Furni Store Project Documentation

Team Members

عبدالله عبدالرقيب عبدالعزيز قائد
أشرف طلال شجاع الدين
وليد طلال العبسي
هشام علي حسين العمار

1. Project Overview

Project Title: Furni Store Platform

The **Furni Store** is a cross-platform application developed to provide a seamless experience for managing an e-commerce platform with two interconnected systems: a web-based PHP platform (admin panel) and a mobile app built using Flutter. The platform integrates via a shared database and API, ensuring consistent data across both systems. It includes key functionalities such as user management, product management, and category management, offering CRUD operations with authentication and role-based access.

2. Objective

To build a comprehensive, integrated system demonstrating the use of design patterns, SOLID principles, and clean coding practices in a cross-platform environment, while adhering to industry best practices for API and data management.

3. Technology Stack

3.1 Front-end:

- **Flutter** for mobile app development (Furni App).
- **PHP** for Website development (Furni Webiste & Dashboard).

3.2 Back-end:

- **PHP** for the web platform (Furni Store).
- **MySQL** for the shared database.

3.3 API:

- **RESTful API** developed in PHP, providing communication between the Flutter app and the PHP-based web platform.

4. SOLID Principles Implementation

The SOLID principles were applied in various areas of the project to ensure clean, maintainable, and scalable code.

4.1 Single Responsibility Principle (SRP)

- Each class in both the API and app has a single responsibility. For example, in the **Furni Store** API, the `UserManager` class handles user-related logic, while `ProductManager` handles product management.

4.2 Open/Closed Principle (OCP)

- Classes are open for extension but closed for modification. This was achieved by creating base repository classes for database operations, allowing future extensions without altering existing code.

4.3 Liskov Substitution Principle (LSP)

- In the Flutter app, abstract classes were defined for user-related features, and concrete implementations were substituted in without affecting the rest of the code.

4.4 Interface Segregation Principle (ISP)

- The API structure was designed to break down different responsibilities into separate interfaces, such as `UserService`, `CategoryService`, and `ProductService`.

4.5 Dependency Inversion Principle (DIP)

- Both the web and mobile platforms use dependency injection to decouple higher-level components from low-level details. For example, in the web platform, the `UserRepository` interacts with `DatabaseConnection`, and both follow the DIP.

5. Design Patterns Used

5.1 Repository Pattern

- Used in both the web and mobile applications to abstract data access logic. It provides a clean separation between business logic and data access. Each entity, such as `User`, `Product`, and `Category`, has its own repository for managing CRUD operations.

5.2 Facade Pattern

- In the **Furni Store** project, a `UserManager` class acts as a facade to the underlying user authentication, validation, and CRUD functionalities, simplifying the interface for other components.

5.3 Adapter Pattern

- Used in the Flutter app to adapt the mobile app's data into the format required by the API, ensuring smooth data integration between the mobile and web platforms.

6. Database Schema

The database is shared between the web-based platform and the mobile app, ensuring synchronization of data across both systems. The schema includes key entities such as:

- **Users:** `user_id`, `username`, `email`, `password`, `role_id`, `is_active`
- **Products:** `product_id`, `product_name`, `price`, `category_id`, `image_url`
- **Categories:** `category_id`, `category_name`

6.1 Entity-Relationship Diagram (ERD)

- The database follows a normalized schema to ensure efficiency and scalability.
- **Foreign Key Relationships:**
  - `product.category_id -> category.category_id`
  - `user.role_id -> role.role_id`

7. API Documentation

7.1 Authentication:

- JWT-based authentication.
- Users must authenticate using their credentials to receive a token, which is used for further API interactions.

7.2 Endpoints:

- **/api/v1/auth/login** - Authenticates a user and returns a token.
- **/api/v1/users** - Provides CRUD operations for users.
- **/api/v1/products** - Provides CRUD operations for products.
- **/api/v1/categories** - Provides CRUD operations for categories.

8. Flutter App Design and Features

8.1 User Management:

- **Login and Registration:** The app features a user login screen and registration screen.
- **JWT Token Management:** Securely stores the authentication token using `shared_preferences`.
- **User CRUD Operations:** Users can create, view, update, and delete their information using the API.

8.2 Product and Category Management:

- Integrated product and category screens that connect to the shared database through the API.

- **State Management:** The app uses the `Provider` package to manage state efficiently across multiple components.

8.3 Splash Screen with Animations:

- Integrated Lottie animations for the splash screen.

9. Testing and Validation

9.1 Unit Testing:

- The codebase includes unit tests for API endpoints and Flutter app components, covering user authentication, product management, and category management.

9.2 Integration Testing:

- Integration tests were performed to ensure smooth communication between the Flutter app, the API, and the database.

9.3 Code Coverage:

- Achieved a test coverage of 80% as per project requirements.

10. Challenges and Solutions

10.1 Cross-Platform Synchronization:

- Challenge: Maintaining consistent data across both platforms.
- Solution: Implemented a shared MySQL database and API to ensure real-time synchronization.

10.2 API Security:

- Challenge: Ensuring secure data transmission between platforms.
- Solution: Implemented JWT-based authentication for secure API access.

11. Conclusion

The **Furni Store** project successfully demonstrates the use of design patterns, SOLID principles, and clean code practices in a real-world application. It integrates two platforms, providing a seamless experience for users and administrators to manage products, categories, and users across the mobile app and web platforms.