NANYANG POLYTECHNIC

Objectives:

In this assignment, you are to make use of what you have learnt in this module to design and develop a Movie viewer application on the Android platform using JetPack compose.

Scenario:

PopCornMovie is a company that allow their users to view movies details. They are looking for a mobile application that allows users to view movie details. They have decided to hire you to develop this application on the Android platform.

Deadline

- 8th Feb 2026, 23:59

This assignment holds a total of 30% of your final ICA. This paper has a total of 100 marks.

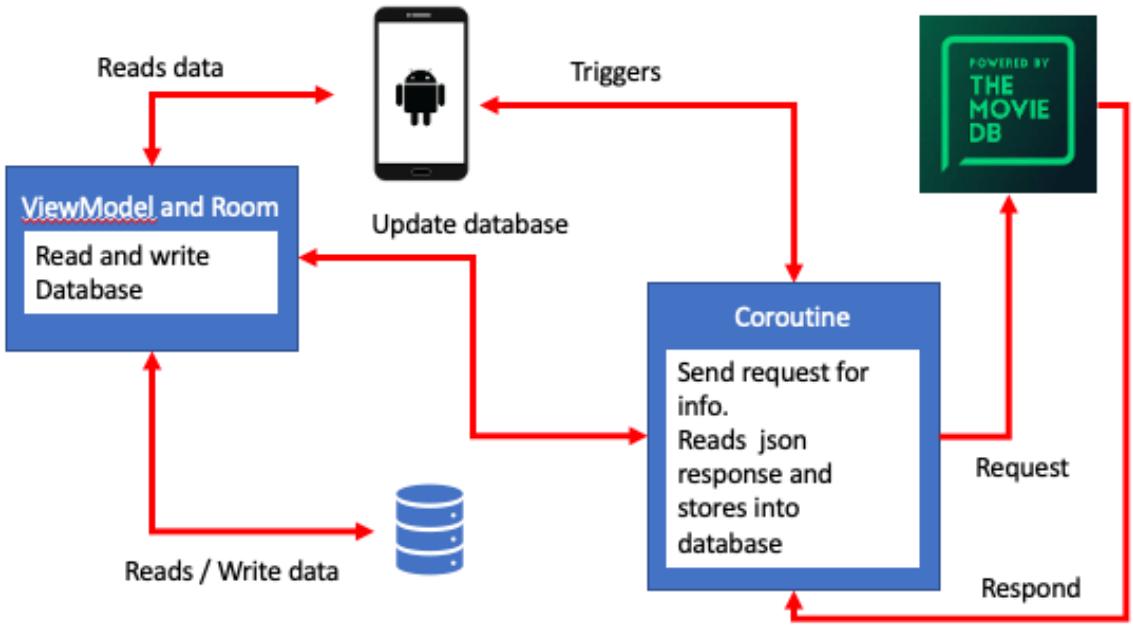
Listed below are the deliverables that is expected from you.
Deliverables:

- Zipped file with
    - source codes that fulfils the assignment requirements. (21212A.zip)
    - APK file. (21212A.apk)

Instructions:

- This is an INDIVIDUAL assignment. Students are to submit their assignment to BrightSpace.
- Zip up the above deliverables and name it after your admin no.
- Marks will be deducted for the following conditions.
    - Applications that cannot run upon first time installation
    - Late submissions
    - Not submitting work based on instructions given in the assignment
- Students caught plagiarizing from other source (Internet, friends, etc..) will cause their submission to be voided.
- All projects must be error free and is able to build and run upon open.
- Project details
    - Minimal SDK: API 34
    - Application Name: [Admin Number] Movie Viewer
    - Package: com.it2161.[Admin number].movieviewer
    - Emulator: Pixel 8 Pro

Overview



In this assignment, you will design and develop an Android application that retrieves movie data from **TheMovieDB (TMDB)** using **Retrofit**. The application will consume RESTful API endpoints provided by TMDB, which return data in **JSON** format.

Data retrieval must be performed **asynchronously using Kotlin coroutines**, and the retrieved data must be processed and persisted locally using **Room**. Application state and data operations should be managed through **ViewModels**, following recommended Android architectural practices.

Implementation

The sections below describe the functional features required for this mobile application. For each feature, the relevant API URL and expected request/response format are provided where applicable. Features without a specified URL do not require an API call to TheMovieDB.

Learners are required to:

- Review the feature specifications carefully
- Design an appropriate application architecture
- Implement the Android application based on the given requirements

All application data must be stored and retrieved using Room, regardless of whether it originates from an API request or is generated locally within the app.

NANYANG POLYTECHNIC

Basic

1. Login and registration
    - User must be able to register and login to access the application.
    - User must be able to use the camera to take a photo to use as profile picture
    - Details should include:
        - User Id
        - Date of birth
        - Password / Confirm password
        - Preferred name
        - Profile picture (to have a default picture if not available)

[2 marks]

2. View and Edit Profile
    - User must be able to view and edit the details mentioned in #1.

[2 marks]

3. Movie List

At the landing screen, the user must be able to toggle between the lists below.

