в життєвому стилі багатьох людей. З поширенням віддаленої роботи та збільшенням частки робочого часу, проведеного вдома, важливість домашніх тренувань зросла, а потреба в інтеграції здорових звичок у повсякденний розпорядок стала більш вираженою. Система, що розробляється, не просто сприяє підтримці фізичної активності, але й стає інструментом для підвищення якості життя, дозволяючи користувачам залишатися активними та здоровими, навіть не виходячи з дому.

Вибір платформи Android для реалізації даної системи визначається її широкою доступністю та популярністю серед споживачів. Згідно з останніми

дослідженнями, Android займає значну частку ринку мобільних операційних систем, що робить його ідеальним вибором для масштабного впровадження нової продукту. З іншого боку, роль серверної частини полягає у забезпеченні стабільності, безпеки та масштабованості системи. Серверна частина обробляє дані користувачів, забезпечує високу швидкість відгуку і здатна витримувати велике навантаження, що є критично важливим для забезпечення надійної та ефективної роботи системи.

У подальшому, дана робота розглядатиме детальні аспекти розроблення мобільного додатку, аналізуватиме потенційні виклики та розроблятиме рекомендації щодо оптимізації та підвищення ефективності використання інформаційної системи для тренувань. Цілі та завдання, визначені у цьому дослідженні, спрямовані на досягнення максимальної користі та задоволення потреб кінцевих користувачів, що стане значущим внеском у сферу мобільних застосунків для здоров'я та фітнесу.

1. АНАЛІЗ ПРОБЛЕМИ

1.1 Сучасний стан та тенденції розвитку електронних систем організації тренувань

1.1.1 Історичний розвиток

Зародження електронних систем для організації тренувань можна відстежити до початку 2000-х років, коли з'явилися перші споживчі фітнес-пристрої, такі як крокоміри та монітори серцевого ритму. Справжній прорив відбувся з появою смартфонів та мобільних додатків, які інтегрували GPS-трекінг, персоналізовані тренувальні плани та зворотний зв'язок від тренерів. Інновації, які започаткували бренди Fitbit та Garmin, сприяли активній інтеграції технологій у повсякденне тренування, відкриваючи нові можливості для моніторингу та поліпшення фізичної активності.[4]

1.1.2 Аналіз сучасного стану

Сучасні електронні системи організації тренувань представляють широкий спектр продуктів, від мобільних додатків до носимих пристроїв, які використовують передові технології для підвищення ефективності тренувань:

Додатки як MyFitnessPal та Strava використовують дані користувачів для створення індивідуальних тренувальних планів, що враховують фізичний стан, цілі та переваги.[5, 6]. Ця персоналізація допомагає користувачам досягти кращих результатів та підтримувати мотивацію.

Системи здатні синхронізуватися з широким спектром пристроїв та платформ, включаючи здоров'я Apple та Google Fit, забезпечуючи користувачам універсальний доступ до їхніх даних про здоров'я та активність.

Використання штучного інтелекту та машинного навчання: AI та ML допомагають аналізувати великі масиви даних для надання рекомендацій,

прогнозування результатів тренувань та навіть раннього виявлення можливих травм.

1.2 Аналіз підходів до проектування backend-частини систем організації тренувань

Аналізуючи підходи до проектування backend-частини мобільних додатків для тренувань, важливо зосередитися на архітектурних рішеннях, які відповідають сучасним вимогам та забезпечують гнучкість, масштабованість та високу продуктивність. Розглядаючи додатки для Android, особливу увагу потрібно приділити таким патернам як MVP (Model-View-Presenter), MVVM (Model-View-ViewModel), MVI (Model-View-Intent) та Clean Architecture. Ці архітектурні рішення допомагають структурувати додаток таким чином, щоб він був зручний для тестування, розвитку та подальшої підтримки.

Значний зсув при розробці додатків для Android спостерігається у бік окремих архітектурних патернів[7], призначених для покращення масштабованості, супроводу та ефективного вирішення окремих проблем. Ось чотири основні архітектурні підходи: MVP, MVVM, MVI та Clean Architecture.

1.2.1 MVP

Архітектура Model-View-Presenter (MVP) адаптована з класичної моделі MVC, щоб краще відповідати розробці Android, розділяючи логіку програми, користувацький інтерфейс та взаємодію з користувачем. У MVP модель обробляє частину даних, представлення керує інтерфейсом, а презентатор виступає в ролі посередника, отримуючи дані з моделі та форматуючи їх для представлення. Такий поділ покращує тестування та управління, хоча і збільшує кількість шаблонного коду і може бути складним для новачків.

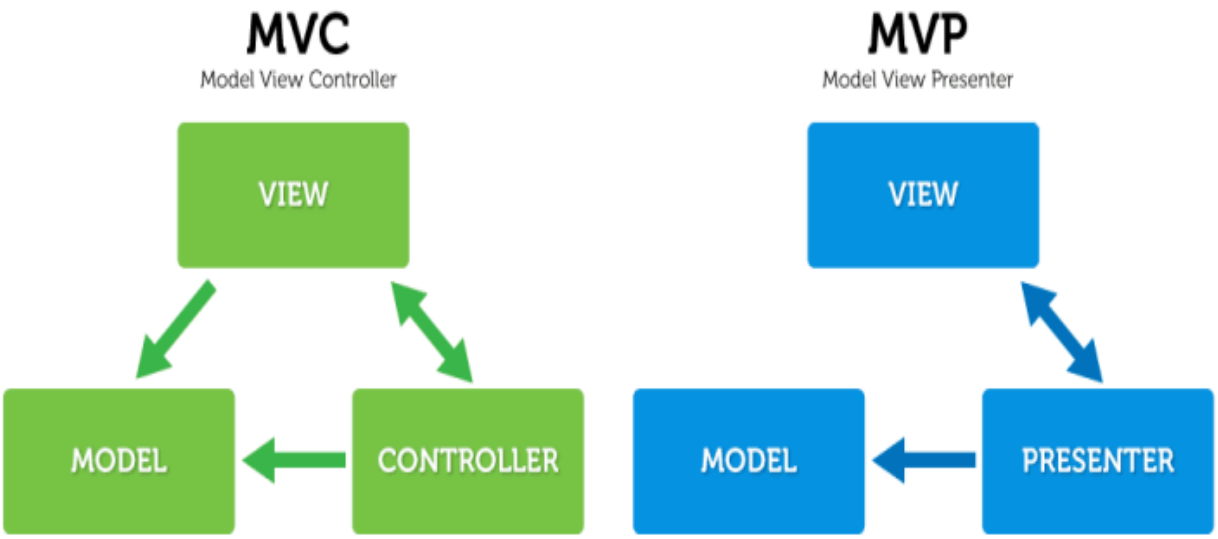


Рисунок 1.1 – Архітектурна модель MVP

1.2.2 MVVM

Model-View-ViewModel (MVVM) - ще одна надійна архітектура, яка підтримує інтерфейси, керовані даними, що робить її ідеальною для розроблення адаптивних і масштабованих додатків. ViewModel в MVVM з'єднує модель і представлення, керуючи даними, які можна спостерігати і які запускають оновлення інтерфейсу користувача. Ця архітектура зменшує потребу в шаблонах завдяки можливостям прив'язки даних і обробляє життєвий цикл даних, пов'язаних з інтерфейсом, що особливо корисно під час зміни конфігурації, наприклад, при повороті екрану. Однак вона може спричинити накладні витрати, які можуть бути непотрібними для простіших завдань інтерфейсу, і має більш круту криву навчання завдяки своїм передовим концепціям, таким як LiveData і прив'язка даних.

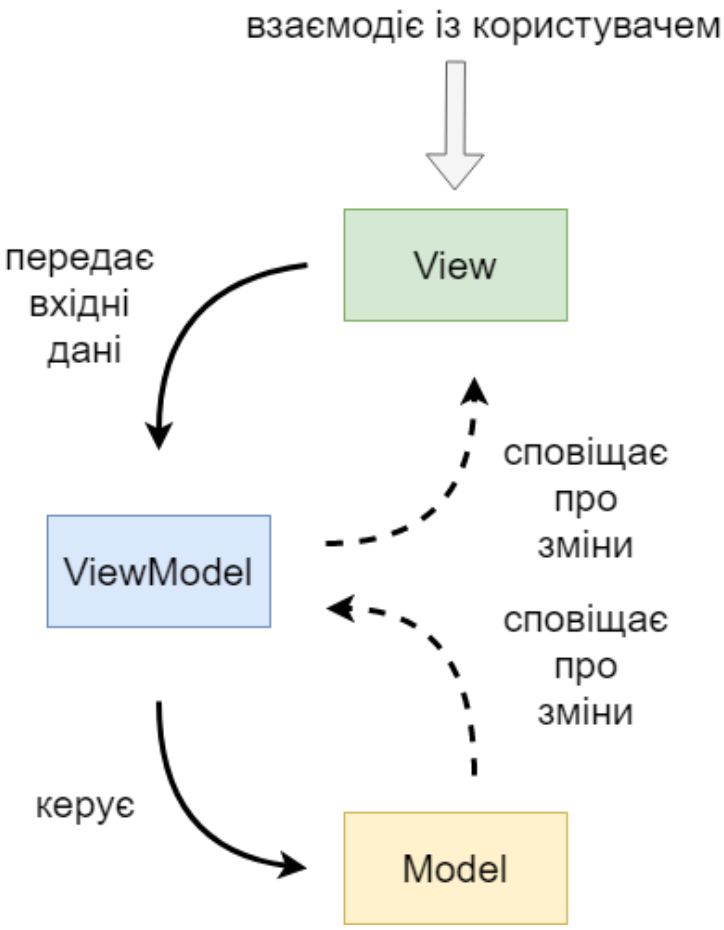


Рисунок 1.2 – Архітектурна модель MVVM

1.2.3 MVI

Model-View-Intent (MVI) - це новітній патерн, що підкреслює односпрямований потік даних і незмінні стани, де View надсилає наміри змінити стан, Model оновлює стан, а View рендерить інтерфейс на основі цього стану. Цей патерн забезпечує передбачуваність і зменшує кількість помилок завдяки управлінню незмінними станами, але є більш складним і краще підходить для більш складних додатків.

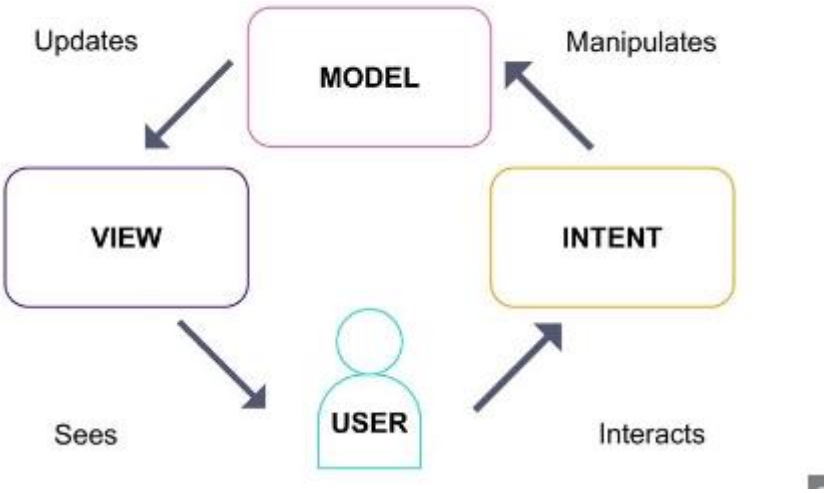


Рисунок 1.3 – Архітектурна модель MVI

1.2.4 Clean Architecture

Clean Architecture-підкреслює суворе розділення, розділяючи програмне забезпечення на шари, кожен з яких має свою власну відповідальність, сприяючи незалежності від фреймворку і покращуючи тестування. Незважаючи на свої переваги, чиста архітектура може призвести до складності в налаштуванні та управлінні і може призвести до надмірної інженерії для більш простих додатків.

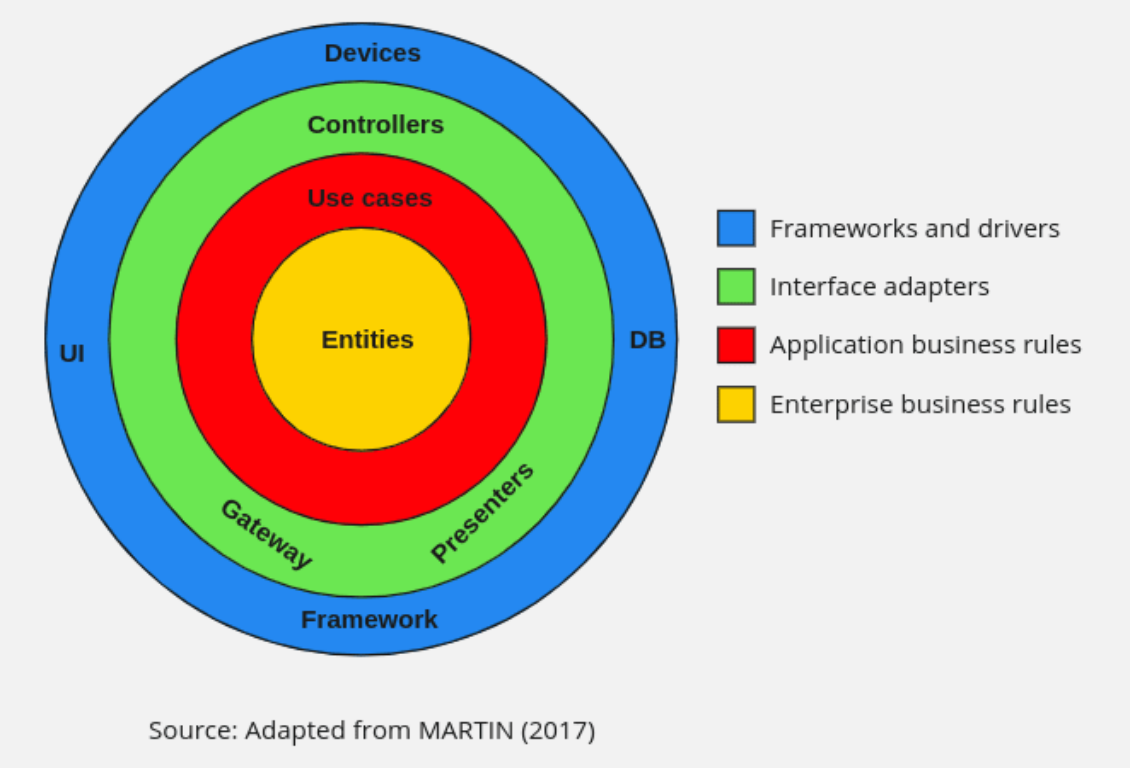


Рисунок 1.4 – Архітектурна модель Clean Architecture

1.3 Формалізована постановка задачі

Розробка інформаційної системи для організації тренувань на мобільній платформі Android має на меті створення зручного, ефективного та інтерактивного інструменту для користувачів, які бажають вести активний спосіб життя без необхідності відвідування спортивних залів. Система забезпечуватиме користувачам можливість вибору, налаштування та проведення індивідуальних тренувань, керування прогресом та отримання зворотного зв'язку від тренерів.

