Endterm Project Report - Native Mobile Development (Android / Kotlin)

Student: Amir Amanshin from SE-2427.
Project: Game Catalog (FreeToGame API)
Tech stack: Kotlin, Jetpack Compose, KSP, Firebase Auth, Firebase Realtime Database.

1. Project Overview

The goal of this project was to develop a native Android application using Kotlin and modern Android architecture principles.
The application provides a game catalog with search functionality, offline support, user authentication, and real-time comments.

The project demonstrates:

-Remote JSON API integration
-Offline-first data persistence
-Clean architecture (MVVM)
-Dependency injection (Hilt)
-Firebase authentication
-Real-time database synchronization

2. Application Features

2.1 Screens Implemented

The application contains the following screens:

1. **Login Screen**

-Email/password authentication via Firebase Auth
-Registration and login support

-Persistent user session

2. **Feed Screen**

   -Displays list of games from remote API
   -Data cached locally in Room
   -Offline viewing supported

3. **Search Screen**

   -Local search with debounce (400ms delay)
   -Case-insensitive filtering
   -Works offline using cached data

4. **Details Screen**

   -Shows full information about selected game
   -Allows adding/removing from favorites
   -Navigation to comments

5. **Favorites Screen**

   -Displays locally stored favorite games
   -Fully available offline
   -Logout option

6. **Comments Screen**

   -Real-time comments per game
   -Live updates using Firebase Realtime Database
   -Users can delete only their own comments

3. Remote API Integration

The application integrates with the **FreeToGame public API**, which provides game metadata in JSON format.

Technologies used:

Retrofit for HTTP requests

Moshi for JSON parsing

Data loading flow:

1. Retrofit fetches JSON from API.
2. Data is converted into DTO models.
3. DTO models are mapped into domain models.
4. Data is saved into Room database.
5. UI observes Room via Flow and updates automatically.

This ensures that the UI is not directly dependent on network calls.

4. Architecture

The project follows the **MVVM architecture pattern** with a Repository layer.

UI Layer (Jetpack Compose)

-Displays UiState
-Sends user events to ViewModel

ViewModel Layer

-Manages state via StateFlow
-Handles business logic
-Calls Repository methods

Repository Layer

-Single source of truth
-Coordinates between:

Remote API

Room database

Firebase services

Data Layer

-Retrofit API interfaces
-Room Entities and DAO
-Firebase Auth & Realtime Database

Dependency Injection is implemented using **Hilt**, allowing clear separation of concerns and testability.

5. Data Models

5.1 Local Database (Room)

GameEntity

- id (Primary Key)
- title
- thumbnailUrl
- shortDescription
- genre
- platform
- updatedAt

FavoriteEntity

- gameId (Primary Key)
- addedAt

5.2 Cloud Data (Firebase Realtime Database)

Comment

- id
- uid (user identifier)
- text
- createdAt

6. Offline-First Strategy

The application implements an offline-first approach:

-Feed data is cached in Room.
-UI observes local database instead of API.
-Details screen reads from local cache.
-Favorites are fully local.
-Search operates on cached data.
-Previously loaded data is available without internet connection.

This ensures better user experience and data availability.

7. Firebase Integration

7.1 Authentication

Firebase Authentication is used with email/password:

-User registration
-User login
-Session persistence
-Automatic navigation based on authentication state

7.2 Realtime Database

Comments feature uses Firebase Realtime Database:

-Comments are synchronized instantly
-Updates appear in real-time across devices
-Only authenticated users can write data
-Users can delete only their own comments

Security rules enforce user-level validation.

12. Conclusion

This project demonstrates the development of a modern Android application. The application meets the course requirements and showcases real-world Android development practices.