- Popular [Default]
  URL: https://developer.themoviedb.org/reference/movie-popular-list
- Top Rated
  URL: https://developer.themoviedb.org/reference/movie-top-rated-list
- Now playing
  URL: https://developer.themoviedb.org/reference/movie-now-playing-list
- Upcoming
  URL: https://developer.themoviedb.org/reference/movie-upcoming-list

Each movie detail comes with an image, make use of the instructions in the following link to retrieve the image: https://developer.themoviedb.org/docs/image-basics

[20 marks]

4. Movie detail

The user must be able view the movie details after selecting it from a list of movies. Display the following details provided by "TheMovieDB".

adult	genres	Original language	Title
Release date	Run time	Vote count	overview
Vote average	revenue		

URL: https://developer.themoviedb.org/reference/movie-details

[5 marks]

5. Reviews

   Display the reviews written by other users regarding a movie.

   URL: https://developer.themoviedb.org/reference/movie-reviews

   [5 marks]

Advanced

1. Favorite movie list
   - The list will be stored locally using DataStore.
   - Tapping on an item in the list will navigate the user to a similar detail list as #4 above.

   [2 marks]

2. Search

   Search for movies by their original, translated and alternative titles.

   Display to the user the results provided by "TheMovieDB".

   [2 marks]

   URL: https://developer.themoviedb.org/reference/search-movie

3. Offline support
   - Online status must be displayed proactively as the network status changes
   - Current movie list or detail must be available when phone goes offline.
   - All other features will displayed as "unavailable" to the user.
     - E.g. User will not be able to view other movie details or toggle to other movie list during offline.

   [2 marks]

Rubric: Feature Completion & Architecture Quality

Level	Completion & Behaviour	Architecture & Tool Usage	% of Feature Marks
Complete with Correct Architecture	Feature is fully implemented and behaves correctly under normal and edge-case usage.	Proper MVVM applied; ViewModel manages state/business logic; Room/DataStore used correctly; clear separation of concerns.	**85% – 100%**
Complete with Minor Architectural Issues	Feature works correctly with minor functional or UI issues.	ViewModel and Room/DataStore are used, but with small flaws (e.g. redundant calls, weak layering).	**65% – 84%**
Complete but Architecture Weak / Inconsistent	Feature appears complete from a user perspective.	ViewModel/Room present but misused or inconsistently applied (e.g. logic split between UI and ViewModel).	**50% – 64%**
Functionally Present with Poor Architecture	Feature is partially implemented or works only in limited cases.	Little or no meaningful separation of concerns; architecture is unclear or incorrect.	**25% – 49%**
Not Implemented / Non-functional	Feature missing, broken, or crashes.	No meaningful architecture usage.	**0% – 24%**

UI/UX

The mobile application will also be accessed based on the following:

- Visual design
    - Balance:
        - Create visually balanced screens with a thoughtful distribution of elements (text, buttons, images).
        - Avoid clutter by leveraging white space effectively.
    - Alignment:
        - Align elements to a grid or structure for neatness and a professional appearance.
    - Hierarchy:
        - Use size, color, and placement to prioritize content and guide the user's attention.
        - For example, make the primary action button more prominent than secondary actions.
- Consistency
    - Consistent use of fonts, colors, iconography, and spacing throughout the app.
    - Ensure similar UI elements (e.g., buttons, cards) look and feel the same across screens
    - Similar actions should produce similar results across the app.
        - Example: A "Save" button should always be in the same location and behave the same way.
    - Provide consistent feedback (e.g., all errors use the same style of message box, success notifications follow the same format).
- Ease of use
    - Intuitive Navigation
        - Clear Structure
            - Navigation menus (e.g., bottom navigation bars, hamburger menus) should be simple and accessible.
            - Include a logical hierarchy that makes it easy for users to locate features and content.
        - Obvious Call-to-Action (CTA)
            - Buttons, links, and other actionable elements should be visually distinct and labeled with clear, concise text.
            - Use recognizable icons with text labels when necessary.

- Minimal Cognitive Load
  - Simplicity:
    - Minimize the number of steps required to complete tasks (e.g., searching for a movie or adding it to a watchlist).
    - Avoid overwhelming users with too many options or excessive information on a single screen.

- Feedback and error handling
  - State Indicators:
    - Clearly differentiate between active, inactive, and hovered/selected states of UI elements (e.g., buttons, tabs).
  - Loading Indicators:
    - Use spinners, progress bars, or skeleton screens to indicate loading and improve perceived performance.
  - Animations and Transitions:
    - Add subtle animations to enhance interactivity (e.g., button presses, screen transitions), but avoid overusing them.

