**INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

# Software Design Document

# *Real Estate Platform*

Course Title : Software Engineering Lab

Course Code : CSE-3638

**Submitted by**

- Nahian Subah Ishma_C223286
- Rehnuma Tasneem_C223288
- Saima Kawsar_C223297
- Sakaratul Ara Tasmia_C223298
- Tahsin Islam Nafisa_C223311

Semester : 6th Section : 6CF

**Submitted to**

Sultana Tasnim Jahan
Assistant Lecturer
Dept. of CSE , IIUC

## *Table of Contents*

Introduction .......... 2
Architectural Pattern .......... 3
Activity Diagram .......... 4
Class Diagram .......... 8
Sequence Diagram .......... 13
State-Machine Diagram .......... 17
Conclusion .......... 20

## Introduction

This document explains the design of our Real Estate Platform, which is a web-based system made to help buyers and agents manage property activities more easily. The platform allows buyers to search for properties, save their favorite ones, and make offers. At the same time, agents can add new property listings, view offers from buyers, and manage their property records.

The main goal of this document is to show how the system is planned and how each part will work together. We are using the MVC architecture, which means the system is divided into three parts: the part that handles data, the part that shows the interface to the user, and the part that manages the logic and flow between them. This helps keep the system clean, organized, and easy to manage.

To explain our design more clearly, we have included several diagrams. The activity diagram shows how users move through different steps like logging in, searching, and making offers. The class diagram shows the important parts of the system, such as users, properties, and payments, and how they are connected. The sequence diagram shows how different parts of the system communicate during specific tasks. The state-machine diagram shows how the condition of objects like property or offer changes over time.

This document will help everyone involved in the project, including developers, testers, and team members, understand how the system will work. It also makes sure that the design is followed properly during the development, making the system easier to build and maintain.

## Architectural Pattern

The architectural pattern illustrated represents a 3-tier architecture model used in the real-estate system, based on the MERN stack—MongoDB, Express.js, React.js, and Node.js. The Presentation Layer is handled by React.js, which interacts directly with the users and collects input from the frontend. The Application Layer, or business logic layer, is managed by Express.js (within the Node.js environment) and is responsible for handling API requests, processing the core logic, and connecting to the database. Finally, the Data Access Layer uses MongoDB to store all persistent data, such as user information, property listings, reviews, and transactions. These data interactions are typically managed using models created with tools like Mongoose, allowing for efficient data storage and retrieval.

