[VAMA Coding Challenge Documentation]

Contents

Introduction:

- This Code documentation is to provide clear, comprehensive, and easily understandable information about the app feature's source code to developers, and reviewers involved in the evaluation process.

- The app's main function is to fetch the data from a network resource and fashionably render the content across different devices.

- The app is written in Kotlin and with Jetpack Comopse for the UI Layer.

Overview:

- The app displays the top 100 music albums across all genres using Apple’s RSS generator.

- The app displays the music albums with or without a network connection after the first time the data is loaded.

- The app is caching all albums data after the first time data is loaded.

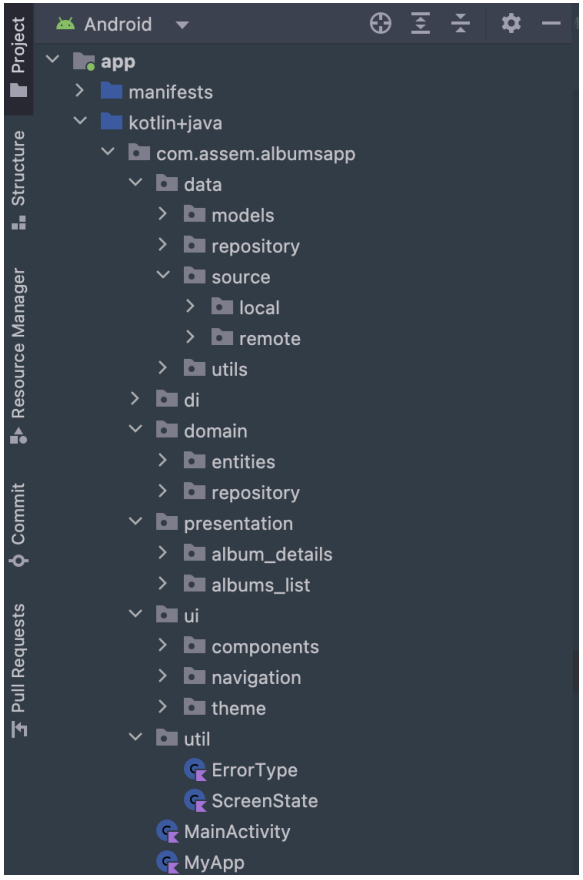
- Users can update the cache by refreshing the screen using swipe to refresh functionality

Work Plan:

Task	Estimated Time	Actual Effort
App Structure and Packages setup	1h	1H
Include required dependencies and third parties	1h	1H
Date Layer Implementation	4h	3H
Domain Layer Implementation	1h	1H
UI Layer Implementation	4h	4H
Error Handling	2h	2H
Cache Functionality Implementation	2h	2H
App UI Enhancements	2h	2H
Total Time	17H	16H

Code Documentation - App Architecture:

- ❖ This app is implemented using Kotlin, Clean Architecture, MVI Design pattern, and Dependency injection.
- ❖ Data access layer: contains code for interacting with the app's data sources, such as the local\remote source.
- ❖ Domain layer: This layer contains the business logic for the feature (Repository & entities).
- ❖ UI layer: This layer contains the code for the user interface and handles user interactions and View Models. We are using Jetpack Compose for implementing the UI.



Code Documentation - Data Layer:

- In the data layer, we are handling the data sources we have.
- We have two data sources one from the network endpoints and another one from the local database.
- We are using “Retrofit” third party to handle API calls.
- We are using “Realm” third party to implement the local database and caching functionality.
- The data Layer is mainly implemented using a Repository Pattern.
- We have mapper classes to map the Remote\Dao models into local entities to be used through the UI Layer.
- We have “ResourcesState” Class to define the states of getting the data process from the data source.

Code Documentation - Domain Layer:

- In the domain layer, We have the entities we use through the UI layer.
- We have the repository interface to define the communication between the domain and data layer.
- We don’t include use cases in such projects as the scope is not that big to use this approach. It may be considered overengineering in such a case.

Code Documentation - UI Layer:

- In the UI layer, We have mainly two screens.
- The first screen displays the list of the albums with album cover image, artist name, and album name.
- The second screen displays the details of the album and all its data returned from the data source.
- We implemented UI using Jetpack compose.
- We have “ScreenState” Class to define the states of the screen (Idle, Loading, Success, Error), so we can inflate the need for UI components based on the state.