[20 marks]

Rubrics for UI/UX

	Advance	Proficient	Functional	Developing
Visual design	• Fully adheres to material design • Layout is visually appealing, professional and polished • Smooth animations and transitions enhance the experience without overwhelming the user **6 marks**	• Adheres to Material Design 3 principles with minor inconsistencies. • Overall design is clean and aesthetically pleasing, but lacks polish or advanced features. **5 – 4 marks**	Layout is functional but lacks attention to detail or refinement **3 – 2 marks**	UI appears cluttered, outdated, or unprofessional. **1 mark**
Consistency	UI elements are consistently styled and behave predictably across all screens. **4 marks**	UI elements are mostly consistent, with occasional minor deviations. **3 marks**	Inconsistent styling across screens (e.g., varying fonts, button styles). **2 marks**	Significant inconsistencies in UI styling and behaviour. **1 mark**
Ease of use	• Intuitive navigation with clear call-to-actions and no unnecessary complexity. • Users can complete core tasks (e.g., searching for movies, viewing details, managing watchlist) easily and quickly **4 marks**	Navigation is straightforward, but some tasks may require additional effort or clarification. **3 marks**	Navigation is somewhat intuitive, but some features are difficult to find or use. **2 marks**	Navigation is confusing, with unclear or missing call-to-actions. **1 mark**
Feedback and error handling	• Users receive clear feedback on interactions (e.g., loading indicators, success/error messages). • Graceful error handling for API failures, empty states, or offline scenarios. **6 marks**	• Basic feedback is provided, but some actions lack visual or textual cues. • Error handling is present but not comprehensive. **5 – 4 marks**	Limited feedback; users may be unsure if actions are successful or if errors occur **3 –2 marks**	No feedback for user actions, and errors are not handled gracefully. **1 mark**

Q&A [40%]

Learners will be given 10 minutes to present their application and respond to a structured Q&A session.

This component assesses the learner's understanding of the generated code, their use of AI prompts, core Android concepts, and their ability to justify architectural, design, and implementation decisions.

The Q&A presentation will include:

- A short walkthrough of the application
- Explanation of at least one AI prompt used and how it was refined or iterated
- Explanation of selected code snippets (e.g. state handling, composables, navigation)
- Explanation of the application architecture and responsibilities of key components (e.g. UI, ViewModel, Repository)

During the presentation, you may be asked to:

- A short walkthrough of the application
- Explanation of at least one AI prompt used and how it was refined or iterated
- Explanation of selected code snippets (e.g. state handling, composables, navigation)
- Explanation of the application architecture and responsibilities of key components (e.g. UI, ViewModel, Repository)

What You Are Expected to Demonstrate

During the Q&A, you should be prepared to explain and discuss the following areas:

1. Code Understanding

- Explain how selected parts of your code work
- Describe how data flows through your application
- Explain how state is managed within your composables
- Demonstrate understanding of how Compose updates the UI

2. Use of AI Tools

- Describe at least one key prompt used and why it was written that way
- Explain any changes, fixes, or refactoring applied to AI-generated code

3. Android Fundamentals

- Explain relevant Android concepts used in your app, such as:

- @Composable functions
- State and recomposition
- ViewModel usage
- Lifecycle considerations
- Justify why these concepts were appropriate for your implementation

4. App Architecture & Design

- Describe the overall architecture of your application
- Explain the responsibilities of key components (e.g. UI, ViewModel, Repository)
- Justify how your design supports separation of concerns
- Explain how data and events move between layers of the app

Q&A Presentation Rubrics

Criteria	Advanced	Proficient	Developing	Beginning	Max
Code Understanding & Explanation	Explains code confidently and accurately, including data flow and Compose behaviour. **13 – 15 marks**	Explains most code correctly with minor gaps or imprecise terms. **10 – 12 marks**	Surface-level explanation of what the code does; limited explanation of how or why. **6 – 9 marks**	Unable to explain key parts or shows misunderstanding of generated code. **0 – 5 marks**	15
AI Prompting & Usage	Clearly explains prompt design, iterations, and refinement of AI output. **5 – 4 marks**	Explains prompts used with some justification and limited iteration. **3 – 8 marks**	Describes prompts used but little reasoning or refinement. **2 marks**	Unable to explain AI usage or demonstrates prompt copying without understanding. **0 – 1 marks**	10
Android Fundamentals	Strong understanding of relevant concepts (state, composables, ViewModel, lifecycle). **9 – 10 marks**	Reasonable understanding with minor inaccuracies. **7 – 8 marks**	Partial or inconsistent understanding of Android concepts. **4 – 6 marks**	Lacks understanding of core Android concepts used in the app. **0 – 3 marks**	10
App Architecture & Design	Clearly explains app architecture and responsibilities (UI → ViewModel → Repository). Justifies design decisions and separation of concerns. **9 – 10 marks**	Basic explanation of architecture with mostly correct responsibilities. **7 – 8 marks**	Limited or unclear explanation of architecture; weak separation of concerns. **3 – 6 marks**	No clear architecture or unable to explain relationships between components. **0 – 2 marks**	5