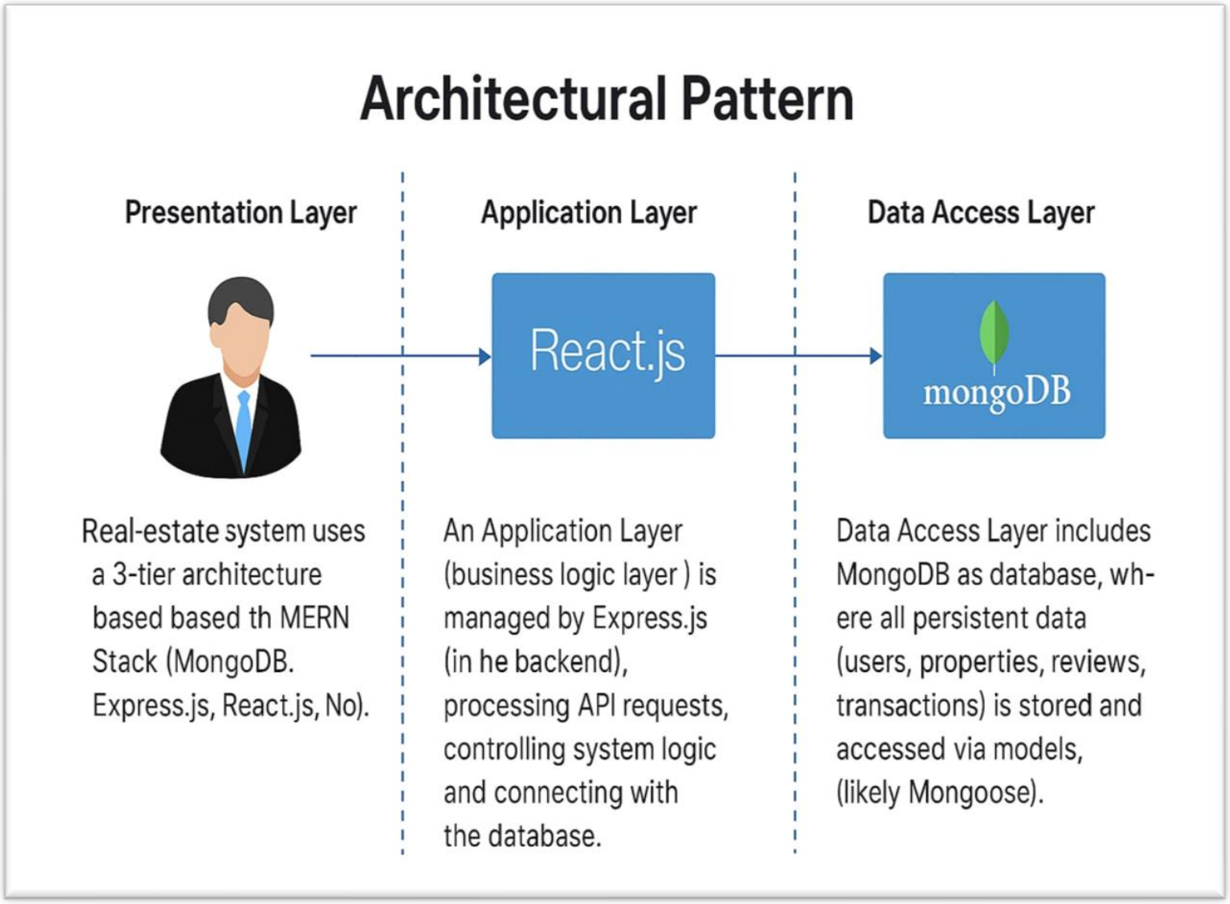
Architectural Pattern
Presentation Layer
Application Layer
Data Access Layer
React.js
mongoDB
Real-estate system uses a 3-tier architecture based based th MERN Stack (MongoDB. Express.js, React.js, No).
An Application Layer (business logic layer ) is managed by Express.js (in he backend), processing API requests, controlling system logic and connecting with the database.
Data Access Layer includes MongoDB as database, wh-ere all persistent data (users, properties, reviews, transactions) is stored and accessed via models, (likely Mongoose).

Fig:01-Architechtural Pattern

## Activity Diagram

An Activity Diagram is a type of UML (Unified Modeling Language) diagram used to show the flow of control or data from one activity (task) to another.It’s like a flowchart, but more structured and formal , especially good for showing how a system behaves step-by-step, including decisions, loops, parallel processes, and start/end points.