Головна мета проекту — надати користувачам додаток, який дозволить їм планувати, виконувати та аналізувати тренування з персоналізацією і інтерактивностю, використовуючи дані для підвищення ефективності фізичних вправ.

Система передбачає декілька ключових завдань і функцій. Насамперед, вона має формувати базу даних, яка включатиме профільні дані користувачів, а також їхню історію та плани тренувань. Також система дозволяє користувачам персоналізувати свої тренування, адаптуючи їх до особистих цілей та фізичних можливостей. Це означає, що кожен зможе скласти індивідуальний тренувальний план.

Функціональні вимоги:

- Механізми реєстрації та авторизації для користувачів.
- Завантаження та управління відеофайлами у профілі користувача.
- Відображення відео та інтерактивних елементів у профілі користувача.
- Планування тренувань за допомогою інтегрованого календаря.
- Інтеграція з обраними компонентами Firebase для зберігання та обробки даних.

Нефункціональні вимоги:

- Висока надійність та доступність системи.
- Швидка відповідь сервера на запити користувачів.
- Стійкість системи до помилок та збоїв.

Завдяки реалізації цих задач та вимог, розроблена система дозволить користувачам ефективно управляти своїми тренуваннями, сприяючи підвищенню загальної фізичної активності та здоров'я.

2. Моделі backend-частини інформаційної системи організації тренувань

2.1 Структурні та функціональні схеми інформаційної системи

Система організації тренувань використовує комплексне рішення Firebase від Google для зберігання даних про користувачів в базі даних, для зберігання фото та відео файлів. Firebase пропонує набір хмарних сервісів, призначених для розроблення додатків для Android.[8] Використання можливостей Firebase дає змогу спростити внутрішню розробку, підвищити рівень залучення користувачів та покращити загальну продуктивність додатків. Firebase є фундаментом для серверної частини інформаційної системи тренувань.

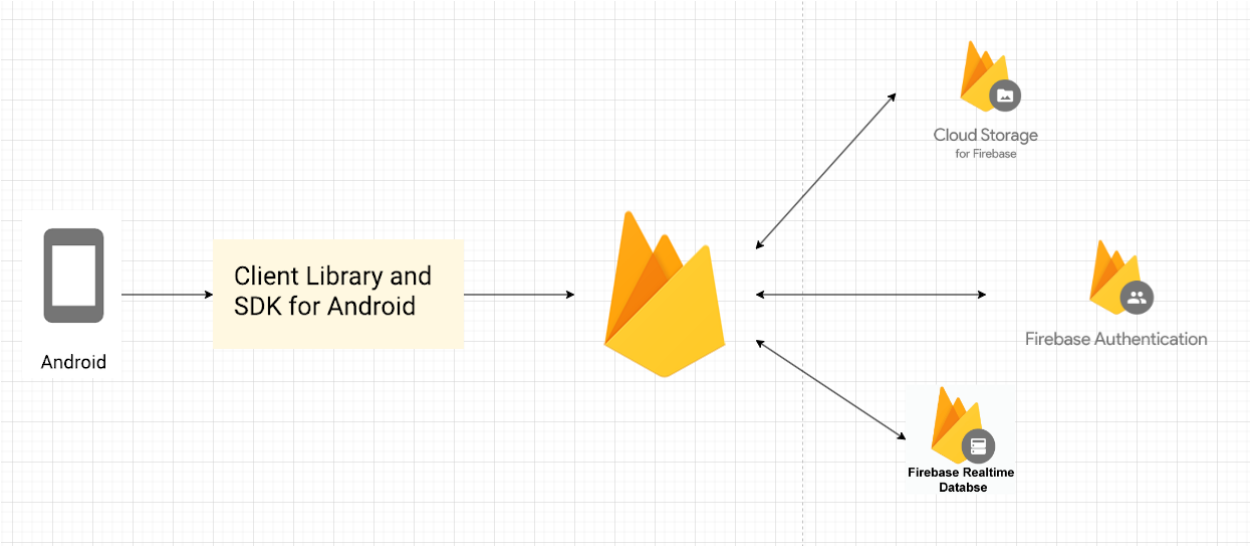


Рисунок 2.1 – Схема взаємодії мобільного додатку та хмарного рішення Firebase

Firebase є прикладом BaaS (Backend as a Service), що є хмарним рішенням, яке надає розробникам автоматизовані рішення для багатьох завдань, які зазвичай виконує бекенд, таких як бази даних, аутентифікація, аналітика, зберігання файлів та інші.

Переваги використання Firebase як BaaS:

- Швидкість розробки - значно скорочує час розробки за рахунок автоматизації багатьох бекенд-завдань.
- Масштабування - автоматичне масштабування інфраструктури для задоволення потреб додатка.
- Спрощення управління - зменшення необхідності управління серверами та інфраструктурою.

Firebase надає потужний набір інструментів, які допомагають розробникам мінімізувати складність та витрати на управління серверною частиною своїх додатків, зосереджуючи увагу на створенні кращого користувацького досвіду.

2.1.1 Firebase Storage

Firebase Storage, невід'ємний компонент платформи Firebase, надає масштабовані рішення для зберігання даних для додатків Android. За допомогою Firebase Storage розробники можуть ефективно зберігати створений користувачем контент, такий як зображення, відео та документи, у безпечний та надійний спосіб. Крім того, Firebase Storage легко інтегрується з іншими службами Firebase, забезпечуючи безперебійне управління даними та їх пошук в Android-додатках.

Firebase Storage організовано у вигляді "бакетів" (buckets), які містять файли. Кожен файл у Firebase Storage має шлях та ім'я, і може бути структурованим у каталогах, хоча фізично Firebase не використовує справжнісінькі каталоги. Замість того, шляхи до файлів містять слеші, які інтерпретуються як структура каталогів. Наприклад такий вигляд має структура Firebase Storage для інформаційної системи:

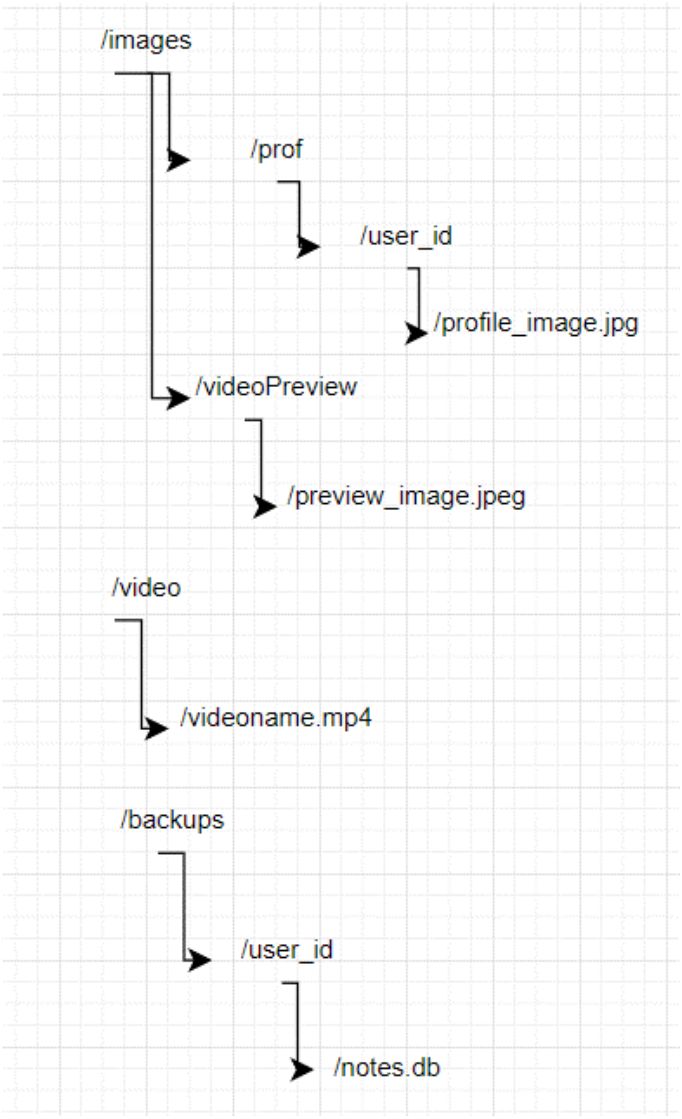


Рисунок 2.2 – Структура файлової системи в Firebase Storage

/images - Ця директорія містить всі зображення, які поділені на два каталоги:

1. /prof - Каталог для зображень профілів користувачів. В якій кожен користувач має окремий підкаталог з назвою, що відповідає його user_id. В кожному такому підкаталозі зберігається файл profile_image.jpg, що є зображенням профілю користувача.
2. /videoPreview - Каталог для зображень, які використовуються для попереднього перегляду відео. Зображення зазвичай мають власну назву в залежності від відео.

/video – Це основна директорія для зберігання відеофайлів. Відеофайли зберігаються безпосередньо в цій директорії та мають унікальні назви. Для конфігурації власного імені використовуєтсья функція System.currentTimeMillis() за допомогою якої ми отримуємо час на момент завантаження в мілісекундах.

```
//завантажуємо саме відео
final StorageReference referenceVideo = storageVideoReference.child( pathString: System.currentTimeMillis() +
        "." + getExt(videoUri));
uploadTaskVideo = referenceVideo.putFile(videoUri);
```

Рисунок 2.3.1 – Отримання назви відео. Реалізація в коді

За такою ж схемою задаємо назву зображенню preview_image.jpeg:

```
final StorageReference referenceImage = storageImageReference.child( pathString: System.currentTimeMillis() +
        ".JPEG");
uploadTaskImage = referenceImage.putBytes(imageData);
```

Рисунок 2.3.2 – Отримання назви прев’ю для відео. Реалізація в коді

/backups - Ця директорія зберігає записи користувачів в календарі та їх нотатки. Ці записи зберігаються в каталог /user_id, назва якого встановлюється відповідно до ідентифікатора користувача в Firebase Authentication. Записи зберігаються у файлі notes.db, який фактично є базою даних SQLite, через яку ми робимо записи нотатків та записуємо дані в календар.

Структура дозволяє легко визначити, де знаходяться файли, пов'язані з користувачами та їхніми відео, та ефективно управляти доступом та взаємодією з цими файлами через додаток за допомогою Firebase SDK.

2.1.2 Firebase Database

Firebase Database пропонує хмарне рішення для баз даних NoSQL, що полегшує зберігання та синхронізацію даних у режимі реального часу. Використання Firebase Database дозволяє розробникам створювати адаптивні та спільні додатки для Android, які динамічно оновлюють вміст на різних

пристроях. Модель даних Firebase Database на основі JSON спрощує структурування та пошук даних, підвищуючи ефективність процесів управління даними в Android-додатках.

Рисунок 2.4 – Структура Firebase Realtime Database

Об'єкт "Users":

Цей об'єкт містить дані користувачів, кожен ідентифікований унікальним ключем (ID користувача). Для кожного користувача зазначені наступні атрибути:

- "email": Електронна пошта користувача.
- "name": Ім'я користувача.
- "pass": Пароль користувача.
- "phone": Телефонний номер користувача.
- "profileUri": URL до зображення профілю користувача, яке зберігається на Firebase Storage.

Приклад користувача:

- Користувач з ID "J5kN6RccRMhlC1uiIRXptCISmy93" має ім'я "test" і електронну пошту "testcase@gmail.com".

Об'єкт "Video":

Цей об'єкт зберігає інформацію про відео, яка включає:

- "videoDuration": Тривалість відео.
- "videoName": Назва відео.
- "videoPreviewImage": URL до зображення попереднього перегляду відео, яке зберігається на Firebase Storage.
- "videoUri": URL до самого відеофайлу, яке також зберігається на Firebase Storage.
- "videoTags": Теги, що описують відео, використовуються для фільтрації запитів. Вони допомагають швидко знаходити відео за певними критеріями, такими як групи м'язів або тип вправ.

Приклади відео:

- Відео з ID "-Nx4-RxoO9M0nvJRF5F1" має назву "John" і тривалість "00:45". Теги: "pectoralis_major, latissimus_dorsi, anterior_deltoid, biceps_brachii, quadriceps, rectus_abdominis".
- Відео з ID "-NxESi8pfnKHOPEL_FJp" має назву "Abdominal" і тривалість "00:45". Теги: "rectus_abdominis, obliques, transverse_abdominis".

2.1.3 Firebase Authentification

Забезпечення безпечної автентифікації користувачів є першочерговим завданням при розробці додатків для Android. Firebase Authentication надає надійне рішення для реалізації механізмів автентифікації в додатках Android, підтримуючи різні методи автентифікації, такі як електронна пошта/пароль, номер телефону та сторонні постачальники автентифікації. Інтегруючи Firebase Authentication, розробники можуть спростити процес реєстрації користувачів і підвищити безпеку додатків, тим самим сприяючи безперебійній та безпечній роботі користувачів.

