**SC2006 Software Engineering**

****

**Lab#3 Deliverables**

| Group Members (Group 45) | |
| --- | --- |
| Name | Matriculation No. |
| Kua Li Min | U2322513H |
| Hudzaifah Bin Muhammad Taufiq | U2320600F |
| Cheng Hui Wen | U2322123G |
| Ng Hoe Ping | U2321991F |
| Seow Jia Xian Jackson | U2322995F |
| Chua Ze Ming | U2321797B |
| Zhang Yuxuan | U2321475L |

Table of Contents

**1. Complete Use Case Model** [**2**](#_tyjcwt)

**2. Design Model** [**8**](#_4d34og8)

**A. Class Diagram** [12](#_ewyopsncq4ui)

**B. Sequence Diagrams** [12](#_ewyopsncq4ui)

I. Account system [12](#_ewyopsncq4ui)

II. Browse system [12](#_ewyopsncq4ui)

III. Listing system [12](#_ewyopsncq4ui)

IV. Offer system [12](#_ewyopsncq4ui)

V. Review system [12](#_ewyopsncq4ui)

**C. Dialog Map Diagram** [12](#_ewyopsncq4ui)

**3. System Architecture** [**9**](#_2s8eyo1)

**4. Application Skeleton** [**10**](#_g6rxf44ycqox)

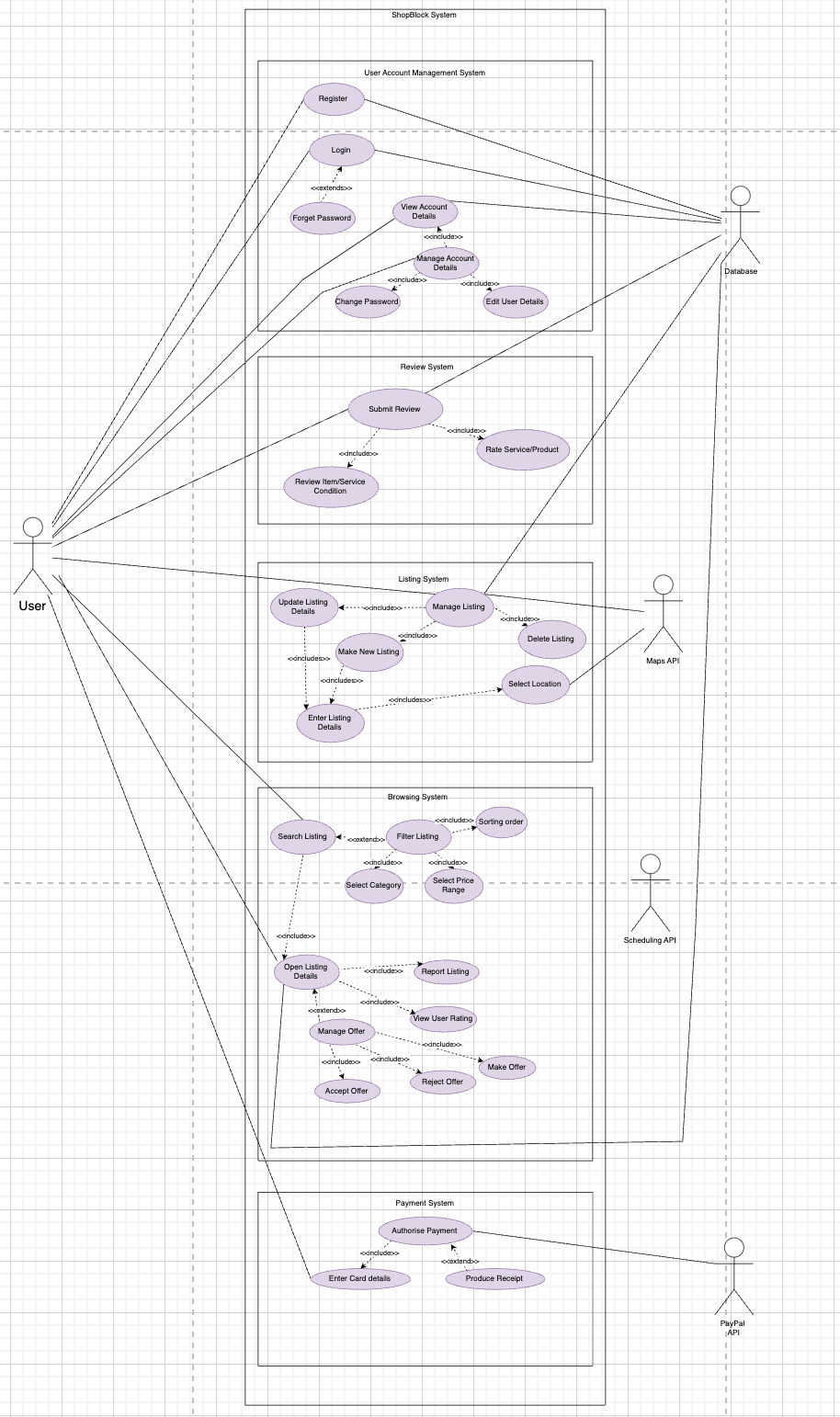
**A. Frontend** [12](#_ewyopsncq4ui)