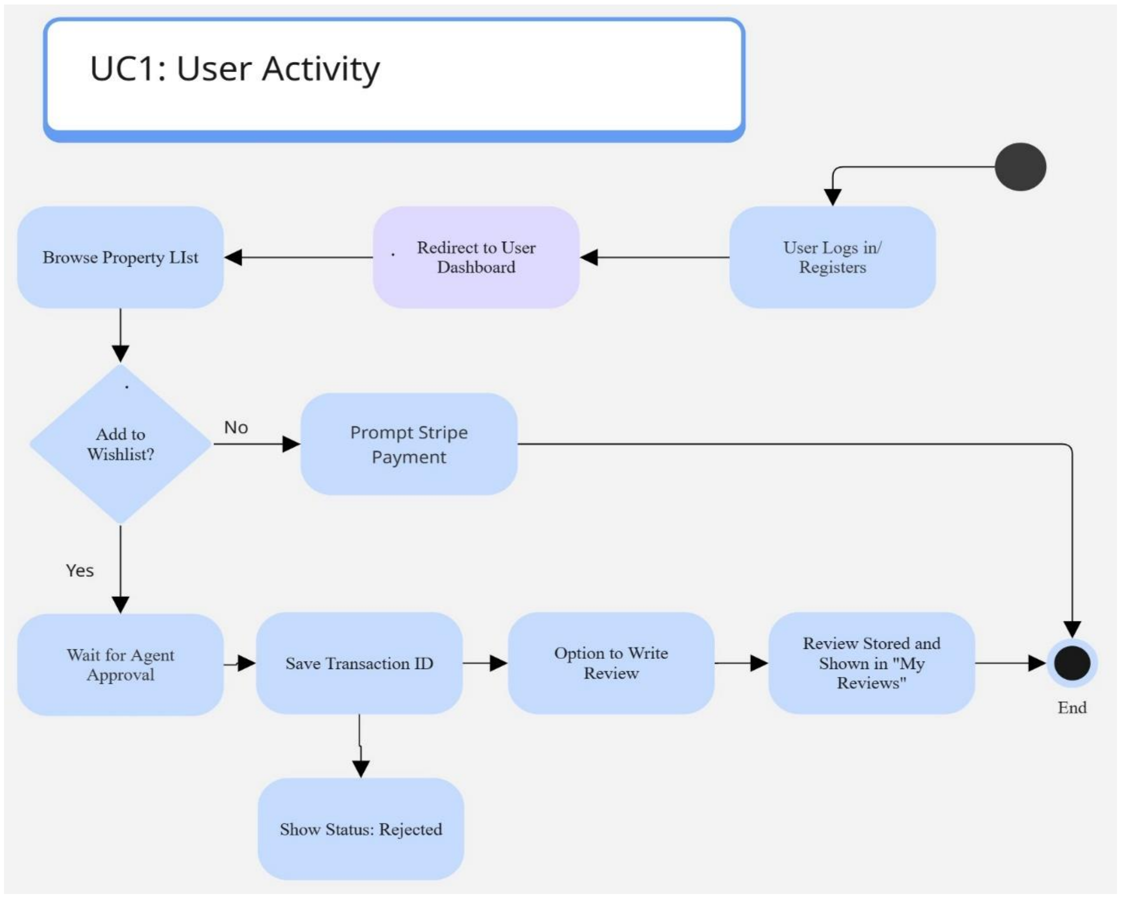

Fig:02-User Activity

This diagram shows how a user interacts with the property platform. After logging in or registering, the user browses property listings. If they add a property to the wishlist, the system waits for agent approval, saves the transaction ID, and allows the user to write a review. If not, the user is prompted to make a Stripe payment before writing a review. All reviews are stored in the "My Reviews" section.

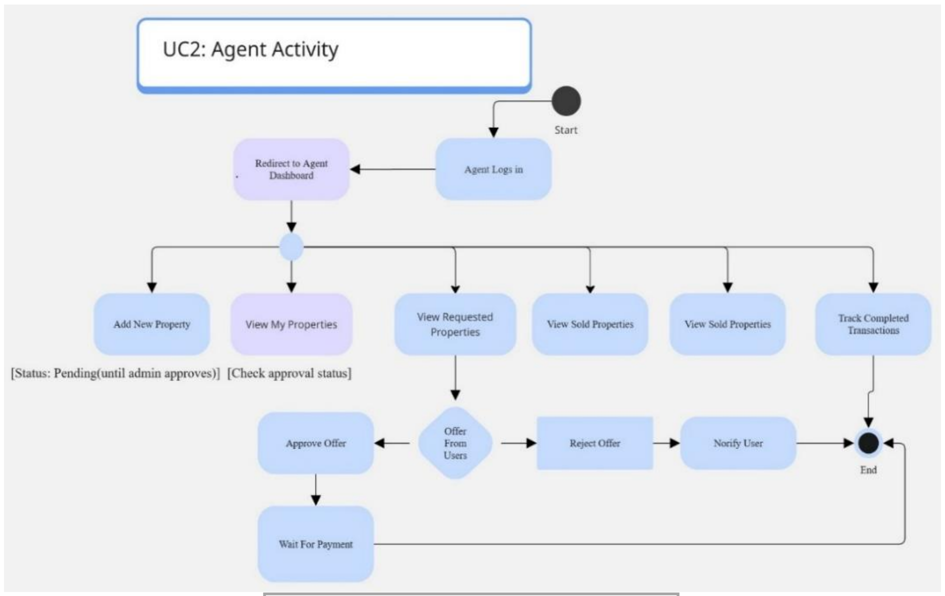

Fig:03-Agent Activity

This activity diagram outlines the agent's workflow on the platform. After logging in, the agent is redirected to their dashboard. They can add new properties (pending admin approval), view their properties, and manage user offers. For each user offer, the agent can either approve it (then wait for payment) or reject it and notify the user.

