🎬 Final Project: Android Movie Tracker

Course: Android Development with Kotlin & Compose
Weight: 70% of Final Grade
Team Size: Individual or Groups of 2-3 Students

1. Project Overview

The goal of this project is to build a modern, production-ready Android application that serves as a personal movie database. The app must fetch live data from the internet, allow users to interact with that data, and persist user preferences locally on the device.

Your application will allow users to browse popular movies, view details, mark movies as **Favorites**, and give a personal **Star Rating (1-5)**.

2. Core Features (Functional Requirements)

A. The Movie Feed (Network)

- **API Integration:** The app must fetch data from **The Movie Database (TMDB)** or similar public API.
- **List View:** Display a list/grid of movies showing the poster, title, and release year.
- **Loading/Error States:** The UI must handle slow internet (Loading spinners) and connection failures (Error messages/Retry buttons).
- **Optional Feature: Search (Bonus Points)**
  - **Search Bar:** Implement a search field in the Top Bar.
  - **Query:** When the user types a title (e.g., "Batman"), the app should query the API and display matching results instead of the default popular list.
  - **UI Update:** The list should update dynamically to show the search results.

B. Movie Details (Navigation)

- Clicking a movie in the feed must navigate to a **Detail Screen**.
- Show full metadata (Synopsis, Genre, Release Date, High-res Poster).

C. Persistence (Room Database)

- **Favorites:** Users can "Heart" or "Favorite" a movie. This status must be saved in a local Room database.
- **Ratings:** Users can assign a 1-5 star rating to a movie. This rating must be saved in the Room database.
- **Offline Access:** The "Favorites" screen should display the user's saved movies even if the internet is disconnected.

D. User Interface (UI)

- Must be built entirely with **Jetpack Compose**.
- Must follow **Material 3** design guidelines (proper colors, typography, and spacing).
- Must support **Light Mode** and **Dark Mode** (automatic or toggle).

3. Technical Requirements (Non-Functional)

1. **Architecture:** You must strictly follow the **MVVM (Model-View-ViewModel)** pattern.
   - **UI:** No logic allowed. Only displays state.
   - **ViewModel:** Handles data transformation, API calls, and DB interactions.
   - **Repository:** (Optional but recommended) Acts as a single source of truth between API and Room.
2. **Coroutines:** All database and network operations must be performed asynchronously using Coroutines/Flow.
3. **Libraries:**
   - **Retrofit:** For networking.
   - **Coil:** For image loading.
   - **Room:** For local storage.
   - **Navigation Compose:** For screen transitions.

4. Deliverables

You will submit your project before 13/01/2025 via **Dropbox Link**:
https://www.dropbox.com/request/yMxoWzzWfUkko8sf2CjK

The folder must contain:

1. The Source Code (Link to git or a ZIP)

- Ensure no API Keys are hardcoded if you pushed to GitHub (if submitting just the zip, hardcoded keys are acceptable for grading purposes).

2. Video Demo (MP4 or Link)

- A short (2-3 minute) video showing the app in action.
- Walk through: Fetching the list, opening details, adding a favorite, rating a movie, restarting the app, and showing that the rating/favorite persisted.

3. Project Report (PDF)

- **Architecture Diagram:** Draw how your Views, ViewModels, and Data layers connect.
- **Challenges:** Explain the hardest part of the project and how you solved it.

- **Team Contribution:** (If working in a group) Explicitly state what each member worked on.
- **Screenshots:** 3-4 key screens of the app.

5. Grading Rubric (Total: 100 Points)

Category	**Points**	**Description**
Functionality	**35**	Does the app fetch data? Do favorites save? Do ratings persist? Is the flow bug-free? *(+5 Bonus points for working Search)*
Architecture (MVVM)	**25**	Is logic separated from UI? Are ViewModels used correctly? Is state managed properly?
Code Quality	**15**	Clean code, proper naming conventions, folder structure, proper use of Git.
UI/UX Design	**15**	Visual polish, correct use of Material 3 components, handling loading/error states.
Report & Video	**10**	Clarity of the report and demonstration video.

Good luck! Build something you are proud to show on your portfolio.