**B. Backend** [12](#_ewyopsncq4ui)

**5. Appendix** [**12**](#_lnxbz9)

## 

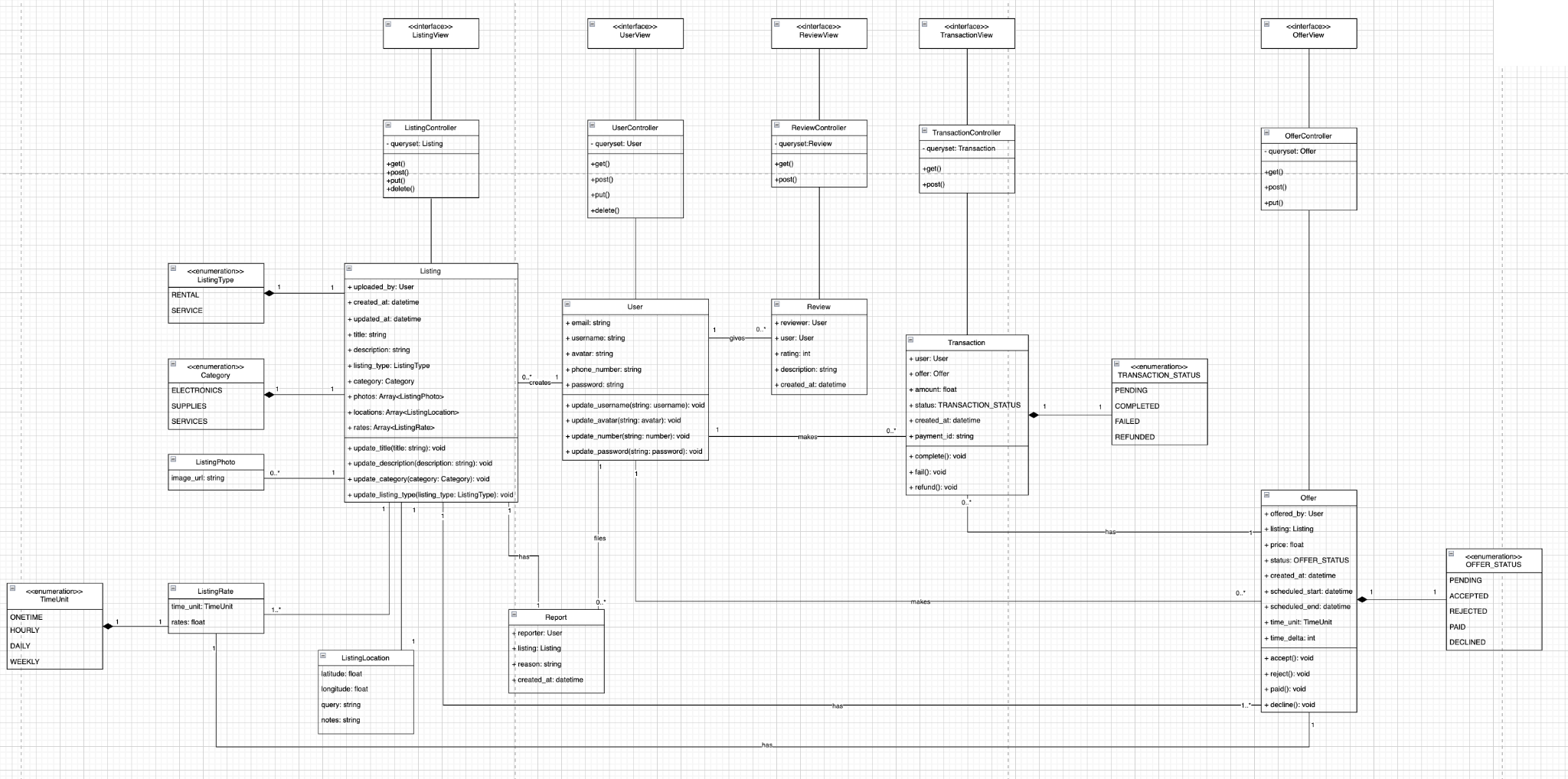
## Complete Use Case Model



If the image is unclear, please refer to the raw file included with the PDF on the associated GitHub repository.