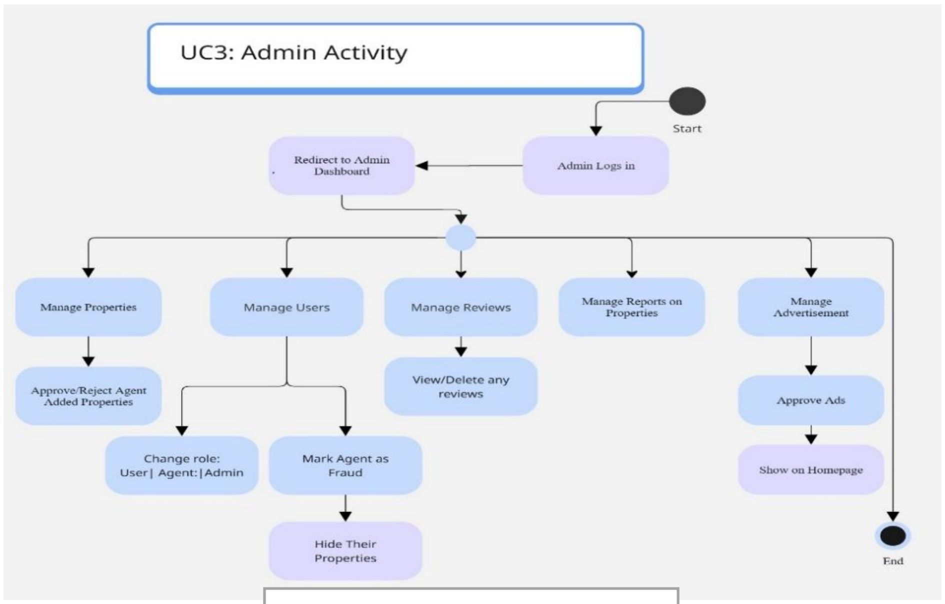

Fig:04-Admin Activity

The admin logs in and is redirected to the dashboard. From there, they can manage properties by approving or rejecting agent submissions, handle users by changing roles or marking agents as fraud, and manage reviews by viewing or deleting them. The admin can also review property reports and approve ads to be shown on the homepage. The process ends after completing any of these tasks.

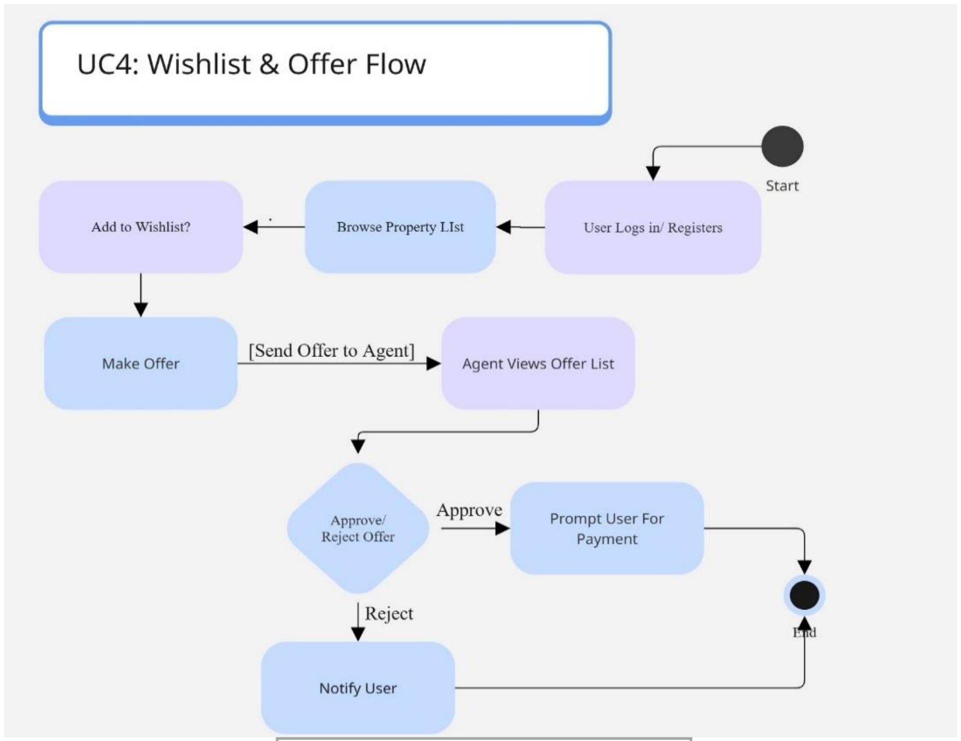

Fig:05-Wishlist & Offer Flow

User logs in, browses properties, and can add to wishlist or make an offer. The agent reviews the offer and either approves or rejects it. If approved, the user is prompted to pay; if rejected, the user is notified.