Приклад використання в коді:

```
// Вхід за поштою та паролем
mAuth.signInWithEmailAndPassword(email.getText().toString(), pass.getText().toString())
        .addOnSuccessListener(authResult -> {
            startActivity(new Intent( packageContext: MainActivity.this, MenuActivity.class));
            finish();
        })
        .addOnFailureListener(e -> Snackbar.make(root,  text: "Помилка авторізації!" + e.getMessage(), Snackbar.LENGTH_LONG).show());
```

Рисунок 2.5 – Авторизація за допомогою пошти та пароля в Firebase Authentification

```
// Сама реєстрація користувача в БД
mAuth.createUserWithEmailAndPassword(email.getText().toString(), pass.getText().toString())
        .addOnSuccessListener(authResult -> {
            Map<String, Object> userMap = new HashMap<>();
            userMap.put( k: "email", email.getText().toString());
            userMap.put( k: "pass", pass.getText().toString());
            userMap.put( k: "name", name.getText().toString());
            userMap.put( k: "phone", phone.getText().toString());

            users.child(FirebaseAuth.getInstance().getCurrentUser().getUid()) DatabaseReference
                    .setValue(userMap) Task<Void>
                    .addOnSuccessListener(unused -> Snackbar.make(root,  text: "Новий користувач успішно створений!", Snackbar.LENGTH_LONG).show())
                    .addOnFailureListener(e -> Snackbar.make(root,  text: "Виникли складнощі при створенні користувача!", Snackbar.LENGTH_LONG).show());
        });
```

Рисунок 2.6 – Реєстрація користувача в Firebase Authentification

Таким чином, Firebase пропонує комплексне хмарне рішення для додатків Android, що включає в себе послуги зберігання, бази даних та автентифікації. Firebase має значні переваги з точки зору синхронізації в режимі реального часу, простоти інтеграції та безпеки.

2.1.4 Функціональні схеми: Потоки даних з використанням Firebase

Для авторизації в системі, процес потоку даних буде наступним:

- Користувач вводить свої облікові дані (емейл та пароль).
- Дані відправляються до Firebase Authentication.
- Firebase перевіряє облікові дані.
- У разі успіху, користувач отримує токен, який підтверджує його аутентифікацію.
- Користувач має доступ до додатку та свого облікового запису

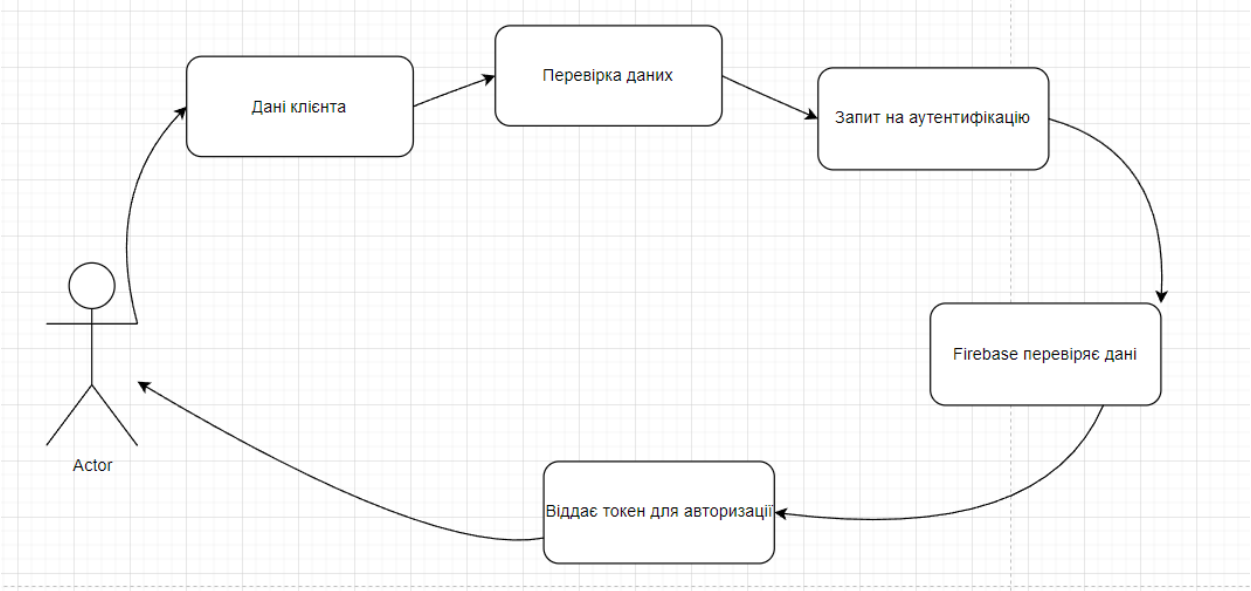


Рисунок 2.7 – Вигляд діаграми послідовностей під час авторизації

Для реєстрації в системі, процес потоку даних буде наступним:

– Користувач вводить необхідну інформацію для реєстрації (емейл, пароль, ім'я тощо).
– Дані відправляються на сервер Firebase.
– Firebase реєструє нового користувача в Firebase Authentification та зберігає дані в Firebase Realtime Database.
– Користувач отримує підтвердження реєстрації.

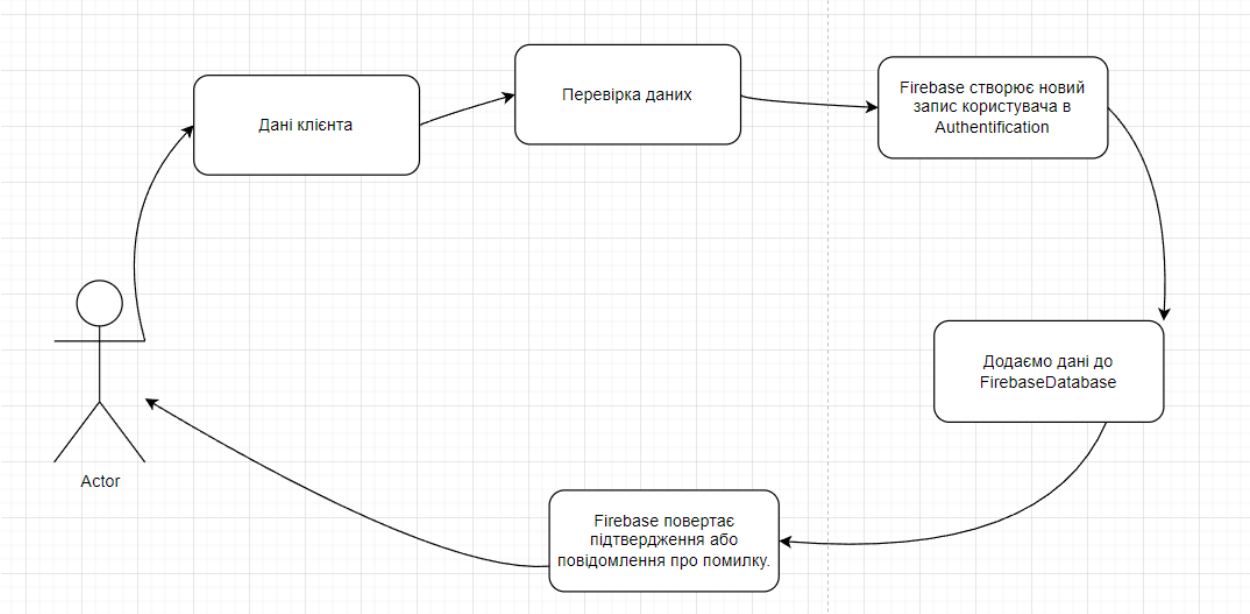


Рисунок 2.8 – Вигляд діаграми послідовностей під час реєстрації

Для завантаження медіафайлів в систему, процес потоку даних буде наступним:

- Користувач вибирає файл для завантаження (наприклад, відео).
- Файл завантажується на Firebase Storage.
- Після завантаження, інформація про файл зберігається у Realtime Database з посиланням на файл.
- Firebase реєструє нового користувача та зберігає дані в Firebase Realtime Database.
- Для відтворення, дані про файл отримуються з Firestore, і відео відтворюється з URL, що зберігається у Storage.

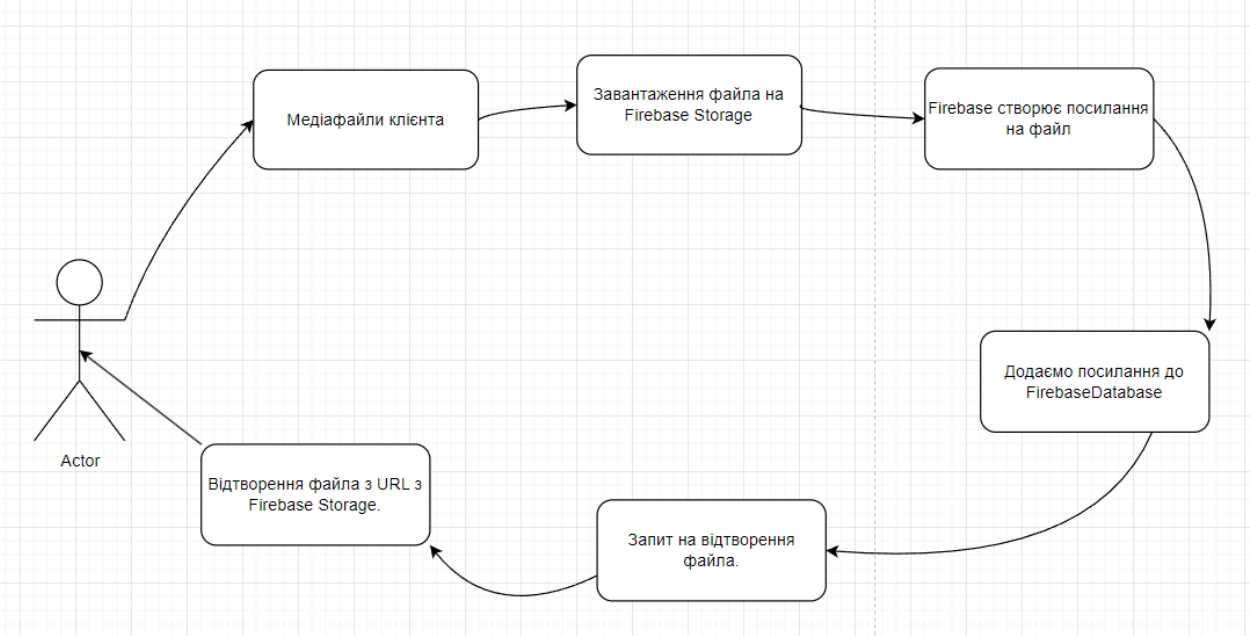


Рисунок 2.9 – Вигляд діаграми послідовностей під час завантаження та відтворення даних

При оновлені даних користувачем, процес потоку даних буде наступним:

- Користувач оновлює дані
- Зміна даних у базі.
- Отримання оновлених даних назад до системи
- Оновлення інтерфейсу користувача з новими даними.

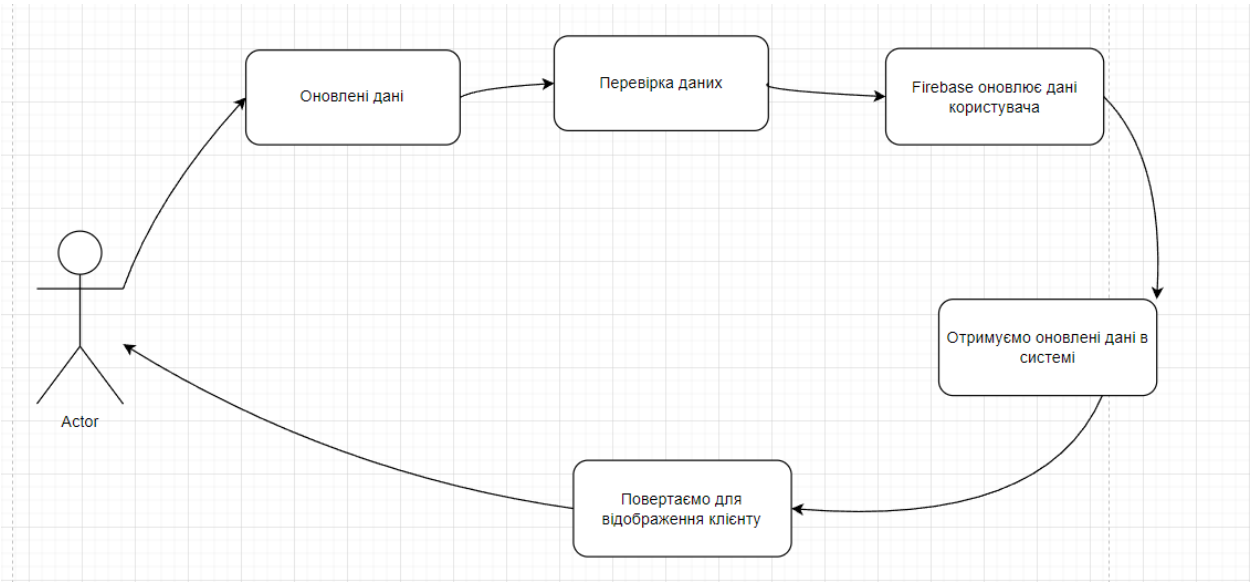


Рисунок 2.10 – Вигляд діаграми послідовностей під час оновлення даних клієнтом

При запису нотатків до календаря, процес потоку даних буде наступним:

– Користувач створює нотатку в календарі.
– Нотатка записується до локальної бази даних SQLite.
– База даних з нотатками завантажується в Firebase Storage.
– Календар оновлюється з новими нотатками для відображення користувачу.

