FoodOrderingProject

Bülay Ecem Çakallı

This project has been developed using Full Stack Development. Full Stack Development consists of 3 main components:

1. Frontend
2. Database
3. Backend

The ability to effectively create these three layers is what we call Full Stack Development.
The purpose of this project is to fully understand these three main components and create a well-structured, comprehensible project.
The project development process is as follows:

Frontend Development Initial Process

Initially, I developed the presentation side of the frontend using the MVVM (Model-View-ViewModel) architecture. Starting with the View layer, I added screens and basic dialogs. Using Mock Data, I created a temporary ViewModel that managed this data in the background. This way, I established the general visual structure of the project and clarified the overall design in my mind.

Database Design

After the basic entities I would use became clear, I decided to create a database. I set up a simple database called FoodOrdering. After setting up the database, I realized that I hadn't created it in ANSI format, so I had to recreate it. The reason I wanted to use ANSI was for compatibility with other SQL platforms.

Backend Development Process

After the database setup, I transitioned to the Web Service layer. Among the three main components of the project, Web Service was the first part I completed fully. While creating this layer, I used approaches similar to Clean Architecture and Vertical Slice Architecture.

Architecture and Approaches - Backend

I didn't use pure Clean Architecture (Presentation, Application, Domain, and Infrastructure layers) in its entirety. However, I created a structure reminiscent of Clean Architecture consisting of 3 main layers: Core, Feature, and Infrastructure. Additionally, the Program.cs file serves as the application's entry point. The Database Controller is separate from other layers as it provides database connectivity. In the Feature folder, I created new folders for each entity and then separated their features within these folders. I tested my endpoints in Postman, and when the tests were successful, I transitioned back to the Frontend side.

Frontend Development Process

With the web service ready, I completed the transition from Mock Data usage to real database connection on the frontend side. For this change, I first added two files called ApiClient and ApiService to the data/remote section to manage API connections. This way, I established the connection using Retrofit. Then, I created the Repository pattern and implementations of these repositories to pull data into my project. Additionally, I made methods usable by using use cases in the app/di (dependency injection) section, but I couldn't achieve the desired DI model fully. It remained at the constructor level, and generally, in the parts where I used use cases, data was being recreated. This is actually an unwanted redundancy, but if the project continues, DI can be easily implemented with some code changes in the future. Also, DTO (Data Transfer Object) conversion operations are performed in mapper classes. Later, I created another page called ConfigHelper to add encryption, and as a result, I secured sensitive information using AES256.

Workflow

When you open the project, you encounter a startup page. On this page, you cannot enter the mainscrollable page, which is the main page, without establishing a connection with the database. Before the database configuration process, a default password is requested. When this password is entered correctly, you are directed to a page called configscreen. After this, if entered correctly, the mainscrollable page opens automatically. Also, there is a back button on the first entry (to startup page), but it's not available afterward. When you enter the mainscrollable page, the data had already been loaded with loadalldata in confighelper. Now, several filtering operations are performed here. The buttons below the search bar are for searching specific food or drinks in the search bar. When any food or drink is selected, a shopping cart button appears at

the bottom right. Above it, you can see the number of food, drinks, etc. you have taken. When this shopping cart button is pressed, you are redirected to CartPage. If you proceed further, you get PaymentMethodDialog. Payment is successful with creditmedcenter and multinet, but not with the others. These 'orders' are also sent to the DB. After having a working workflow, I went into more detail. I decided to add encryption. As a result, I opened another file called ConfigHelper.

Final Additions

I realized that what was requested from me was not to show that there was no database connection on the startup page and then go to config, but rather to open config at the beginning, set a password there, and then enter that password in the dialog asked to reopen that config. I corrected my code in this way and changed the initial startup page structure in MainActivity. This was my final addition. As a result, I completed these three layers of my project.

Future Process

In the future, the DI I mentioned can be implemented. Additionally, work can be done on the DevOps layer. Also, Unit Testing can be performed.

Device Compatibility - SDK Level Compatibility

Compatibility with different Android SDK levels was considered during project development. The minimum SDK level was set as API Level 21 (Android 5.0 Lollipop) to reach a wide user base. The target SDK level was kept current as API Level 34 (Android 14) to ensure compliance with Google Play Store requirements. This way, it can run smoothly on all Android devices with Android 5.0 and above. Additionally, avoiding the use of deprecated APIs is planned to maintain compatibility with future Android versions.