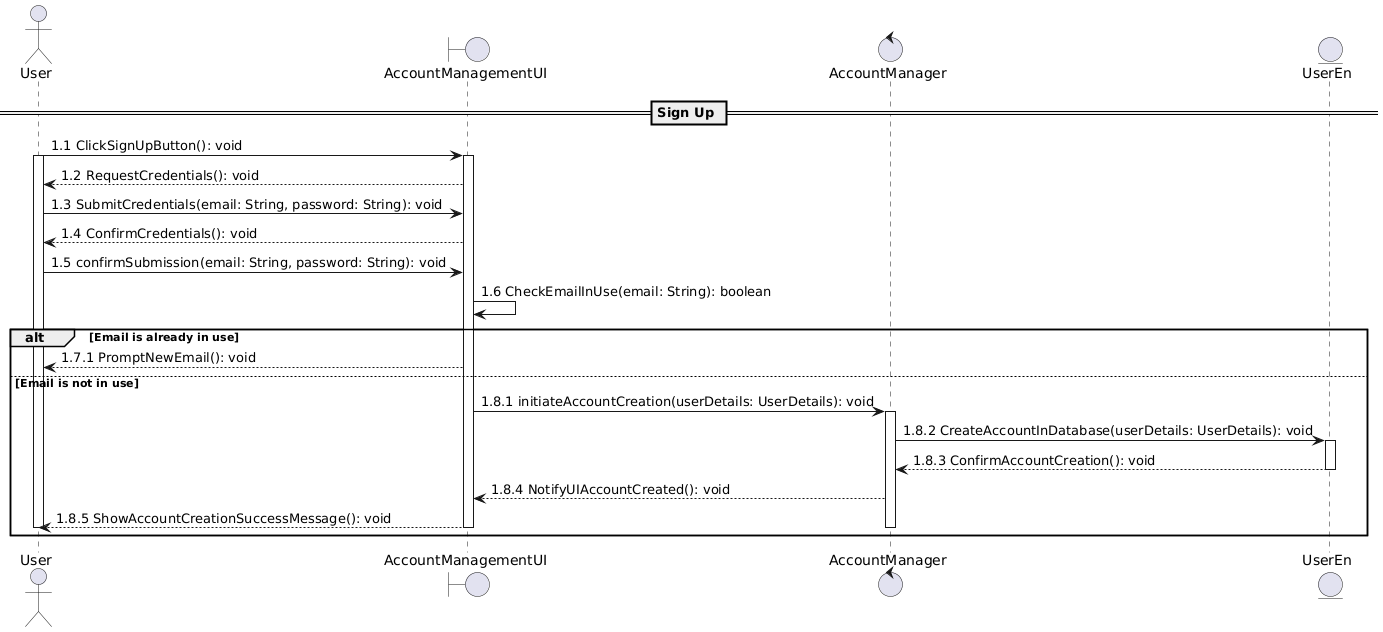
## Design Model

1. Class Diagram

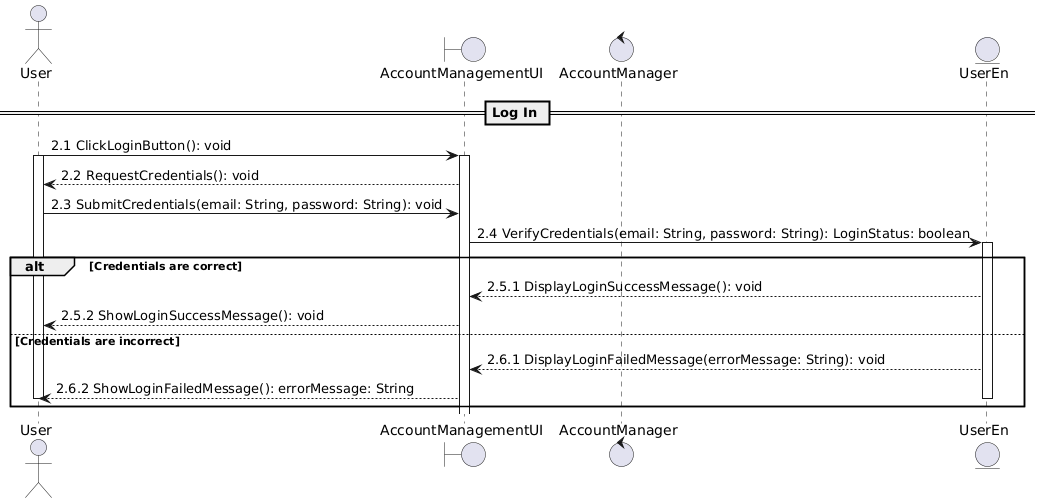


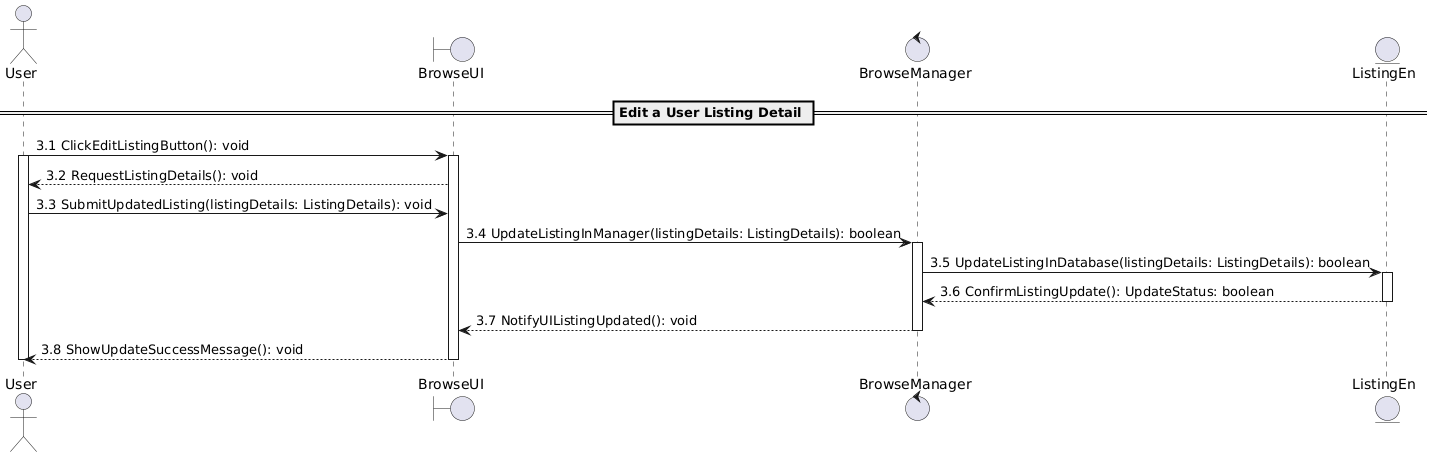
If the image is unclear, please refer to the raw file included with the PDF on the associated GitHub repository.