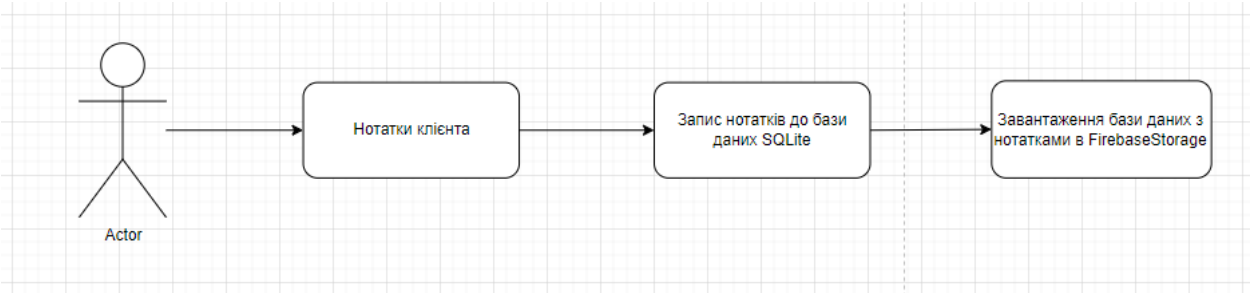


Рисунок 2.11 – Вигляд діаграми послідовностей під час запису нотатків до Firebase Storage

При завантаженні бази даних з Firebase Storage, процес потоку даних буде наступним:

– Користувач ініціює завантаження бази даних.

– Отримання бази даних з Firebase Storage.
– Завантаження нотаток з бази даних.
– Відображення нотаток в календарі для користувача.

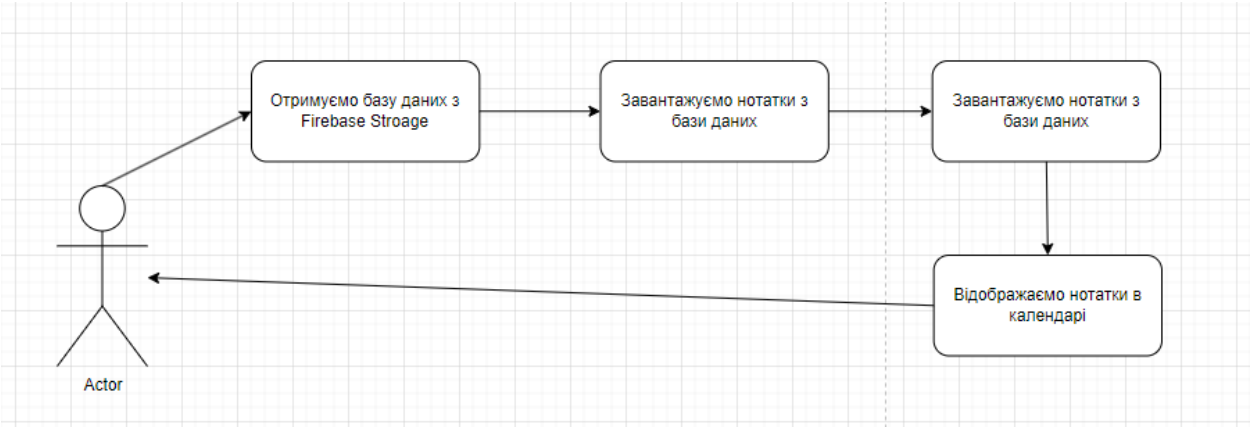


Рисунок 2.12 – Вигляд діаграми послідовностей під час завантаження бази даних нотатків з Firebase Storage

2.2 Архітектура проектування backend-частини інформаційної системи

Архітектура проектування MVVM складається з трьох основних компонентів:

– Model - Забезпечує представлення даних та бізнес-логіки, які керують даними. Модель відповідає за доступ до джерел даних, бізнес-правил та за їх збереження та обробку.
– View - Відповідає за візуальне представлення даних (UI), тобто за те, що бачить користувач. View повністю відокремлена від моделей і не містить бізнес-логіки, лише логіку для відображення даних.
– ViewModel - Є посередником між View і Model, виконуючи функції з управління даними, що потрібні для View. ViewModel обробляє всю логіку інтерфейсу, отримує дані з моделей та форматує їх для відображення.

Приклад з структури додатку:

- **MenuActivity (View), ProfileActivity (View)** - Класи, які представляють UI компоненти та логіку для взаємодії з користувачем. Він відображає дані користувача і слухає події від користувача, передаючи відповідні команди в ProfileViewModel.
- **ProfileViewModel (ViewModel)** - Управляє даними для ProfileActivity, запитує оновлення від UserRepository, і реагує на дії користувача, ініціюючи зміни в моделі або оновлення View через LiveData. LiveData - це клас із бібліотеки Android Architecture Components, який створений для зберігання даних, які потрібно спостерігати.
- **UserRepository (Model)** - Взаємодіє з Firebase для отримання та відправлення даних користувачів. Цей клас виступає як головне джерело даних і не залежить від ViewModel.

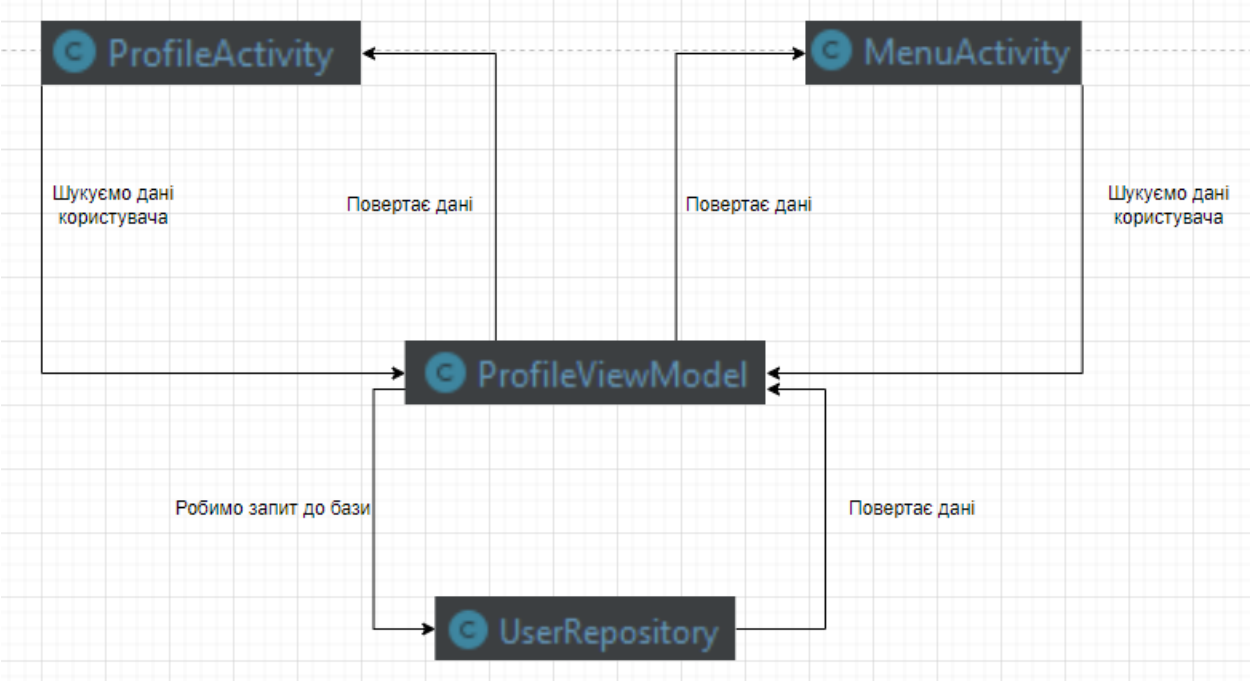


Рисунок 2.13 – Схема роботи елементів профілю системи за архітектурою MVVM

Переваги використання MVVM:

- Розділення відповідальності: Кожен компонент MVVM відповідає лише за свою частину логіки.

- Поліпшення тестування: Компоненти можуть бути легко тестовані окремо.
- Легкість управління та розширення коду: Завдяки чіткому розділенню логіки, у системі легше вносити зміни та додавати нові функції.

Використання MVVM у проекті дозволяє забезпечити краще управління залежностями між різними частинами програми, підвищити можливість повторного використання коду та спростити підтримку та масштабування продукту.

3. Реалізація інформаційної системи організації тренувань

3.1 Короткий опис програмного забезпечення

Розробка інформаційної системи для організації процесу тренувань на мобільній платформі Android вимагає інтеграції передових технологій і платформ для забезпечення функціональності та ефективності. Важливими компонентами в архітектурі системи є використання Android Studio як основного середовища розроблення та Firebase як хмарної платформи, що надає ряд сервісів для підтримки серверної частини.

3.1.1 Середовище розроблення Android Studio

Android Studio є офіційним інтегрованим середовищем розроблення (IDE) для платформи Android, розробленим компанією Google. Це середовище розробленнястворене на базі IntelliJ IDEA від JetBrains і забезпечує розробників всім необхідним набором інструментів для ефективної розроблення мобільних додатків.

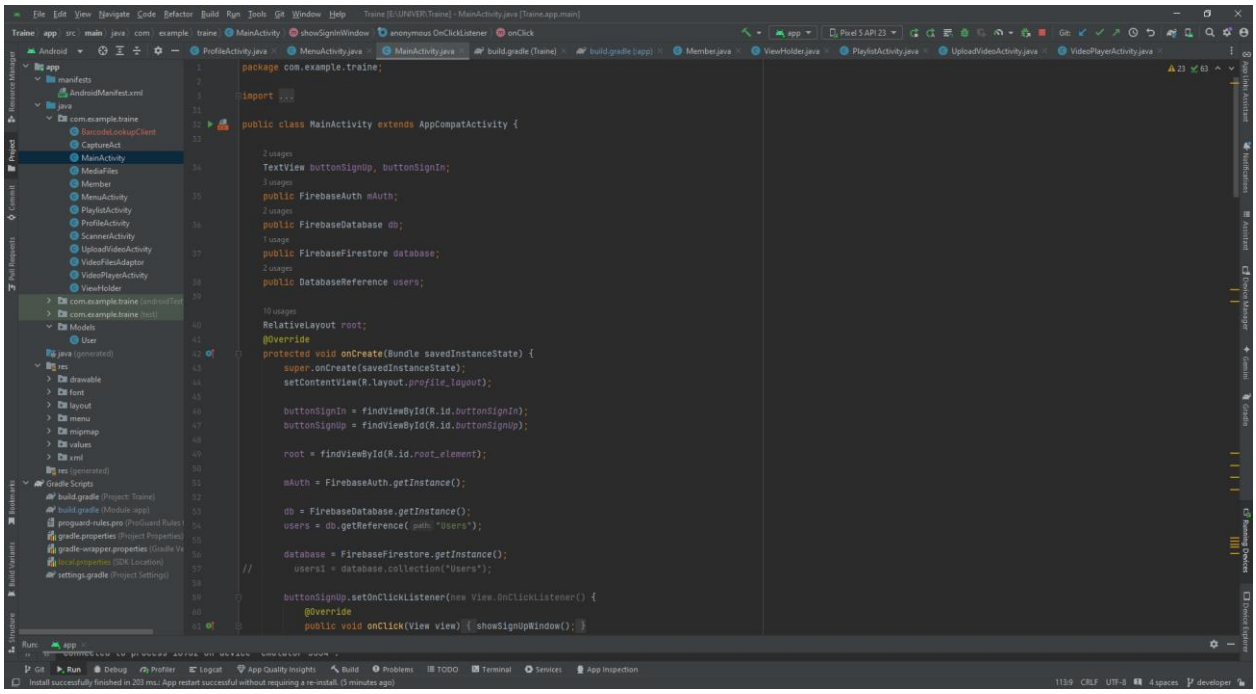


Рисунок 3.1 – Інтерфейс (IDE) Android Studio

Android Studio пропонує багатий набір функціональних можливостей, що сприяють підвищенню продуктивності розроблення та якості кінцевого продукту. Особливо важливими є інструменти, такі як Android Profiler та Logcat, які дозволяють глибоко аналізувати роботу додатків.

Android Profiler в Android Studio – це потужний інструмент для моніторингу реального часу різних метрик додатку, включаючи використання ЦПУ, споживання пам'яті, активність мережі та статистику використання батареї. Профайлер дозволяє розробникам точно визначати та оптимізувати вузькі місця в продуктивності додатку. Розширені графіки та таймлайни забезпечують детальний огляд процесів, що виконуються у додатку, дозволяючи аналізувати конкретні події, такі як виклики до бази даних, запити HTTP, та операції читання/запису в пам'ять.

Рисунок 3.2 – Інтерфейс Android Profiler у Android Studio

Logcat є інструментом для перегляду системних логів у Android, що надає інформацію про різні події, які відбуваються в операційній системі та додатках. Він дозволяє фільтрувати повідомлення за кількома критеріями, включаючи рівень серйозності (наприклад, помилка, попередження, інформація) та додаток-джерело. Розробники можуть використовувати Logcat для діагностики та виправлення помилок в коді, моніторингу системних подій або перевірки виведення дебаг-повідомлень від їхнього додатку.

