**Technical Planning**

**Technical Architecture Overview**

The Necuiltonolli mobile application is built using modern Android development principles, ensuring scalability, performance, and a smooth user experience. Below is the high-level overview of the app’s architecture, broken down into key components:

**Client-Side (Android App)**

The Android application is the core interface where users will interact with the app, including managing their media collection, tracking progress, and accessing different features. The client-side architecture will be built following MVVM (Model-View-ViewModel), which is a modern design pattern that enhances separation of concerns and allows for easy testing and maintenance.

a. View (UI Layer)

* Role: The View is responsible for displaying data to the user and capturing user input.
* Technology:
* Jetpack Compose or XML Layouts (depending on the design preference).
* Material Design components for modern, responsive UI elements.
* Navigation Component for managing app navigation.

Features:

* Displays lists of media categories (books, movies, shows, animes, and games).
* Provides options to filter by status (finished, unfinished) and category.
* Forms for adding, editing, and deleting media items.

b. ViewModel (Business Logic Layer)

* Role: The ViewModel is responsible for handling UI-related data and acting as a mediator between the UI (View) and data (Model). It processes and formats data before sending it to the UI.
* Technology:
* Jetpack ViewModel for managing UI-related data in a lifecycle-conscious way.
* LiveData for observing changes to data and updating the UI accordingly.

Features:

* Handles logic for managing the media collection (adding, modifying, deleting items).
* Filters and sorts data based on user input (status, category, etc.).
* Communicates with the Repository layer to fetch or save data.

c. Model (Data Layer)

* Role: The Model is responsible for defining the data structures and handling data persistence.

Technology:

* Room Database or SQLite for local data storage.
* Entities (data classes) to define the structure of media items (e.g., book title, category, status, etc.).
* DAO (Data Access Object) interfaces for querying and managing the database.
* Repositories to abstract the data operations, making it easy to switch between different data sources (e.g., Room, API).

**Local Storage and Offline Capabilities**

Since the app will initially focus on offline capabilities, it will store all media items locally on the device using the Room database.

a. Room Database Setup

* Entities: Data models representing media categories (books, movies, shows, etc.).
* DAO: SQL operations for inserting, updating, deleting, and querying media items.
* Database Class: The central point for interacting with Room for the media collection.

**Optional Future Features**

These are features that could be added in future updates to improve the app’s functionality.

a. Cloud Synchronization (Future Consideration)

* Role: Sync media collection data across multiple devices.

Technology:

* Firebase Firestore or Google Cloud for cloud storage and synchronization.
* Firebase Authentication for managing user accounts and profiles.
* Features: Users could log in to sync their collection data across multiple devices, ensuring that their data is always available wherever they go.

b. Analytics and Crash Reporting (Future Consideration)

* Role: Monitor app usage and detect issues.

Technology:

* Firebase Analytics for tracking user interactions and behavior.
* Firebase Crashlytics for tracking app crashes and errors to improve stability.

**Technical Stack**

* Programming Language: Kotlin (modern, concise, and preferred for Android development)
* UI Framework: Jetpack Compose (or XML Layouts if preferred)
* Database: Room Database (for local persistence)
* Architecture: MVVM (Model-View-ViewModel)
* UI Components: Jetpack Navigation, Material Design
* Offline Storage: Room Database (SQLite abstraction)

**Technical Challenges**

**Managing Complex Data Structures**

Challenge:

* Your app will store various types of media (books, movies, TV shows, animes, and games), each with different attributes. Managing these diverse datasets (titles, categories, statuses, dates, etc.) efficiently within a single database can be complex.

Potential Solutions:

* Design a unified schema where each media type (book, movie, anime, etc.) inherits common attributes (like title, status, etc.) but allows for specific attributes unique to each category.
* Use Room Entities and DAO (Data Access Objects) effectively to handle each media category’s operations.
* Keep the schema flexible enough to allow future extensions (such as adding new media categories).

**Handling Large Media Collections Efficiently**

Challenge:

* As users start adding a large number of items to their collection, performance may degrade, especially when querying and displaying long lists of media items.

Potential Solutions:

* Use pagination or lazy loading techniques to fetch and display data incrementally (for example, show 20 items at a time).
* Implement indexing in the database to speed up queries based on user filtering or sorting.
* Utilize Room Database's LiveData to observe data changes and efficiently update the UI without reloading all data.