Case Study: Sample Movie App

References- [Flutter challenge Kit](https://docs.google.com/document/d/1JU9k20GeKZiHegdHLP4Y_p4zJEh_hC6A430ONZ5Cfg4/edit)

App Overview:

The app is a movie browsing application built using Flutter, designed to allow users to discover and explore various movies. It includes features such as searching for movies, viewing details of individual movies, and bookmarking favorite movies for future reference.

Coding Decisions and Design Patterns:

1. MVC (Model-View-Controller):
   1. The app follows the MVC pattern to separate concerns and maintain code organization.
   2. Models represent data structures such as Movie, User, etc.
   3. Views are implemented using Flutter widgets, defining the UI components and layout.
   4. Controllers handle business logic, such as fetching movie data from APIs and managing user interactions.
2. Bloc Pattern for State Management:
   1. Bloc pattern is utilized for managing state in the app.
   2. Separate Bloc classes are created for different components of the app, such as MovieBloc, SearchBloc, etc.
   3. Blocs handle data transformation and state propagation, ensuring a clean separation of UI and business logic.
3. Repository Pattern for Data Management:
   1. Repository pattern is used to abstract data sources and provide a unified interface for accessing data.
   2. Separate repository classes are created for handling local data (e.g., cached movies) and remote data (e.g., fetching movies from APIs).
   3. This pattern allows for easier switching between different data sources and facilitates testing.
4. Dependency Injection:
   1. Dependency injection is used to manage dependencies between different components of the app.
   2. Libraries like get\_it or Flutter's built-in Provider are employed to inject dependencies into classes, improving code maintainability and testability.
5. Error Handling:
   1. Robust error handling mechanisms are implemented throughout the app.
   2. Error states are managed using Blocs, providing feedback to the user in case of network errors, API failures, or other exceptions.
   3. Flutter's ErrorWidget is used to display informative error messages and guide users on how to proceed.
6. UI/UX Design Principles:
   1. Attention is given to UI/UX details to ensure a visually appealing and user-friendly experience.
   2. Material Design guidelines are followed to maintain consistency and familiarity for users.
   3. Responsive layouts and intuitive navigation are implemented to accommodate various device sizes and user interactions.

By following these coding decisions and design patterns, the app achieves a balance between performance, maintainability, and user experience, providing a seamless movie browsing experience for users.