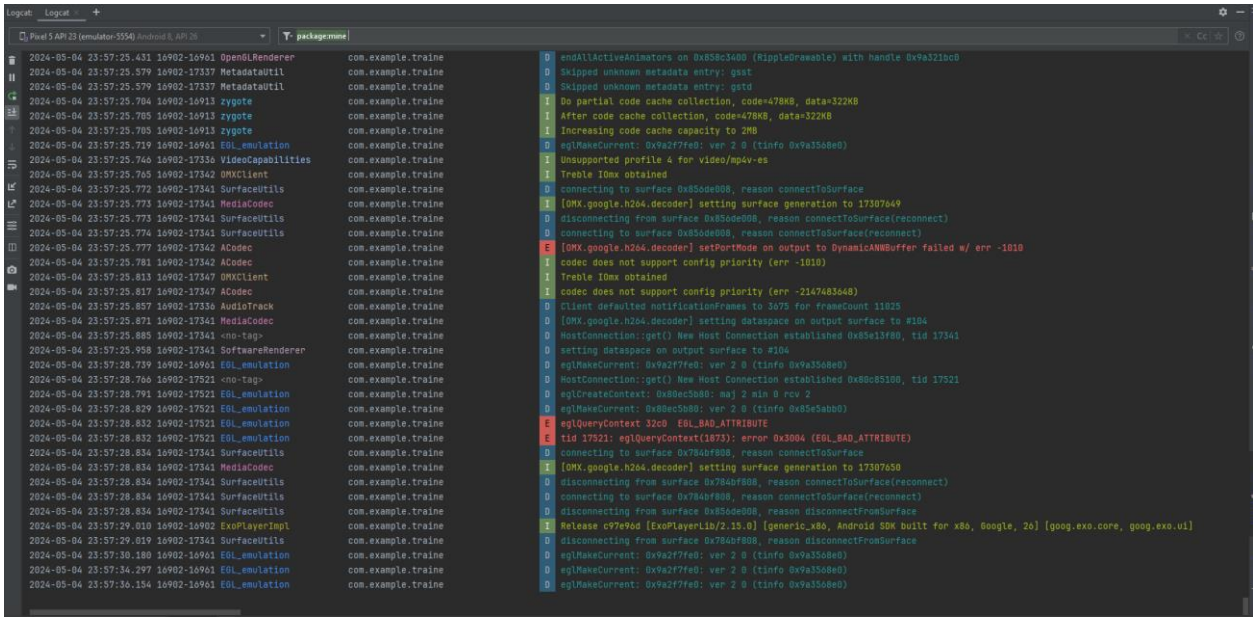


Рисунок 3.3 – Інтерфейс Logcat у Android Studio

Android Studio не тільки спрощує процес створення додатків для Android, надаючи інтегровані інструменти для розроблення, дебагінгу та тестування, але й підвищує якість кінцевих продуктів завдяки розширеним можливостям для профілювання та моніторингу їх роботи. Використання Android Profiler та Logcat у комплексі дозволяє розробникам досягати високих стандартів продуктивності та стабільності, необхідних для сучасних мобільних додатків.

3.2 Опис програмного інтерфейсу backend-частини інформаційної системи організації тренувань

Backend-частина системи була розроблена з метою забезпечення надійної та ефективної обробки даних, а також взаємодії з мобільним додатком. Цей компонент відповідає за аутентифікацію користувачів, зберігання інформації про тренування в базі даних, зберігання зображень та відео, які потребують централізованої логіки обробки.

3.2.1 Мова програмування Java в порівнянні з Kotlin

Java, яка була запроваджена в 1995 році компанією Sun Microsystems, завжди була визнана за її стабільність, переносимість та об'єктно-орієнтованість, що зробило її однією з найпопулярніших мов програмування, особливо для розроблення серверних додатків. Ці властивості Java дозволяють ефективно виконувати обробку великих обсягів даних і взаємодію з різноманітними системними компонентами, що є критично важливим для комерційних додатків, які обслуговують мільйони користувачів. Крім того, її велика кількість доступних бібліотек та фреймворків значно спрощує процес розроблення та підтримку програмного забезпечення.

Завдяки строгій типізації та використанню віртуальної машини Java (JVM), яка ізолює виконання байт-коду від апаратної платформи, Java надає високий рівень безпеки. Це знижує ризики пов'язані з виконанням шкідливих програм і зламом систем. Ще однією значущою перевагою Java є її здатність до масштабування, яка включає як горизонтальне, так і вертикальне масштабування систем, що дозволяє збільшувати продуктивність з ростом навантаження.

Проте, порівняно з Kotlin, який був спеціально розроблений для усунення деяких недоліків Java, таких як складність мови та велика кількість

шаблонного коду, Java може здаватися менш привабливою. Kotlin пропонує більш чистий та виразний синтаксис, що знижує кількість коду, необхідного для реалізації стандартних задач програмування. Також Kotlin включає сучасні можливості програмування, такі як розширені функції, лямбди та інлайнові функції, які можуть значно підвищити продуктивність розробника.

Parameter	Java	Kotlin
Static memebers	Almost the same. Thinking of the solution is the most time-consuming part	Almost the same. Thinking of the solution is the most time-consuming part
Perfomance	Almost the same. Both compile to ByteCode	Almost the same. Both compile to ByteCode
Stability	Has stable versions with long-term maintenance	Almost the same. Both compile to ByteCode
Documentation	Good, easy to find	Good, a little bit harder to find
Popularity	Extremely popular worldwide	Not so popular worldwide
Community	Mostly Indian, very broad	Mostly Russian, comparatively little
Talent pool	Not in the top-list	In the list of the most popular technologies 2020 according to StackOverflow Dev Survey
Easiness to learn	Easy to learn	Can be tricky to learn if you are not a good abstract thinker

Рисунок 3.4 – Порівняння Java та Kotlin

Враховуючи ці аспекти, Java залишається перевіреним вибором для розроблення бекенд-частин мобільних додатків через свою стабільність, безпеку та масштабування, хоча Kotlin набирає популярності серед розробників завдяки своїм сучасним можливостям і зменшеній складності. Незважаючи на нові можливості, які пропонує Kotlin, Java продовжує бути важливим інструментом у арсеналі розробників бекенду, забезпечуючи надійну основу для великих і складних систем.

3.2.2 Gradle

Gradle є потужним інструментом автоматизації, який значно змінив процес розроблення програмного забезпечення, особливо у контексті мобільних додатків та серверних рішень. Вперше введений у 2007 році, Gradle використовує гнучкий підхід на основі графа залежностей, що

дозволяє моделювати власне виконання завдань та залежності між ними динамічно. Основна відмінність Gradle від попередників, таких як Apache Ant та Maven, полягає в його можливості підтримки різноманітних мов програмування та платформ, включно з Java, C/C++, Python та іншими.[12]

Ключовою особливістю Gradle є його використання DSL (Domain-Specific Language), який базується на Groovy чи Kotlin, що надає розробникам масштабованість та гнучкість у налаштуванні скриптів збірки. Це включає автоматизацію компіляції коду, управління залежностями, пакування додатків та їх розгортання, а також запуск тестів і підтримку різноманітних плагінів, що розширюють функціонал інструменту.

```
buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:4.4.0'
    }
}// Top-level build file where you can add configuration options common to all sub-projects/modules.
plugins {
    id 'com.android.application' version '8.4.0' apply false
    id 'com.android.library' version '8.4.0' apply false
    id 'com.google.gms.google-services' version '4.4.0' apply false
}

allprojects {
    repositories {
        google()
        jcenter()
        maven { url "https://maven.google.com" }
        maven { url 'https://jitpack.io' }
    }
}
```

Рисунок 3.5.1 – Конфігураційний файл build.gradle інформаційної системи тренувань. Частина 1

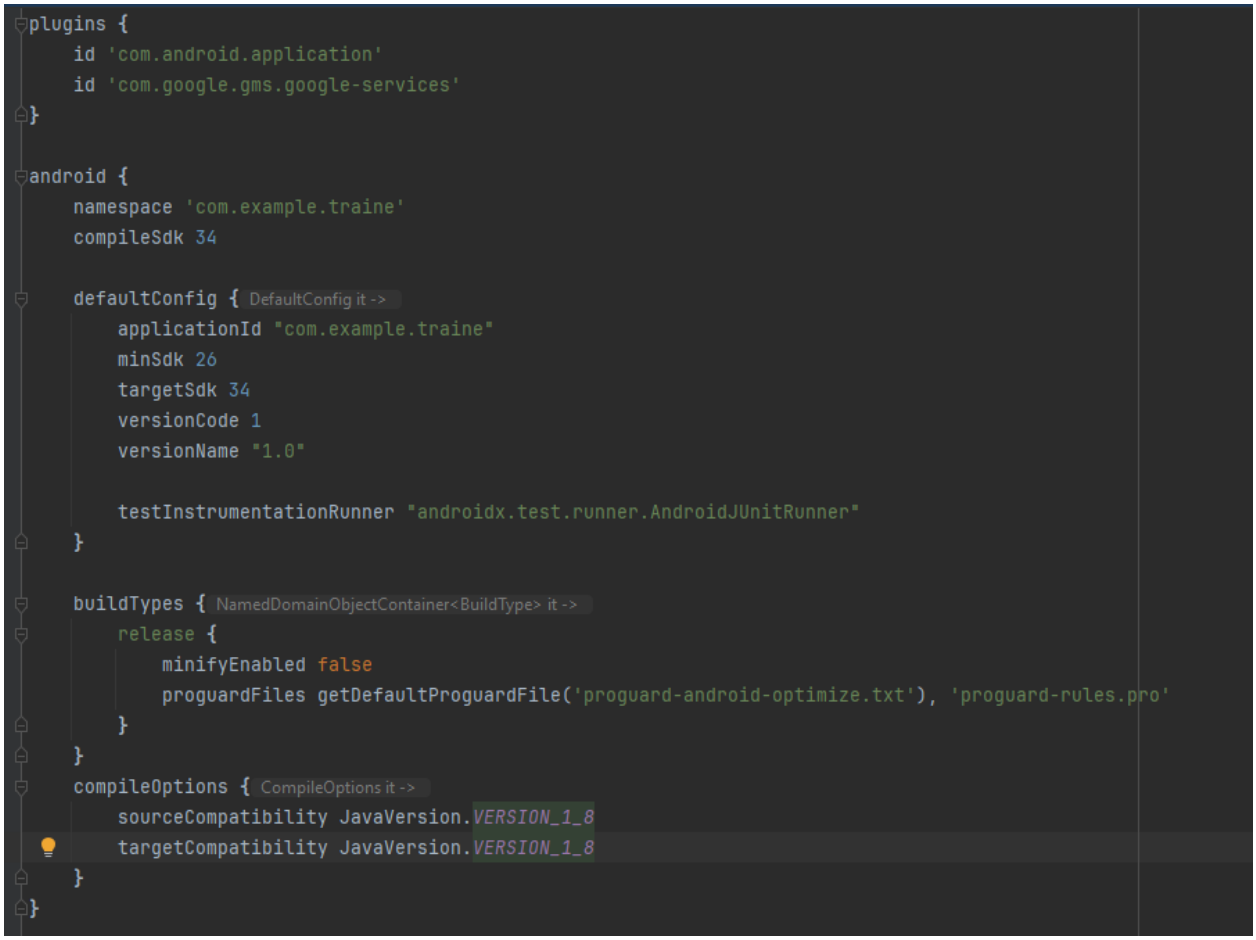


Рисунок 3.5.2 – Конфігураційний build.gradle файл інформаційної системи тренувань. Частина 2

У контексті розроблення мобільних додатків на Android, Gradle є особливо важливим, оскільки він є стандартною системою зборки для Android Studio, офіційного IDE для Android розроблення. Його інтеграція з Android SDK дозволяє автоматично управляти залежностями через Android Gradle плагін, що спрощує процес розроблення та розгортання Android додатків.

Інтеграція Gradle з Firebase, платформою для розроблення мобільних та веб-додатків, ще більше розширює можливості розробників. Firebase надає різноманітні облачні сервіси, які включають аутентифікацію, зберігання даних, аналітику, конфігурацію та відправку повідомлень. Використовуючи Gradle, розробники можуть легко додавати та управляти Firebase бібліотеками як частиною процесу збірки додатку, використовуючи механізм залежностей

Gradle для включення необхідних Firebase модулів. Таке рішення автоматизує процес підключення та конфігурації Firebase сервісів, забезпечуючи інтеграцію безпеки та ефективне управління ресурсами проекту.

Ця інтеграція відіграє ключову роль у забезпеченні плавності розроблення та розгортання додатків, оскільки дозволяє розробникам використовувати потужні можливості Firebase разом із автоматизованими інструментами Gradle для забезпечення високої продуктивності, надійності та масштабування їх додатків. Отже, використання Gradle у співпраці з Firebase є визначним прикладом того, як сучасні інструменти можуть сприяти більш ефективній та інтегрованій розробці програмного забезпечення в умовах, що швидко змінюються.