1. Sequence Diagram

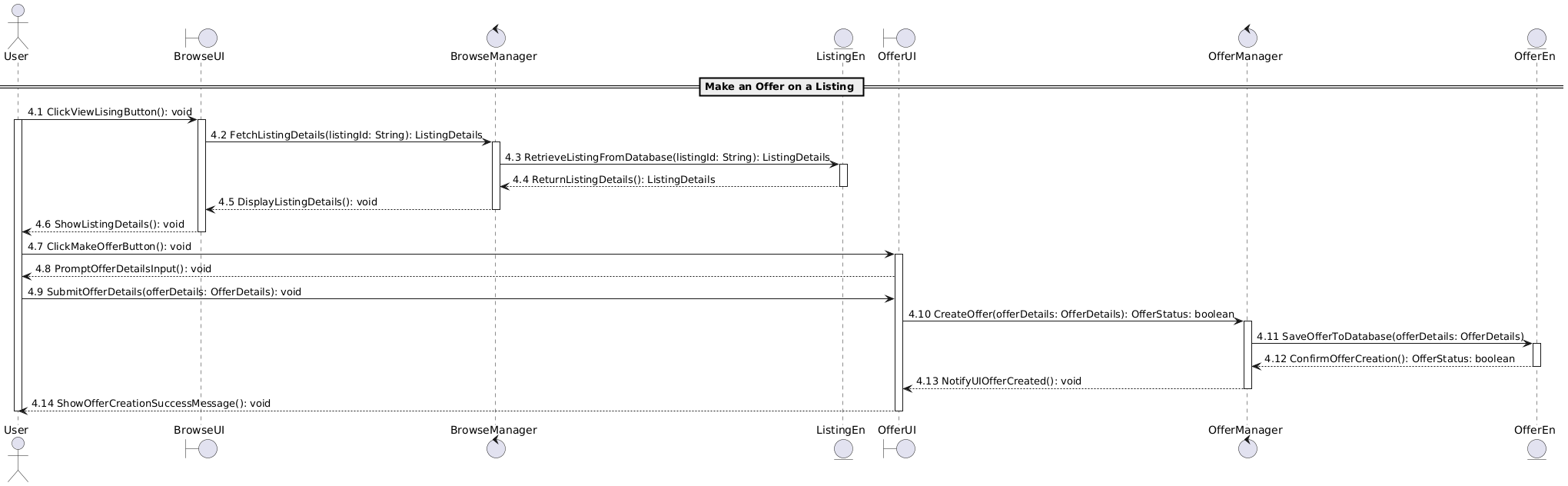
B.1 Sign Up

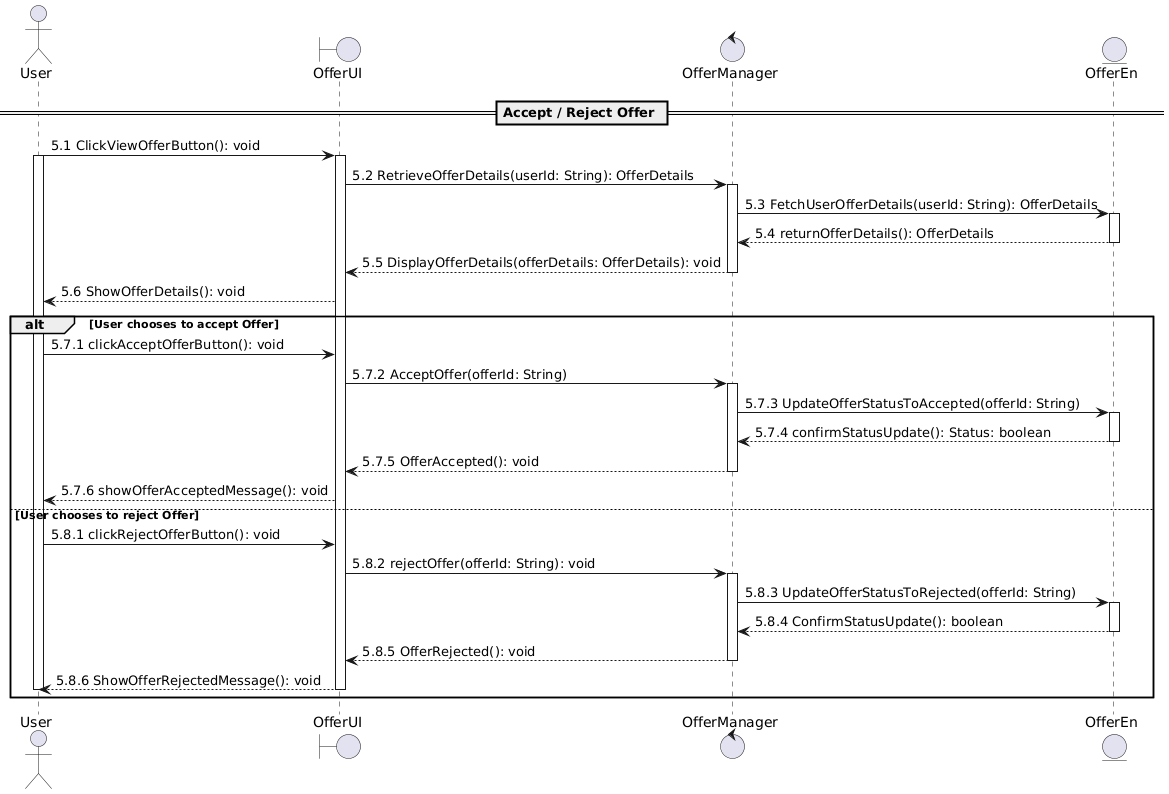
B.2 Login In

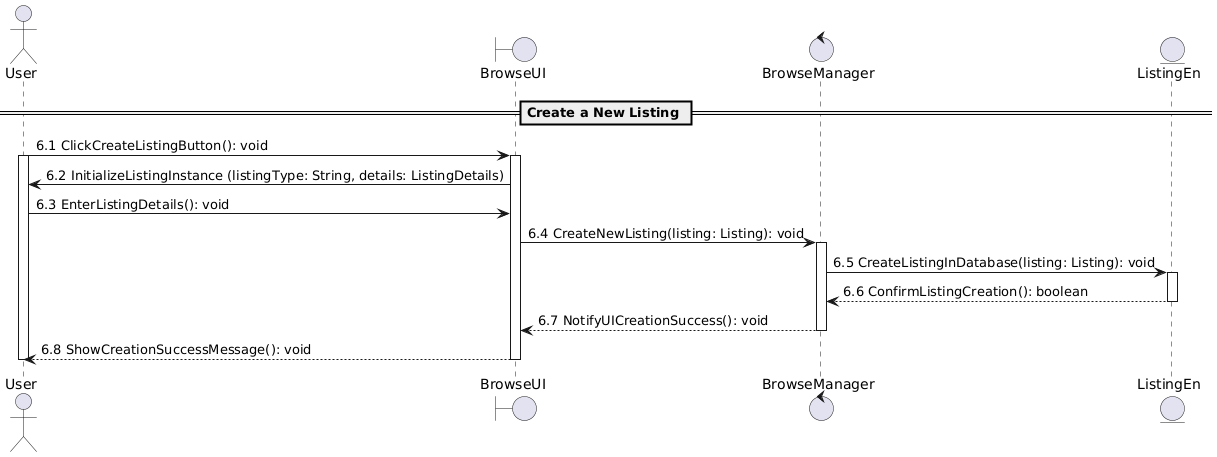