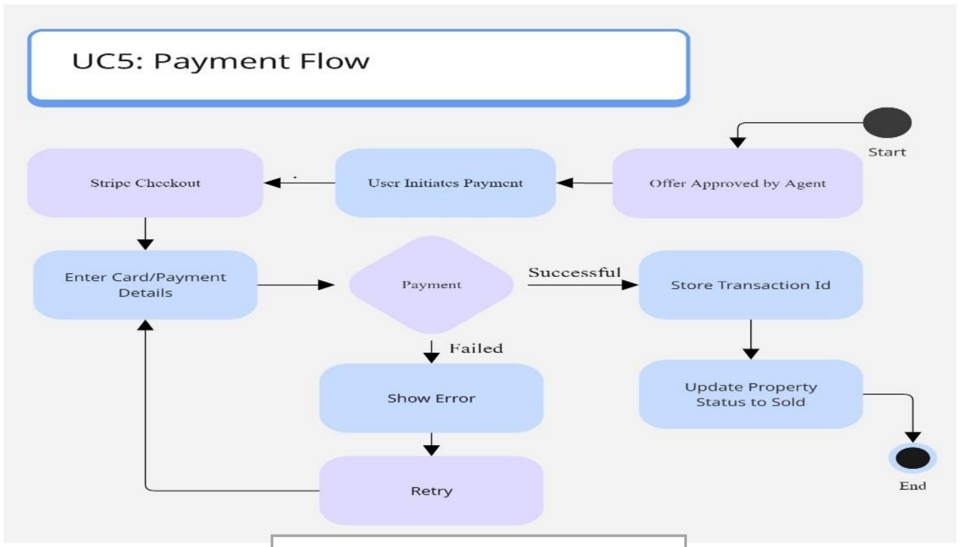

Fig:06-Payment Flow

After an agent approves an offer, the user initiates payment via Stripe Checkout and enters their card details. If the payment is successful, the system stores the transaction ID and marks the property as sold. If it fails, an error is shown and the user can retry the payment.

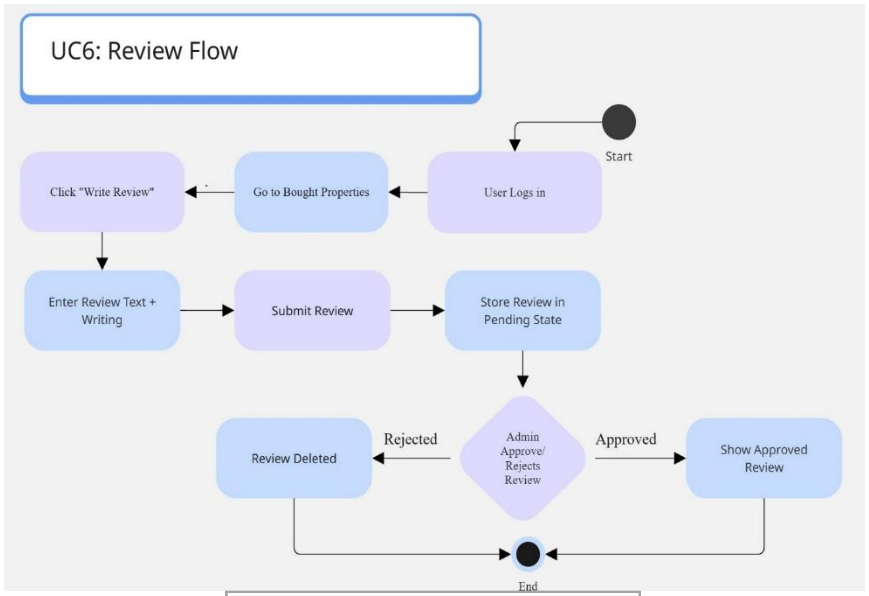

Fig:07-Review Flow

This flow shows how a user writes a review. First, the user logs in and goes to the “Bought Properties” section. Then they click on “Write Review” and enter their review text. After writing, they submit the review. The review is saved in a pending state. An admin checks the review and decides whether to approve or reject it. If approved, the review is shown to everyone. If rejected, the review is deleted. This is how the review process works in the system.