3.2.3 Інтеграція Firebase до системи

Для підтримки функціональності програми було використано низку бібліотек Firebase, які дозволяють реалізувати аутентифікацію користувачів, управління даними та зберігання інформації у хмарі. Інтеграція цих бібліотек була здійснена через файл налаштувань Gradle:

```
dependencies {

    // debugImplementation because LeakCanary should only run in debug builds.
    //debugImplementation 'com.squareup.leakcanary:leakcanary-android:2.14'
    // Import the Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:32.4.1')
    implementation 'com.google.firebase:firebase-analytics'
    implementation 'com.google.firebase:firebase-core:21.1.1'
    implementation 'com.google.firebase:firebase-auth:22.2.0'
    implementation 'com.google.firebase:firebase-database:20.3.0'

    implementation("com.google.firebase:firebase-storage")
    implementation 'com.firebaseui:firebase-ui-database:4.3.2'
```

Рисунок 3.5.3 – Конфігураційний build.gradle файл інформаційної системи тренувань з бібліотеками Firebase. Частина 3

Таблиця 3.1 – Опис використаних бібліотек Firebase

Бібліотека	Опис
com.google.firebase:firebase-bom:32.4.1	Firebase Bill of Materials (BoM), використовується для визначення версій інших бібліотек Firebase
com.google.firebase:firebase-analytics	Бібліотека для Firebase Analytics, використовується для збору та аналізу даних про використання додатку
com.google.firebase:firebase-core:21.1.1	Основна бібліотека Firebase, необхідна для роботи інших бібліотек Firebase
com.google.firebase:firebase-auth:22.2.0	Бібліотека для Firebase Authentication, використовується для аутентифікації користувачів через різні методи
com.google.firebase:firebase-database:20.3.0	Бібліотека для Firebase Realtime Database, використовується для зберігання та синхронізації даних у реальному часі
com.google.firebase:firebase-storage	Бібліотека для Firebase Storage, використовується для зберігання та керування файлами у хмарі
com.firebaseui:firebase-ui-database:4.3.2	Бібліотека для FirebaseUI, забезпечує готові користувацькі інтерфейси для взаємодії з Firebase Realtime Database

3.2.4 LeakCanary

LeakCanary є потужним інструментом для розробників Android-додатків, що спрощує процес виявлення витоків пам'яті у додатках. Цей

інструмент автоматично відстежує створені об'єкти та їхні посилання, щоб визначити, чи не зберігаються вони у пам'яті довше, ніж потрібно, тим самим потенційно викликаючи витік пам'яті.[13]

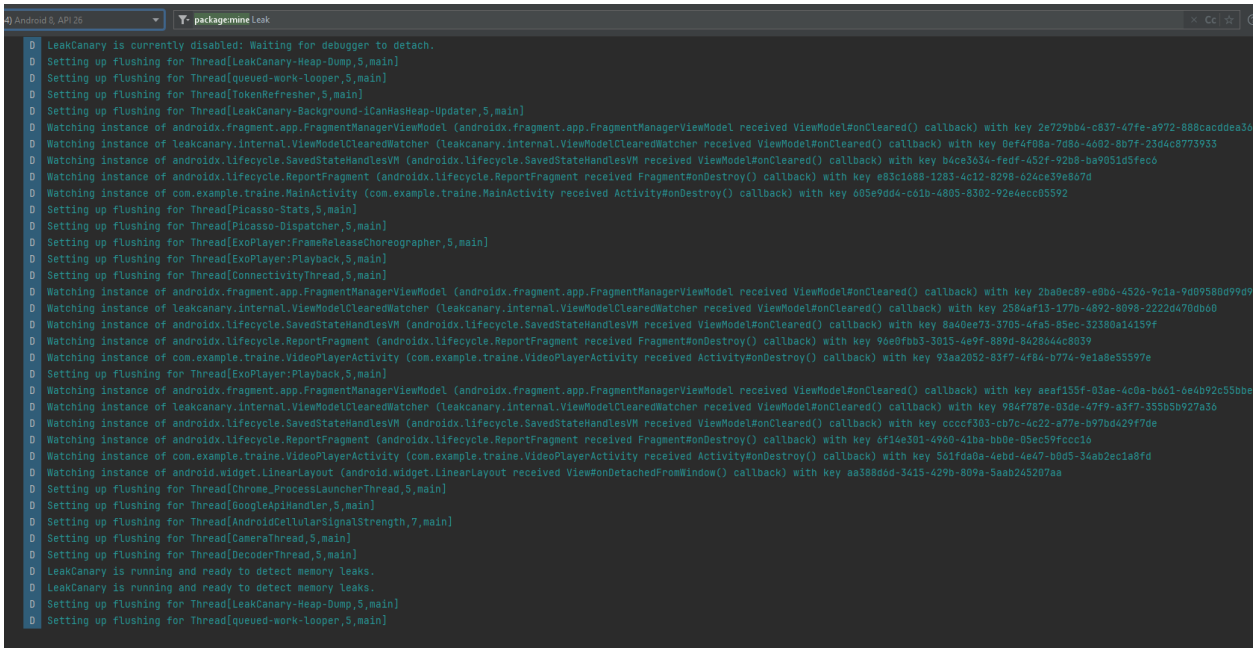


Рисунок 3.6 – Лог LeakCanary в інтерфейсі Android Logcat

В логах видно, що LeakCanary спочатку був вимкнений і очікував відключення відладчика. Це стандартна процедура безпеки, щоб забезпечити, що збір даних відбувається в умовах реальної роботи додатка, а не в режимі відладки.

Логи вказують на налаштування потоків для збору інформації про стан додатка, включаючи потоки для збору знімків пам'яті та обробки фонових задач. Це забезпечує, що збір даних про витоки пам'яті мінімально впливає на продуктивність додатка.

LeakCanary активно спостерігає за об'єктами, які отримали зворотні виклики про очищення або знищення (наприклад, ViewModel, Fragment, Activity), що є ключовим для виявлення витоків. Кожному об'єкту присвоюється унікальний ключ, який використовується для ідентифікації в логах.

Логи показують активність різних потоків у додатку, таких як обробка відео, збір статистики, обробка зображень (Picasso), що вказує на високу рівень паралельної обробки і потребу в точному відстеженні стану різних компонентів для виявлення витоків пам'яті.

З аналізу логів LeakCanary випливає, що інструмент активно моніторить додаток і виявляє потенційні витоки пам'яті. Однак у логах не було знайдено явних повідомлень, що підтверджують фактичні витоки, що свідчить про можливу ефективність наявних механізмів управління пам'яттю. Це вказує на високий рівень оптимізації додатку з точки зору управління пам'яттю.

3.3 Опис програмної реалізації

Структура проекту має 5 каталогів, кожен з яких відповідає за свою активність та бізнес логіку.

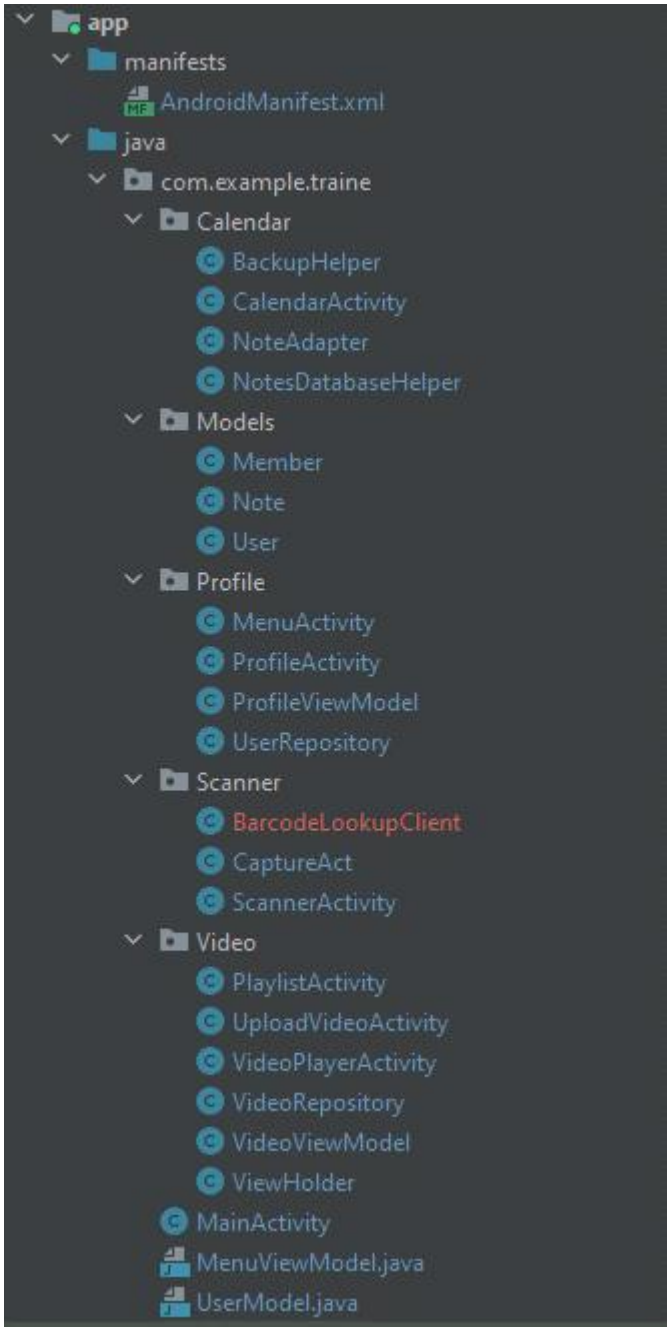


Рисунок 3.7 – Структура додатку

Каталог Models відповідає за класи, в яких зберігаються інформація про відео(клас Member), користувача(клас User) та його нотатки(клас Note). Ці

класи мають методи геттерів та сеттерів, за допомогою яких ми можемо записувати та отримувати дані.

Особливої уваги слід надати класу User, оскільки він тільки записує дані не тільки в локальному класі, а й має можливість передавати дані до інших класів. Усе це реалізовано за допомогою наслідування від класу Parcelable.

```
public class User implements Parcelable {
    5 usages
    private String name, email, pass, phone, profileUri, uid;

    public User() {}

    public User(String name, String email, String pass, String phone, String profileUri, String uid) {
        this.name = name;
        this.email = email;
        this.phone = phone;
        this.profileUri = profileUri;
        this.pass = pass;
        this.uid = uid;
    }

    protected User(Parcel in) {
        name = in.readString();
        email = in.readString();
        phone = in.readString();
        profileUri = in.readString();
        pass = in.readString();
        uid = in.readString();
    }

    public static final Creator<User> CREATOR = new Creator<User>() {
        @Override
        public User createFromParcel(Parcel in) { return new User(in); }

        @Override
        public User[] newArray(int size) { return new User[size]; }
    };
```

Рисунок 3.8 – Клас User

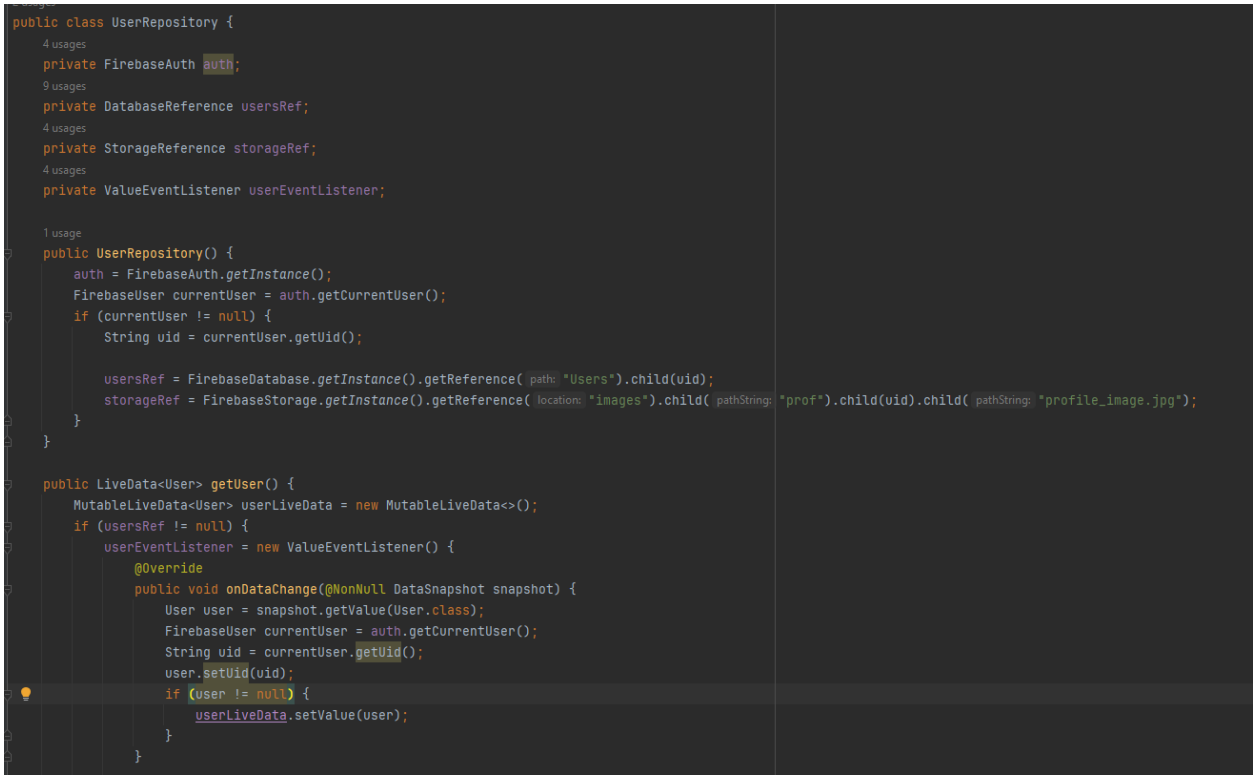
В каталозі Profile містяться класи активностей MenuActivity та ProfileActivity, які відповідають за відображення даних користувача. Логіка для цих класів прописана в ProfileViewModel, який відповідає за передачу даних до активностей. Для отримання даних з бази даних Firebase ми

використовуємо модель UserRepository, яка завантажує дані, обробляє їх та записує в клас User.

```java
package com.example.traine.Profile;

import ...

4 usages
public class ProfileViewModel extends ViewModel {
    5 usages
    private UserRepository userRepository;
    4 usages
    private MutableLiveData<User> userLiveData;

    no usages
    public ProfileViewModel() {
        userRepository = new UserRepository();
        userLiveData = new MutableLiveData<>();
        loadUser();
    }

    public LiveData<User> getUser() { return userLiveData; }

    1 usage
    public void updateUserInformation(String type, String newValue, String currentPassword) {
        userRepository.updateUserInformation(type, newValue, currentPassword, task -> {
            if (task.isSuccessful()) {
                loadUser(); // Refresh user data
            }
        });
    }

    1 usage
    public void uploadImage(Uri imageUri) {
        userRepository.uploadImage(imageUri, task -> {
            if (task.isSuccessful()) {
                loadUser(); // Refresh user data
            }
        });
    }

    public void setUser(User user) { userLiveData.setValue(user); }
```

Рисунок 3.9 – Клас ProfileViewModel

Оскільки архітектура профіля є MVVM, то важливо прослідковувати зміни в даних користувача та відображати зміни, якщо такі були. Для такої реалізації використовується параметр MutableLiveData<User> userLiveData.[16]



```
public class UserRepository {
    4 usages
    private FirebaseAuth auth;
    9 usages
    private DatabaseReference usersRef;
    4 usages
    private StorageReference storageRef;
    4 usages
    private ValueEventListener userEventListener;

    1 usage
    public UserRepository() {
        auth = FirebaseAuth.getInstance();
        FirebaseUser currentUser = auth.getCurrentUser();
        if (currentUser != null) {
            String uid = currentUser.getUid();

            usersRef = FirebaseDatabase.getInstance().getReference( path: "Users").child(uid);
            storageRef = FirebaseStorage.getInstance().getReference( location: "images").child( pathString: "prof").child(uid).child( pathString: "profile_image.jpg");
        }
    }

    public LiveData<User> getUser() {
        MutableLiveData<User> userLiveData = new MutableLiveData<>();
        if (usersRef != null) {
            userEventListener = new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    User user = snapshot.getValue(User.class);
                    FirebaseUser currentUser = auth.getCurrentUser();
                    String uid = currentUser.getUid();
                    user.setUid(uid);
                    if (user != null) {
                        userLiveData.setValue(user);
                    }
                }
```

Рисунок 3.10 – Клас UserRepository

В каталозі Video розміщено клас активності, який надає користувачу можливість перегляду списку відео. Через архітектуру MVVM реалізовано бізнес логіку для отримання даних з бази даних Firebase. За завантаження даних відповідає клас VideoRepositoty, а для передачі їх в активність використано той самий принцип, що й для профілю користувача, тобто зчитування даних відео в LiveData та передача через VideoViewModel.