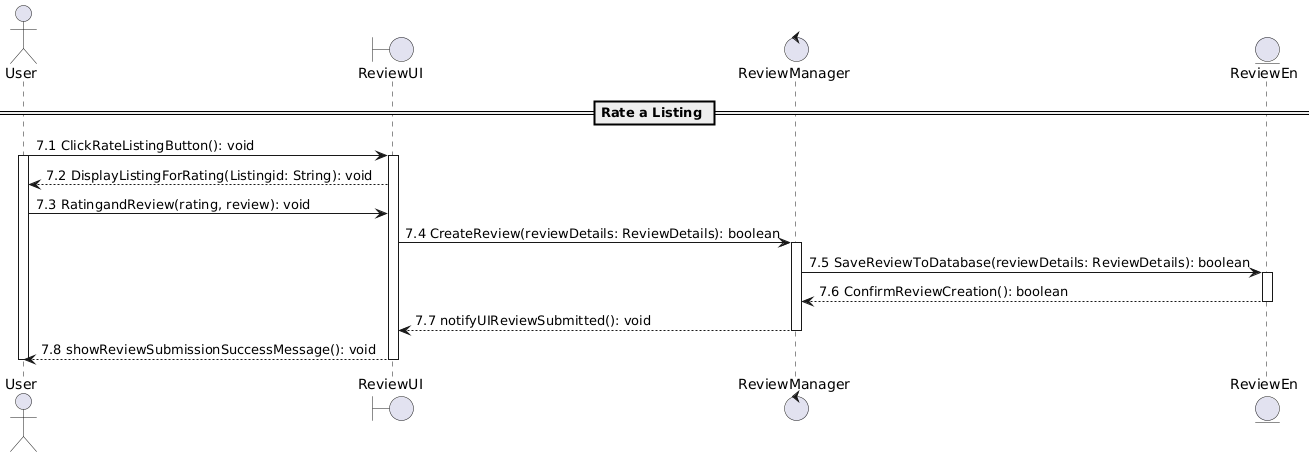
B.3 Edit Listing details

B.4 Make Offer



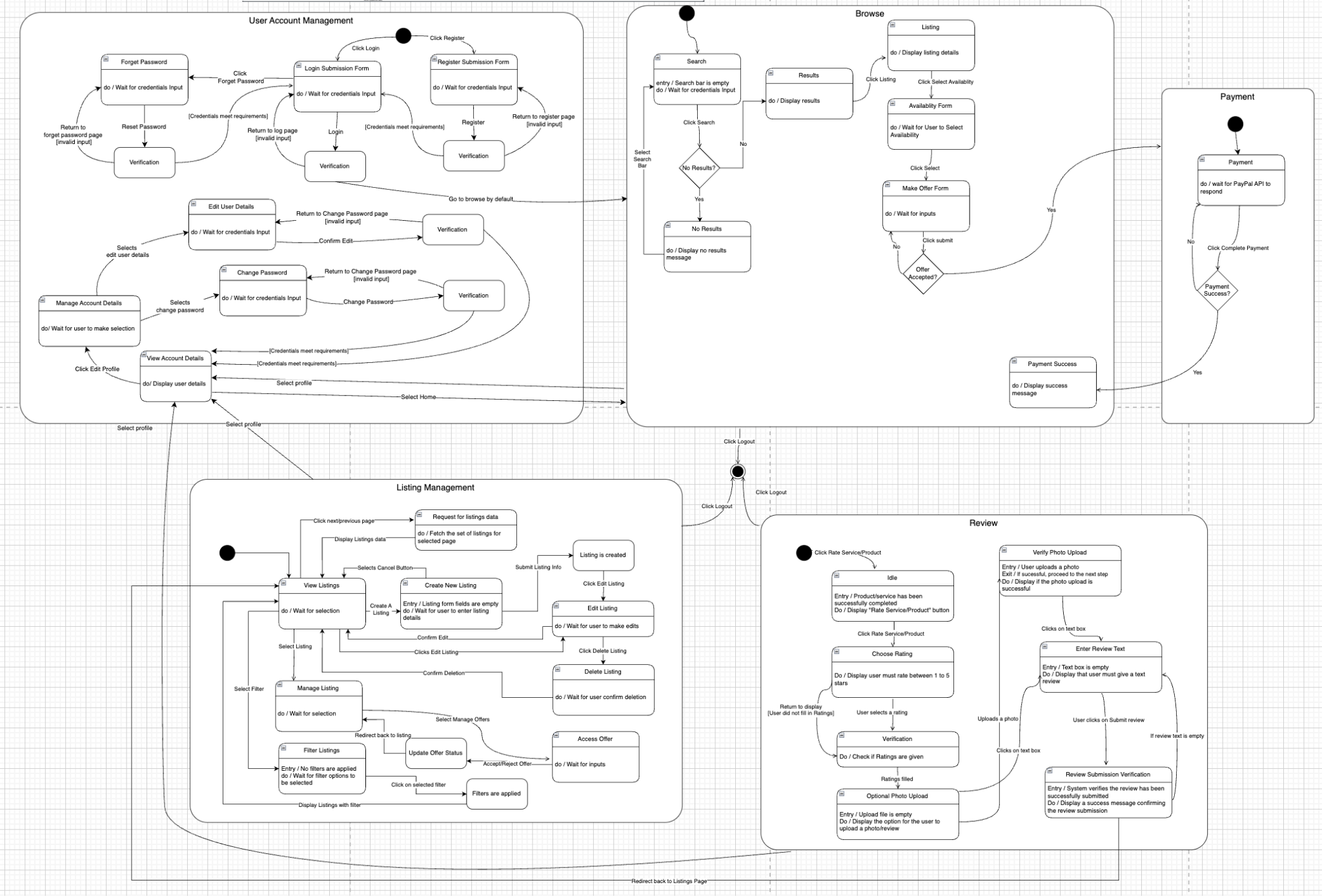
B.5 Accept/Reject Offer

B.6 Create listing  
   
  
B.7 Review a listing



If the image is unclear, please refer to the raw file included with the PDF on the associated GitHub repository.