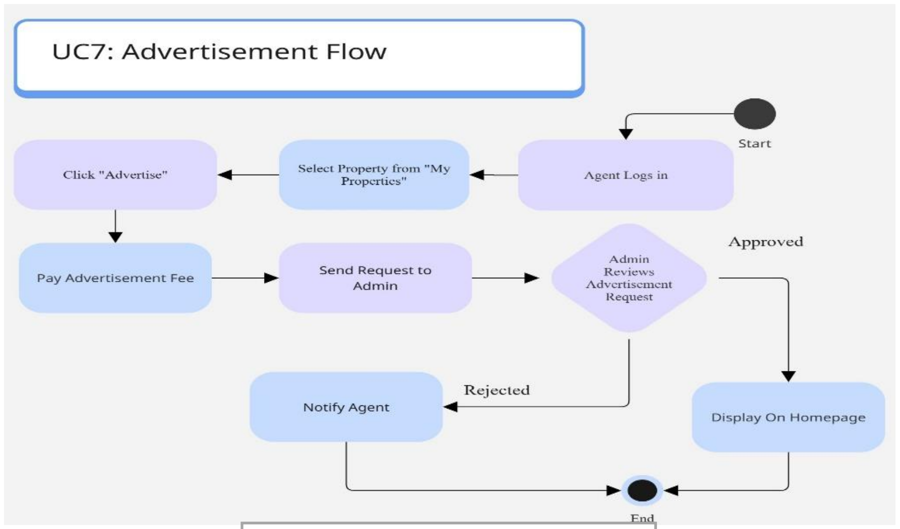

Fig:08-Advertisement Flow

This flow shows how an agent advertises a property. First, the agent logs in and selects a property from "My Properties". Then they click on "Advertise" and pay the advertisement fee. After that, a request is sent to the admin. The admin reviews the request. If approved, the property is shown on the homepage. If rejected, the agent gets a notification. This is how property advertisement works in the system.

## Class Diagram

It describes the organization of a system by showing its classes, their attributes, and operations. It also highlights how these classes are related to each other. This gives a clear view of the system's structure and how different objects are interconnected, helping to understand the system from a structural perspective.

### Boundary Classes:

This Boundary Class Diagram represents the key user interface components of a real estate platform. The LoginActivity handles user login, registration, and Google login, directing users to their role-based dashboards. The DashboardActivity displays the user's dashboard with an option to log out. The PaymentActivity manages payment processing by starting and confirming payment sessions. Through the AdvertisementActivity, users can create property advertisements and view them on the home page. The PropertyDetailsActivity allows users to see property details, add properties to their wishlist, make offers, and submit reviews. The WishlistActivity shows saved properties and enables users to remove items. Finally, the ReviewsActivity displays property reviews and allows users to delete their own reviews. Together, these boundary classes define how users interact with the platform's various features.