Додаткової уваги слід також надати методам getFilteredVideoQuery(), який повертає список усіх відео з відповідним тегом й таким чином реалізовано фільтрацію відео для користувача.

```
public class VideoViewModel extends ViewModel {
    3 usages
    private VideoRepository videoRepository;
    1 usage
    private MutableLiveData<List<Member>> videos = new MutableLiveData<>();

    no usages
    public VideoViewModel() { videoRepository = new VideoRepository(); }

    no usages
    public LiveData<List<Member>> getVideos() { return videos; }

    1 usage
    public Query getVideoQuery() { return videoRepository.getVideoQuery(); }

    1 usage
    public Query getFilteredVideoQuery(String tag) {
        return videoRepository.getFilteredVideoQuery(tag);
    }

    2 usages
    public void launchVideoPlayer(Context context, Member member) {
        Intent intent = new Intent(context, VideoPlayerActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intent.putExtra( name: "video_title", member.getVideoName());
        intent.putExtra( name: "video_uri", member.getVideoUri());
        context.startActivity(intent);
    }
}
```

Рисунок 3.11 – Клас VideoViewModel

```
2 usages
public class VideoRepository {
    3 usages
    private DatabaseReference videoReference;

    1 usage
    public VideoRepository() {
        FirebaseDatabase db = FirebaseDatabase.getInstance();
        videoReference = db.getReference( path: "Video");
    }

    1 usage
    public Query getVideoQuery() { return videoReference; }

    1 usage
    public Query getFilteredVideoQuery(String tag) {
        return videoReference.orderByChild( path: "videoTags").equalTo(tag);
    }

}
```

Рисунок 3.12 – Клас VideoRepository

Оскільки дані відео вже завантажені з Firebase, то можемо передавати їх до плеєру, який в свою чергу, відображає дані користувача. Важливо зазначити, що дані до плеєра передаються, тільки при відкритті відео та у форматі String. Плеєр, який налаштовано в класі VideoPlayerActivity зчитує отримане посилання на відео та завантажує його з Firebase Storage.

```java
public class VideoPlayerActivity extends AppCompatActivity {
    3 usages
    PlayerView playerView;
    15 usages
    SimpleExoPlayer exoPlayer;
    2 usages
    String videoUri;

    2 usages
    ImageView buttonMain;
    2 usages
    String videoTitle;
    2 usages
    TextView title;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_video_player);
        initViews();
        fetchDataFromIntent();
        initializePlayer();
    }

    1 usage
    private void initViews() {
        playerView = findViewById(R.id.exoplayer_view);
        title = findViewById(R.id.video_title);
        buttonMain = findViewById(R.id.buttonToMainActivity);
        buttonMain.setOnClickListener(view -> finish());
    }

    1 usage
    private void fetchDataFromIntent() {
        videoTitle = getIntent().getStringExtra( name: "video_title");
        videoUri = getIntent().getStringExtra( name: "video_uri");
        title.setText(videoTitle);
    }
```

Рисунок 3.13 – Клас VideoPlayerActivity

Тека Calender відповідає за календар користувача та нотатки в ньому. Для збереження даних календаря в системі реалізовано базу даних SQLite, яка зберігається у пам’яті пристрою та оновлюється через FirebaseStorage в залежності від ідентифікатора користувача в системі.

За створення та керування базою даних відповідає клас NotesDatabaseHelper.

```
public class NotesDatabaseHelper extends SQLiteOpenHelper {

    1 usage
    private static final String DATABASE_NAME = "notes.db";
    1 usage
    private static final int DATABASE_VERSION = 1;

    6 usages
    private static final String TABLE_NOTES = "notes";
    5 usages
    private static final String COLUMN_ID = "_id";
    6 usages
    private static final String COLUMN_YEAR = "year";
    6 usages
    private static final String COLUMN_MONTH = "month";
    6 usages
    private static final String COLUMN_DAY = "day";
    5 usages
    private static final String COLUMN_TEXT = "text";

    1 usage
    private static final String TABLE_CREATE =
            "CREATE TABLE " + TABLE_NOTES + " (" +
                    COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    COLUMN_YEAR + " INTEGER, " +
                    COLUMN_MONTH + " INTEGER, " +
                    COLUMN_DAY + " INTEGER, " +
                    COLUMN_TEXT + " TEXT" +
                    ");";

    1 usage
    public NotesDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, factory: null, DATABASE_VERSION);
    }
```

Рисунок 3.14 – Клас NotesDatabaseHelper

Таблиця 3.2 – Опис таблиці SQLite

Назва поля	Тип	Опис	Обмеження
_id	INTEGER	Ідентифікатор нотатки	PRIMARY KEY AUTOINCREMENT
year	INTEGER	Рік створення нотатки	
month	INTEGER	Місяць створення	

		нотатки	
day	INTEGER	День створення нотатки	
text	TEXT	Текст нотатки	

Оновлення бази даних SQLite в FirebaseStorage реалізоване через клас BackupHelper.

```
public class BackupHelper {

    3 usages
    private Context context;
    2 usages
    private FirebaseStorage storage;
    4 usages
    private StorageReference storageReference;
    8 usages
    private User user;

    1 usage
    public BackupHelper(Context context, User user) {
        this.context = context;
        this.storage = FirebaseStorage.getInstance();
        this.storageReference = storage.getReference();
        this.user = user;
    }

    2 usages
    public void backupDatabaseToFirebase() {
        if (user == null) {
            Log.e( tag: "BackupHelper",  msg: "User is null");
            return;
        }

        File database = context.getDatabasePath( s: "notes.db");

        if (database.exists()) {
            Uri file = Uri.fromFile(database);
            Log.d( tag: "Firebase",  msg: "Path to db"+"backups/" + user.getUid() + "/notes.db");
            StorageReference databaseRef = storageReference.child( pathString: "backups/" + user.getUid() + "/notes.db");

            databaseRef.putFile(file) UploadTask
                    .addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
```

Рисунок 3.15 – Клас BackupHelper

ВИСНОВОК

Використання сучасних технологій та стандартів програмування дозволило створити ефективну інформаційну систему, орієнтовану на потреби користувачів, які бажають оптимізувати свої тренування за допомогою мобільних технологій. Серверна частина системи забезпечує стабільну роботу додатку, високу швидкість обробки даних і зручність використання, що робить її надійним інструментом для планування та аналізу тренувань в різних спортивних дисциплінах.

Використання Firebase як основи для серверної частини дозволило виконати централізоване управління користувацькими даними, аутентифікацією, зберіганням файлів та іншими функціями, необхідними для забезпечення функціональності мобільного додатку. Ця хмарна платформа від Google забезпечує високу надійність, доступність сервісів та масштабованість рішень, що є критично важливим для спортивних додатків, де користувачі очікують безперебійної та ефективної роботи системи.

Окремої уваги заслуговує аспект витоків пам'яті, який може значно погіршити продуктивність та користувацький досвід. Використання інструменту LeakCanary в розробці допомагає виявляти та усувати витоки пам'яті в Android додатках, забезпечуючи стабільність та надійність системи. LeakCanary є ефективним інструментом для моніторингу використання пам'яті у реальному часі, ідентифікації неналежного управління пам'яттю та допомагає вирішити проблеми до випуску додатку на ринок.

СПИСОК ЛІТЕРАТУРИ

1) Glučina, M., & Pleić, T. (2024). DIGITAL EDUCATIONAL CONTENT-THE CONCEPT OF HISTORY INTERACTIVE VIDEO LESSONS. In *INTED2024 Proceedings* (pp. 1556-1561).
2) Teodorescu, S., Bota, A., Popescu, V., Mezei, M., & Urzeala, C. (2021). Sports training during COVID-19 first lockdown—A romanian coaches' experience. *Sustainability*, *13*(18), 10275.
3) Lazareska, L., & Jakimoski, K. (2017). Analysis of the advantages and Disadvantages of Android and iOS Systems and Converting Applications from Android to iOS Platform and Vice Versa. *American Journal of Software Engineering and Applications*, *6*(5), 116-120.
4) Gascoigne, E., Vladutiu, C., Smith-Ryan, A., Dude, A., & Manuck, T. (2024). 174 Physical activity and sedentary behavior and spontaneous preterm birth (SPTB): Insights from Fitbit tracker data. *American Journal of Obstetrics & Gynecology*, *230*(1), S109-S110.
5) MyFitnessPal. Лічильник калорій. [Електронний ресурс] – Режим доступу: https://www.myfitnesspal.com (дата звернення 28/04/2024)
6) Strava. Журнал тренувань. [Електронний ресурс] – Режим доступу: https://www.strava.com/features (дата звернення 28/04/2024)
7) Daoudi, A., ElBoussaidi, G., Moha, N., & Kpodjedo, S. (2019, April). An exploratory study of mvc-based architectural patterns in android apps. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (pp. 1711-1720).
8) Khawas, C., & Shah, P. (2018). Application of firebase in android app development-a study. *International Journal of Computer Applications*, *179*(46), 49-53.
9) Документація розробника для Firebase [Електроний ресурс] URL – https://firebase.google.com/docs (дата звернення 03/05/2024)
10) Wolfson, M., & Felker, D. (2013). *Android developer tools essentials: Android Studio to Zipalign*. " O'Reilly Media, Inc.".

11) Pelgrims, K. (2015). *Gradle for Android*. Packt Publishing Ltd.
12) Gradle. Система автоматичної збірки. Документація. [Електронний ресурс] – Режим доступу: https://docs.gradle.org/current/userguide/userguide.html
13) LeakCanary - бібліотека для виявлення витоків пам'яті для Android. Firebase [Електроний ресурс] URL - https://square.github.io/leakcanary/ (дата звернення 03/05/2024)
14))Oliveira, W., Oliveira, R., & Castor, F. (2017, May). A study on the energy consumption of android app development approaches. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)* (pp. 42-52).
15) Microsoft Documentation. (n.d.). "Best Practices for Cloud Applications". [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/best-practices/ (дата звернення 03/05/2024)
16) MutableLiveData. Java клас. [Електронний ресурс] – Режим доступу: https://developer.android.com/reference/android/arch/lifecycle/MutableLiveData (дата звернення 07/05/2024)
17) Інформаційна система організації процесу тренувань. Github. [Електронний ресурс] – Режим доступу: https://github.com/PikabuT0T/Trainee (дата звернення 28/05/2024)

ДОДАТОК А

ProfileViewModel.java

```
package com.example.traine.Profile;

import android.net.Uri;

import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

import com.example.traine.Models.User;

public class ProfileViewModel extends ViewModel {
    private UserRepository userRepository;
    private MutableLiveData<User> userLiveData;

    public ProfileViewModel() {
        userRepository = new UserRepository();
        userLiveData = new MutableLiveData<>();
        loadUser();
    }

    public LiveData<User> getUser() {
        return userLiveData;
    }

    public void updateUserInformation(String type, String newValue, String
currentPassword) {
        userRepository.updateUserInformation(type, newValue, currentPassword,
task -> {
            if (task.isSuccessful()) {
                loadUser(); // Refresh user data
            }
        });
    }

    public void uploadImage(Uri imageUri) {
        userRepository.uploadImage(imageUri, task -> {
            if (task.isSuccessful()) {
                loadUser(); // Refresh user data
            }
        });
    }

    public void setUser(User user) {
        userLiveData.setValue(user);
    }

    private void loadUser() {
        userRepository.getUser().observeForever(user -> {
            if (user != null) {
                userLiveData.setValue(user);
            }
        });
    }

    public void detachListener() {
        userRepository.detachListener();
    }
}
```