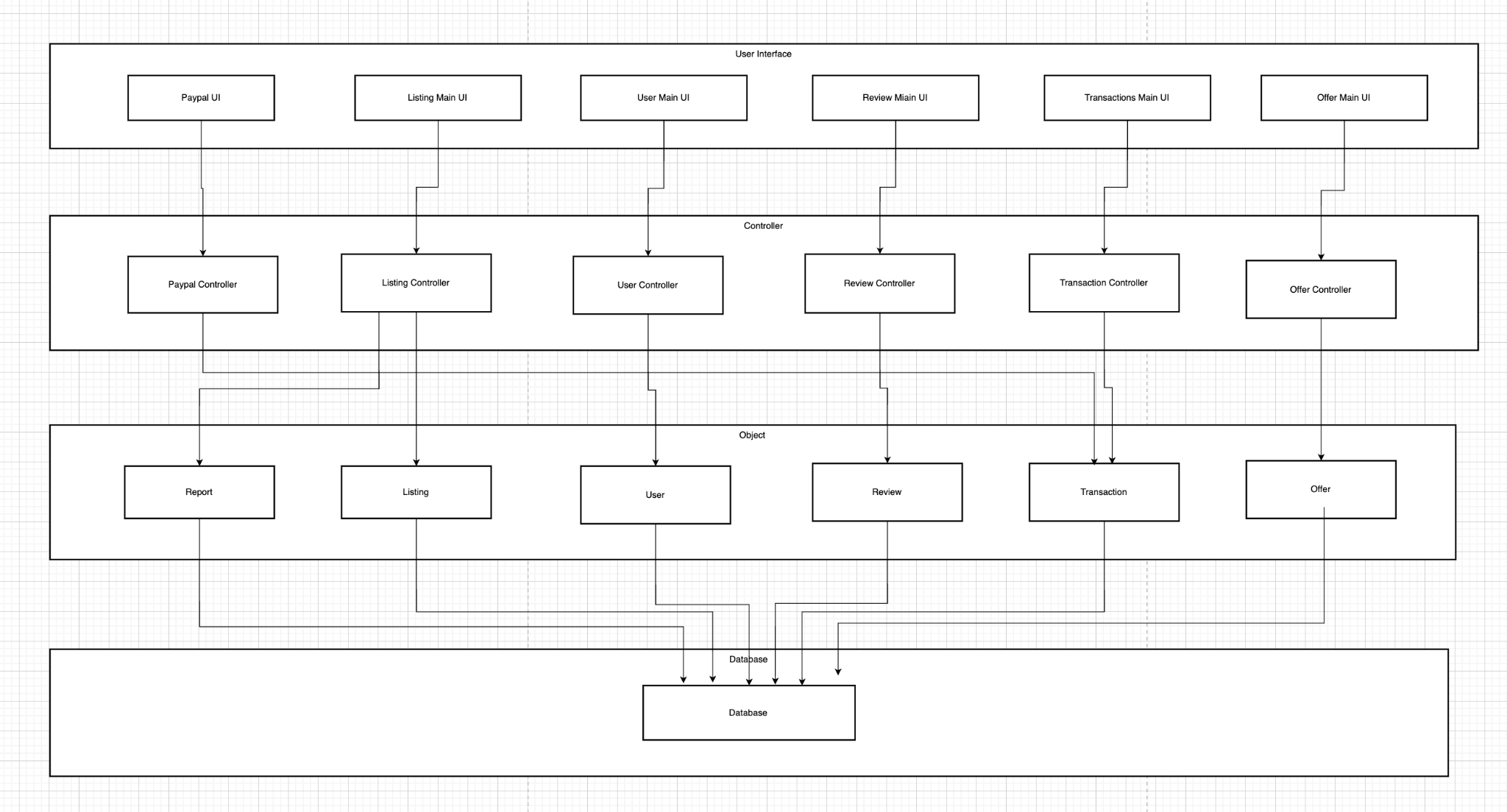
1. Dialogue Map Diagram



If the image is unclear, please refer to the raw file included with the PDF on the associated GitHub repository.

## System Architecture

If the image is unclear, please refer to the raw file included with the PDF on the associated GitHub repository.



1. **User Interface Layer**

This layer serves as the interaction point between users and the ShopBlock platform. It provides well-structured interfaces tailored to the various tasks users need to perform. Each UI ensures an intuitive experience while coordinating with controllers to manage the business logic effectively.

**PayPalUI**

Provides users with an easy way to complete transactions using PayPal services.

It interacts with the PayPal Controller to initiate payments, confirm them, or issue refunds.

**ListingMainUI**

Focuses on managing product and service listings.

It works closely with the Listing Controller to add, edit, delete, and explore listings or related offers.

**UserMainUI**

Handles user profile management, including login, registration, and password updates.

It communicates with the User Controller to keep user information up-to-date and handle authentication seamlessly.

**ReviewMainUI**

Enables users to submit feedback and ratings for sellers or products.

It relies on the Review Controller to manage review submission and modification.

**TransactionsMainUI**

Tracks and manages user transactions.

This interface works with the Transaction Controller to monitor payment statuses, refunds, and transaction histories.

**OfferMainUI**

Lets users explore, create, and manage offers associated with listings.

It collaborates with the Offer Controller to accept, decline, or propose new offers.

1. **Controller Layer**

This layer is responsible for handling the ShopBlocks logic. Each controller class provides specific services to the User Interface Layer, retrieving necessary data from the Object Layer to ensure smooth operation.

**PayPal Controller**

Manages all PayPal-related actions, including payment initiation, confirmation, and refunds.

**Listing Controller**

Handles listing operations such as adding, updating, and removing products or services.

**User Controller**

Manages user operations, including login, registration, and profile updates, as well as retrieving data from external APIs when needed.

**Review Controller**

Oversees review-related functions, including submission, editing, and retrieval of user feedback.

**Transaction Controller**

Handles the transaction process, tracking payment statuses, managing refunds, and updating transaction records.

**Offer Controller**

Manages offer operations, enabling users to create, accept, or decline offers and facilitating seamless negotiations on the platform.

1. **Object Layer**

This layer contains the core entity classes used by the App Logic Layer to implement business operations. The entities are stored and managed in the database within the Persistent Data Layer. Below is an overview of the key entities in this layer:

**Report**

Stores reports related to listings, offers, or user behavior.

**Listing**

Manages product or service details, including title, description, type, and availability.

