Project Name: SpaceX Launch

Author: Shaffic Muhatsi

Date: 29/09/2023

**Technical Decisions Documentation**

Architecture Choice: The project follows the MVVM (Model-View-ViewModel) architectural pattern. MVVM was chosen because it separates concerns, making it easier to manage UI logic, data, and interactions. This pattern also aligns well with Jetpack Compose.

Repository Pattern: The project uses the Repository pattern to abstract data sources (API and local database). This separation allows for easier testing and switching between data sources in the future.

Jetpack Compose

Jetpack Compose: Jetpack Compose was chosen for the user interface due to its declarative nature, which simplifies UI development and encourages a more maintainable and flexible codebase.

Composable Functions: UI components are organized into reusable Composable functions to maintain code modularity and improve readability.

Data Fetching

Retrofit: Retrofit is used to make network requests to the SpaceX API. It offers a type-safe and efficient way to interact with RESTful APIs.

Room Database: Room is used for local storage and caching of launch data. This choice enables offline access and faster retrieval of previously viewed launches.

Testing

Unit Testing: Unit tests are written using JUnit and Mockito to test the business logic, including the ViewModel and Repository. Mock data is used to isolate the tests from external dependencies.

Integration Testing: Integration tests are written using Jetpack Compose's testing tools, including ComposeTestRule and TestComposeContent. These tests ensure that UI components are displayed correctly and interact as expected.

User Experience

Bookmarking Feature: A bookmarking feature is implemented, allowing users to mark launches for later viewing. Bookmarked launches are stored locally in the Room Database.

Error Handling: Appropriate error handling is implemented, including displaying error messages to the user when network requests fail or other errors occur.

Loading Indicators: Loading indicators are used to provide feedback to users while data is being fetched from the network.

Dependency Injection: dependency injection framework like koin for better management of dependencies, especially if the project grows in complexity.

Dependencies

Retrofit: Version X.X.X - Used for making network requests to the SpaceX API.

Room: Version X.X.X - Used for local database storage.

Jetpack Compose: Version X.X.X - Used for building the user interface.

JUnit: Version X.X - Used for unit testing.

Mockito: Version X.X - Used for mocking dependencies in unit tests.

Dependency Injection: Koin

Future Considerations

Pagination: Implement pagination for the launch list to handle a large number of launches efficiently.

UI/UX Improvements: Continuously improve the user interface and user experience based on user feedback and design guidelines.