UserRepository.java

```
package com.example.traine.Profile;

import android.app.Activity;
import android.net.Uri;

import androidx.annotation.NonNull;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;

import java.util.concurrent.Executor;

import com.example.traine.Models.User;

public class UserRepository {
    private FirebaseAuth auth;
    private DatabaseReference usersRef;
    private StorageReference storageRef;
    private ValueEventListener userEventListener;

    public UserRepository() {
        auth = FirebaseAuth.getInstance();
        FirebaseUser currentUser = auth.getCurrentUser();
        if (currentUser != null) {
            String uid = currentUser.getUid();

            usersRef =
FirebaseDatabase.getInstance().getReference("Users").child(uid);
            storageRef =
FirebaseStorage.getInstance().getReference("images").child("prof").child(uid)
.child("profile_image.jpg");
        }
    }

    public LiveData<User> getUser() {
        MutableLiveData<User> userLiveData = new MutableLiveData<>();
        if (usersRef != null) {
            userEventListener = new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot snapshot) {
                    User user = snapshot.getValue(User.class);
                    FirebaseUser currentUser = auth.getCurrentUser();
                    String uid = currentUser.getUid();
                    user.setUid(uid);
                    if (user != null) {
                        userLiveData.setValue(user);
                    }
                }
```

```
                @Override
                public void onCancelled(@NonNull DatabaseError error) {
                    // Handle error
                }
            };
            usersRef.addValueEventListener(userEventListener);
        }
        return userLiveData;
    }

    public void detachListener() {
        if (usersRef != null && userEventListener != null) {
            usersRef.removeEventListener(userEventListener);
        }
    }

    public void updateUserInformation(String type, String newValue, String
currentPassword, OnCompleteListener<Void> onCompleteListener) {
        FirebaseUser user = auth.getCurrentUser();
        if (user == null) {
            return;
        }

        Task<Void> updateTask;

        switch (type) {
            case "email":
                updateTask = user.updateEmail(newValue);
                break;
            case "password":
                updateTask = user.updatePassword(newValue);
                break;
            case "name":
                updateTask = usersRef.child("name").setValue(newValue);
                break;
            case "phone":
                updateTask = usersRef.child("phone").setValue(newValue);
                break;
            default:
                updateTask = usersRef.child(type).setValue(newValue);
                break;
        }

        updateTask.addOnCompleteListener(onCompleteListener);
    }

    public void uploadImage(Uri imageUri, OnCompleteListener<Uri>
onCompleteListener) {
        if (storageRef != null) {
            storageRef.putFile(imageUri).addOnSuccessListener(taskSnapshot ->
{
                storageRef.getDownloadUrl().addOnSuccessListener(uri -> {
                    String newUri = uri.toString();

usersRef.child("profileUri").setValue(newUri).addOnCompleteListener(task -> {
                        if (task.isSuccessful()) {
                            onCompleteListener.onComplete(new Task<Uri>() {
                                @Override
                                public boolean isComplete() {
                                    return true;
                                }

                                @Override
```

```
                                public boolean isSuccessful() {
                                    return true;
                                }

                                @Override
                                public boolean isCanceled() {
                                    return false;
                                }

                                @Override
                                public Uri getResult() {
                                    return uri;
                                }

                                @Override
                                public <X extends Throwable> Uri
getResult(@NonNull Class<X> aClass) throws X {
                                    return uri;
                                }


                                @NonNull
                                @Override
                                public Task<Uri>
addOnFailureListener(@NonNull OnFailureListener onFailureListener) {
                                    return null;
                                }

                                @NonNull
                                @Override
                                public Task<Uri>
addOnFailureListener(@NonNull Activity activity, @NonNull OnFailureListener
onFailureListener) {
                                    return null;
                                }

                                @NonNull
                                @Override
                                public Task<Uri>
addOnFailureListener(@NonNull Executor executor, @NonNull OnFailureListener
onFailureListener) {
                                    return null;
                                }

                                @NonNull
                                @Override
                                public Task<Uri>
addOnSuccessListener(@NonNull OnSuccessListener<? super Uri>
onSuccessListener) {
                                    return null;
                                }

                                @NonNull
                                @Override
                                public Task<Uri>
addOnSuccessListener(@NonNull Activity activity, @NonNull OnSuccessListener<?
super Uri> onSuccessListener) {
                                    return null;
                                }

                                @NonNull
                                @Override
                                public Task<Uri>
```

```
addOnSuccessListener(@NonNull Executor executor, @NonNull OnSuccessListener<?
super Uri> onSuccessListener) {
                                    return null;
                                }

                                @Override
                                public Exception getException() {
                                    return null;
                                }
                            });
                        }
                    });
                });
            });
        }
    }
}
```

User.java

```
package com.example.traine.Models;

import android.os.Parcel;
import android.os.Parcelable;

public class User implements Parcelable {
    private String name, email, pass, phone, profileUri, uid;

    public User() {}

    public User(String name, String email, String pass, String phone, String
profileUri, String uid) {
        this.name = name;
        this.email = email;
        this.phone = phone;
        this.profileUri = profileUri;
        this.pass = pass;
        this.uid = uid;
    }

    protected User(Parcel in) {
        name = in.readString();
        email = in.readString();
        phone = in.readString();
        profileUri = in.readString();
        pass = in.readString();
        uid = in.readString();
    }

    public static final Creator<User> CREATOR = new Creator<User>() {
        @Override
        public User createFromParcel(Parcel in) {
            return new User(in);
        }

        @Override
        public User[] newArray(int size) {
            return new User[size];
        }
    };

    public String getName() {
        return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPass() {
        return pass;
    }

    public void setPass(String pass) {
        this.pass = pass;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getProfileUri() {
        return profileUri;
    }

    public void setProfileUri(String profileUri) {
        this.profileUri = profileUri;
    }

    public String getUid() {
        return uid;
    }

    public void setUid(String uid) {
        this.uid = uid;
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);
        dest.writeString(email);
        dest.writeString(phone);
        dest.writeString(profileUri);
        dest.writeString(pass);
        dest.writeString(uid);
    }
}
```

VideoViewModel.java

```
package com.example.traine.Video;

import android.content.Context;
import android.content.Intent;

import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

import com.example.traine.Models.Member;
import com.google.firebase.database.Query;

import java.util.List;

public class VideoViewModel extends ViewModel {
    private VideoRepository videoRepository;
    private MutableLiveData<List<Member>> videos = new MutableLiveData<>();

    public VideoViewModel() {
        videoRepository = new VideoRepository();
    }

    public LiveData<List<Member>> getVideos() {
        return videos;
    }

    public Query getVideoQuery() {
        return videoRepository.getVideoQuery();
    }

    public Query getFilteredVideoQuery(String tag) {
        return videoRepository.getFilteredVideoQuery(tag);
    }

    public void launchVideoPlayer(Context context, Member member) {
        Intent intent = new Intent(context, VideoPlayerActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        intent.putExtra("video_title", member.getVideoName());
        intent.putExtra("video_uri", member.getVideoUri());
        context.startActivity(intent);
    }
}
```

VideoRepository.java

```
package com.example.traine.Video;

import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.Query;


public class VideoRepository {
    private DatabaseReference videoReference;

    public VideoRepository() {
        FirebaseDatabase db = FirebaseDatabase.getInstance();
        videoReference = db.getReference("Video");
    }

    public Query getVideoQuery() {
        return videoReference;
    }
```

```
    public Query getFilteredVideoQuery(String tag) {
        return videoReference.orderByChild("videoTags").equalTo(tag);
    }

}
```

Member.java

```
package com.example.traine.Models;

public class Member {
    String videoUri;
    String videoName;
    String videoPreviewImageUri;
    String videoDuration;
    String videoTags;

    public Member() {
    }

    public Member(String videoUri, String videoName, String videoDuration,
String videoPreviewImageUri, String videoTags) {
        this.videoUri = videoUri;
        this.videoName = videoName;
        this.videoDuration = videoDuration;
        this.videoPreviewImageUri = videoPreviewImageUri;
        this.videoTags = videoTags;
    }

    public String getVideoUri() {
        return videoUri;
    }

    public void setVideoUri(String videoUri) {
        this.videoUri = videoUri;
    }

    public String getVideoName() {
        return videoName;
    }

    public void setVideoName(String videoName) {
        this.videoName = videoName;
    }

    public String getVideoPreviewImage() {
        return videoPreviewImageUri;
    }

    public void setVideoPreviewImage(String videoPreviewImage) {
        this.videoPreviewImageUri = videoPreviewImage;
    }

    public String getVideoDuration() {
        return videoDuration;
    }

    public void setVideoDuration(String videoDuration) {
        this.videoDuration = videoDuration;
    }

    public String getVideoTags() {
```

```
        return videoTags;
    }

    public void setVideoTags(String videoTags) {
        this.videoTags = videoTags;
    }
}
```

NotesDatabaseHelper.java

```
package com.example.traine.Calendar;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import com.example.traine.Models.Note;

import java.util.ArrayList;
import java.util.List;

public class NotesDatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "notes.db";
    private static final int DATABASE_VERSION = 1;

    private static final String TABLE_NOTES = "notes";
    private static final String COLUMN_ID = "_id";
    private static final String COLUMN_YEAR = "year";
    private static final String COLUMN_MONTH = "month";
    private static final String COLUMN_DAY = "day";
    private static final String COLUMN_TEXT = "text";

    private static final String TABLE_CREATE =
            "CREATE TABLE " + TABLE_NOTES + " (" +
                    COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    COLUMN_YEAR + " INTEGER, " +
                    COLUMN_MONTH + " INTEGER, " +
                    COLUMN_DAY + " INTEGER, " +
                    COLUMN_TEXT + " TEXT" +
                    ");";

    public NotesDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(TABLE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NOTES);
        onCreate(db);
    }

    public void createNewTable(){
        SQLiteDatabase db = this.getReadableDatabase();
        onUpgrade(db, 1,1);
```

```
    }
    public void addNote(Note note) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(COLUMN_YEAR, note.getYear());
        values.put(COLUMN_MONTH, note.getMonth());
        values.put(COLUMN_DAY, note.getDayOfMonth());
        values.put(COLUMN_TEXT, note.getText());

        db.insert(TABLE_NOTES, null, values);
        db.close();
    }

    public List<Note> getNotesForDate(int year, int month, int day) {
        List<Note> notes = new ArrayList<>();
        SQLiteDatabase db = this.getReadableDatabase();

        String[] columns = {
                COLUMN_ID,
                COLUMN_YEAR,
                COLUMN_MONTH,
                COLUMN_DAY,
                COLUMN_TEXT
        };

        String selection = COLUMN_YEAR + " = ? AND " + COLUMN_MONTH + " = ?
AND " + COLUMN_DAY + " = ?";
        String[] selectionArgs = {String.valueOf(year),
String.valueOf(month), String.valueOf(day)};

        Cursor cursor = db.query(TABLE_NOTES, columns, selection,
selectionArgs, null, null, null);

        if (cursor.moveToFirst()) {
            do {
                int id =
cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_ID));
                int noteYear =
cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_YEAR));
                int noteMonth =
cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_MONTH));
                int noteDay =
cursor.getInt(cursor.getColumnIndexOrThrow(COLUMN_DAY));
                String text =
cursor.getString(cursor.getColumnIndexOrThrow(COLUMN_TEXT));

                Note note = new Note(id, noteYear, noteMonth, noteDay, text);
                notes.add(note);
            } while (cursor.moveToNext());
        }
        cursor.close();
        db.close();

        return notes;
    }

    public void updateNote(Note note) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(COLUMN_YEAR, note.getYear());
        values.put(COLUMN_MONTH, note.getMonth());
        values.put(COLUMN_DAY, note.getDayOfMonth());
        values.put(COLUMN_TEXT, note.getText());
```

```
        String selection = COLUMN_ID + " = ?";
        String[] selectionArgs = { String.valueOf(note.getId()) };

        db.update(TABLE_NOTES, values, selection, selectionArgs);
        db.close();
    }

    public void deleteNoteById(int id) {
        SQLiteDatabase db = this.getWritableDatabase();
        String selection = COLUMN_ID + " = ?";
        String[] selectionArgs = { String.valueOf(id) };
        db.delete(TABLE_NOTES, selection, selectionArgs);
        db.close();
    }

    public interface NotesCallback {
        void onCallback(List<Note> notes);
    }
}
```

BackupHelper.java

```
package com.example.traine.Calendar;

import android.content.Context;
import android.net.Uri;
import android.util.Log;

import androidx.annotation.NonNull;

import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.storage.FileDownloadTask;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageMetadata;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;

import java.io.File;

import com.example.traine.Models.User;

public class BackupHelper {

    private Context context;
    private FirebaseStorage storage;
    private StorageReference storageReference;
    private User user;

    public BackupHelper(Context context, User user) {
        this.context = context;
        this.storage = FirebaseStorage.getInstance();
        this.storageReference = storage.getReference();
        this.user = user;
    }

    public void backupDatabaseToFirebase() {
        if (user == null) {
            Log.e("BackupHelper", "User is null");
            return;
        }
```

```
        File database = context.getDatabasePath("notes.db");

        if (database.exists()) {
            Uri file = Uri.fromFile(database);
            Log.d("Firebase", "Path to db"+"backups/" + user.getUid() +
"/notes.db");
            StorageReference databaseRef = storageReference.child("backups/"
+ user.getUid() + "/notes.db");

            databaseRef.putFile(file)
                    .addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() {
                        @Override
                        public void onSuccess(UploadTask.TaskSnapshot
taskSnapshot) {
                            Log.d("Firebase", "Database backup successful");
                        }
                    })
                    .addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            Log.e("Firebase", "Database backup failed", e);
                        }
                    });
        } else {
            Log.e("BackupHelper", "Database file does not exist");
        }
    }

    public void downloadDatabaseFromFirebase(final BackupCallback callback) {
        if (user == null) {
            Log.e("BackupHelper", "User is null");
            callback.onFailure();
            return;
        }

        final File localFile = context.getDatabasePath("notes.db");
        StorageReference databaseRef = storageReference.child("backups/" +
user.getUid() + "/notes.db");

        databaseRef.getFile(localFile)
                .addOnSuccessListener(new
OnSuccessListener<FileDownloadTask.TaskSnapshot>() {
                    @Override
                    public void onSuccess(FileDownloadTask.TaskSnapshot
taskSnapshot) {
                        Log.d("Firebase", "Database download successful");
                        callback.onSuccess();
                    }
                })
                .addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        Log.e("Firebase", "Database download failed", e);
                        callback.onFailure();
                    }
                });
    }

    public void checkDatabaseExists(final BackupCallback callback) {
        if (user == null) {
            Log.e("BackupHelper", "User is null");
            callback.onFailure();
```

```
            return;
        }

        StorageReference databaseRef = storageReference.child("backups/" +
user.getUid() + "/notes.db");

        databaseRef.getMetadata()
                .addOnSuccessListener(new
OnSuccessListener<StorageMetadata>() {
                    @Override
                    public void onSuccess(StorageMetadata storageMetadata) {
                        // File exists
                        Log.d("Firebase", "Database file exists in storage");
                        callback.onSuccess();
                    }
                })
                .addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        // File does not exist

                        Log.e("Firebase", "Database file does not exist in
storage", e);
                        callback.onFailure();
                    }
                });
    }

    public interface BackupCallback {
        void onSuccess();
        void onFailure();
    }
}
```