**User**

Contains all relevant user information such as name, email, password, profile picture, and role within the platform.

**Review**

Stores user feedback on listings, including ratings, text reviews, and flagged status or reason (if applicable).

**Transaction**

Manages financial transactions, including the transaction amount, status, and associated user or listing.

**Offer**

Contains details of offers made on listings, including price, status (accepted or rejected), and creation timestamps.

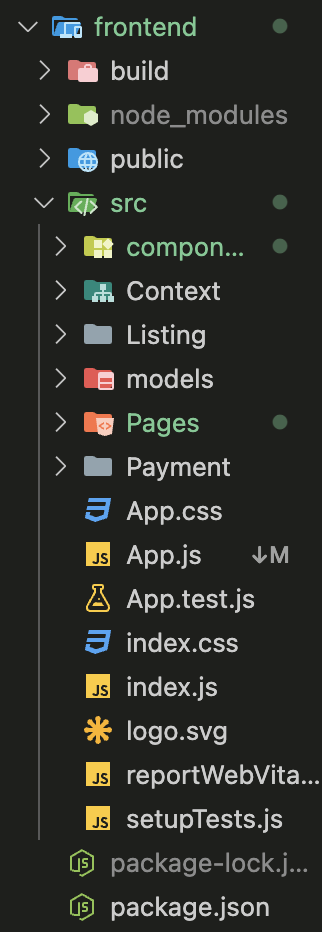
1. **Database Layer**

This layer is used to house the database responsible for storing and managing all entities from the Object Layer. It ensures the integrity, persistence, and availability of data to the system, facilitating smooth interactions between the Object Layer and the other components in the application.

## Application Skeleton

If the image is unclear, please refer to the raw file included with the PDF on the associated GitHub repository.

1. **Frontend**



The frontend of the ShopBlock platform is built with React.js, a framework for creating user interfaces. It’s organized into folders and files to keep the code neat, making it easier to maintain and collaborate on. Key folders include Context (for managing global state like user authentication), Models (for defining data structures like Users or Listings), and Helpers/Utilities (for reusable functions). The Public folder stores static files like logos, while the Build folder contains the final production-ready version of the app. Dependencies are managed by npm or yarn in node\_modules.

**Components**

This folder holds reusable UI elements, like widgets.

Examples: A navbar that shows on multiple pages or buttons that can be used throughout the app.

**Pages**

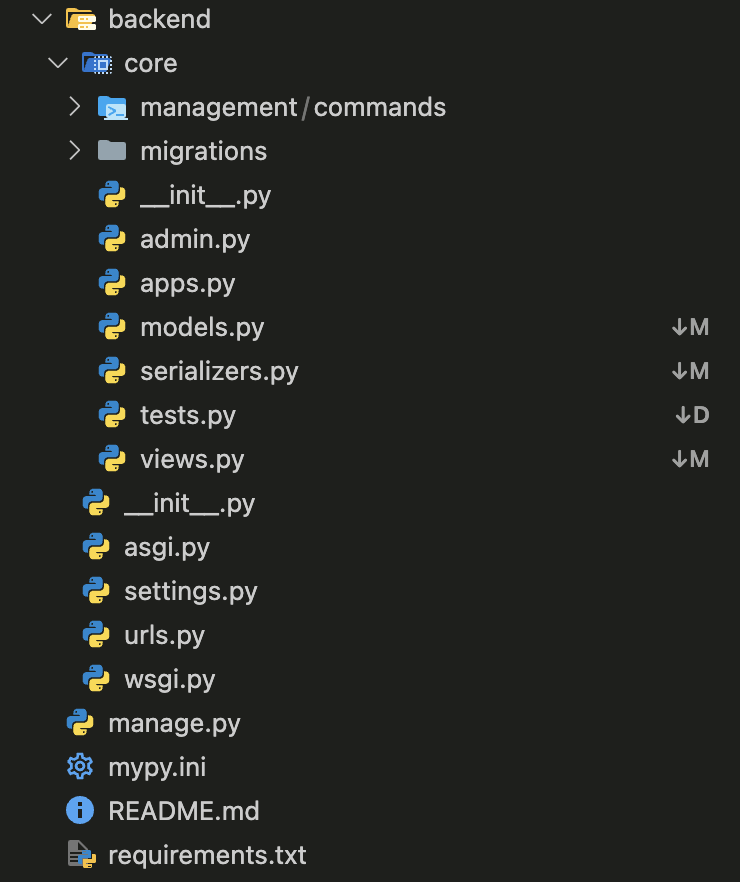
This folder contains the individual screens users interact with.

Examples: Signup page, Listing page, Payment page, and others.

**App.tsx / App.js**

This is the starting point of the application, where everything comes together. It pulls in all the components and pages to create a smooth, unified experience for the user.

1. **Backend**

**management / commands folder**

Defines management commands for custom tasks in Django. Typically used for operations like database seeding, cleaning up data, or batch processing outside the regular app flow.

**models.py**

Defines the database structure by mapping models to tables, with attributes representing columns and methods for managing records.

**serializers.py**

Handles data conversion between complex objects and JSON, validating input for views and preparing data for APIs.

**views.py**

Manages request handling by processing inputs, interacting with models/serializers, and sending appropriate responses to the client.

### 