Introduction

This application is a sample project developed to showcase my technical expertise in Flutter development, focusing on UI composition, data persistence, and networking integration as outlined in the job requirements. The app has been built using clean architecture principles with modularized layers for **data**, **domain**, and **presentation**, ensuring maintainability and scalability.
The project demonstrates key skills such as UI design with Flutter widgets, data persistence using **Hive** and **SharedPreferences**, and API integrations for both REST and GraphQL using **Dio**. It also incorporates **Riverpod** for state management, **Logger** for efficient debugging, and modern libraries like **Freezed** and **JsonSerializable** for code generation and serialization. Designed to work seamlessly across Android, iOS, and Web platforms, this app highlights my ability to create production-ready solutions that meet high-quality standards.

Features Overview

This application consists of several features designed to demonstrate the core competencies required for the position. Each feature is implemented as a separate module, adhering to **clean architecture** principles and ensuring a modular and scalable codebase.

The **Home Feature** serves as the central hub of the app, managing the bottom navigation bar and providing a foundation for the other features. It ensures seamless transitions and interaction between the app's main sections.

The **Todo Feature** showcases data persistence using Hive. Users can add, view, and delete tasks with a simple and intuitive UI. This feature demonstrates my ability to implement local storage solutions effectively and highlights my proficiency in managing state with Riverpod.

The **Rest Feature** integrates with a REST API using Dio. It fetches user data from a test API and displays it in a list format. The implementation includes interceptors for logging and error handling, showcasing my expertise in building robust and maintainable networking modules.

The **GraphQL Feature** connects to a test GraphQL API and fetches user data dynamically. This feature demonstrates my familiarity with GraphQL integration and reinforces my ability to work with diverse networking paradigms.

The **Settings Feature** provides theme management functionality. Users can toggle between themes, with preferences stored using SharedPreferences. This feature highlights my attention to user experience and persistent settings management.
Each feature has been carefully designed to demonstrate a specific area of technical expertise, ensuring full alignment with the job requirements.

Key Technologies and Tools

This project leverages modern Flutter development tools and libraries to ensure high-quality and maintainable code. Below are the key technologies and tools used:

- **Flutter Widgets**: Utilized for building the app's UI, ensuring a responsive and consistent design across platforms.

- **Hive**: Used for local storage in the Todo feature, enabling efficient and lightweight data persistence.
- **SharedPreferences**: Implemented in the Settings feature for saving and managing user preferences.
- **Dio**: A powerful HTTP client used for REST and GraphQL API integration, with interceptors for logging and error handling.
- **GraphQL**: Demonstrated via a separate feature to showcase fetching and managing data from GraphQL APIs.
- **Riverpod**: Used for state management, ensuring a clean and reactive approach to managing app state.
- **Logger**: Integrated for enhanced debugging, providing detailed insights into API requests and app behavior.
- **Freezed**: Employed for data class generation, ensuring concise and boilerplate-free code.
- **JsonSerializable**: Used for seamless JSON serialization and deserialization, simplifying API data handling.
- **Clean Architecture**: The app follows this architecture to ensure separation of concerns, scalability, and ease of maintenance.

These technologies collectively demonstrate my expertise in building production-ready Flutter applications that are robust, scalable, and maintainable.