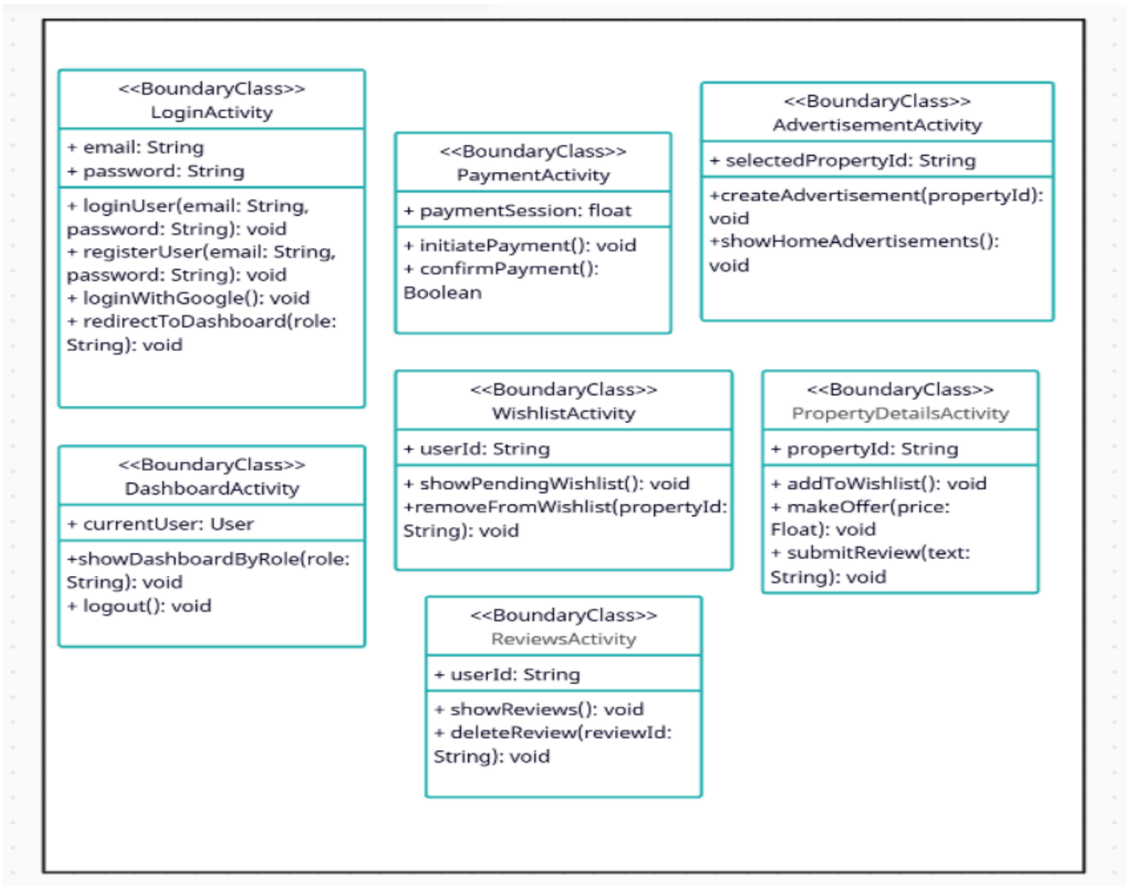
<<BoundaryClass>>
LoginActivity
+ email: String
+ password: String
+ loginUser(email: String, password: String): void
+ registerUser(email: String, password: String): void
+ loginWithGoogle(): void
+ redirectToDashboard(role: String): void
<<BoundaryClass>>
PaymentActivity
+ paymentSession: float
+ initiatePayment(): void
+ confirmPayment(): Boolean
<<BoundaryClass>>
AdvertisementActivity
+ selectedPropertyId: String
+createAdvertisement(propertyId): void
+showHomeAdvertisements(): void
<<BoundaryClass>>
DashboardActivity
+ currentUser: User
+showDashboardByRole(role: String): void
+ logout(): void
<<BoundaryClass>>
WishlistActivity
+ userId: String
+ showPendingWishlist(): void
+removeFromWishlist(propertyId: String): void
<<BoundaryClass>>
PropertyDetailsActivity
+ propertyId: String
+ addToWishlist(): void
+ makeOffer(price: Float): void
+ submitReview(text: String): void
<<BoundaryClass>>
ReviewsActivity
+ userId: String
+ showReviews(): void
+ deleteReview(reviewId: String): void

Fig:09-Boundary Class Diagram

## Entity Classes:

This Entity Class Diagram represents the main data structure of a real estate platform. It includes classes like User, Property, Wishlist, Payment, Review, Report, and Advertisement.The User class stores basic info like name, email, password, role, and status. Based on roles, there are two types: Buyer and Agent. Buyers can manage their wishlist, view bought properties, and make offers. Agents can add properties and approve or reject offers.The Property class holds information such as title, price, location, and the agent who listed it. The Advertisement class shows which property is advertised and for how long.The Wishlist class connects users and properties with offer price and transaction ID. The Payment class tracks payment details like amount and status.The Review class allows users to write feedback on properties. The Report class lets users report others with a reason and status.Overall, this diagram shows how the core data and actions are connected in the system.

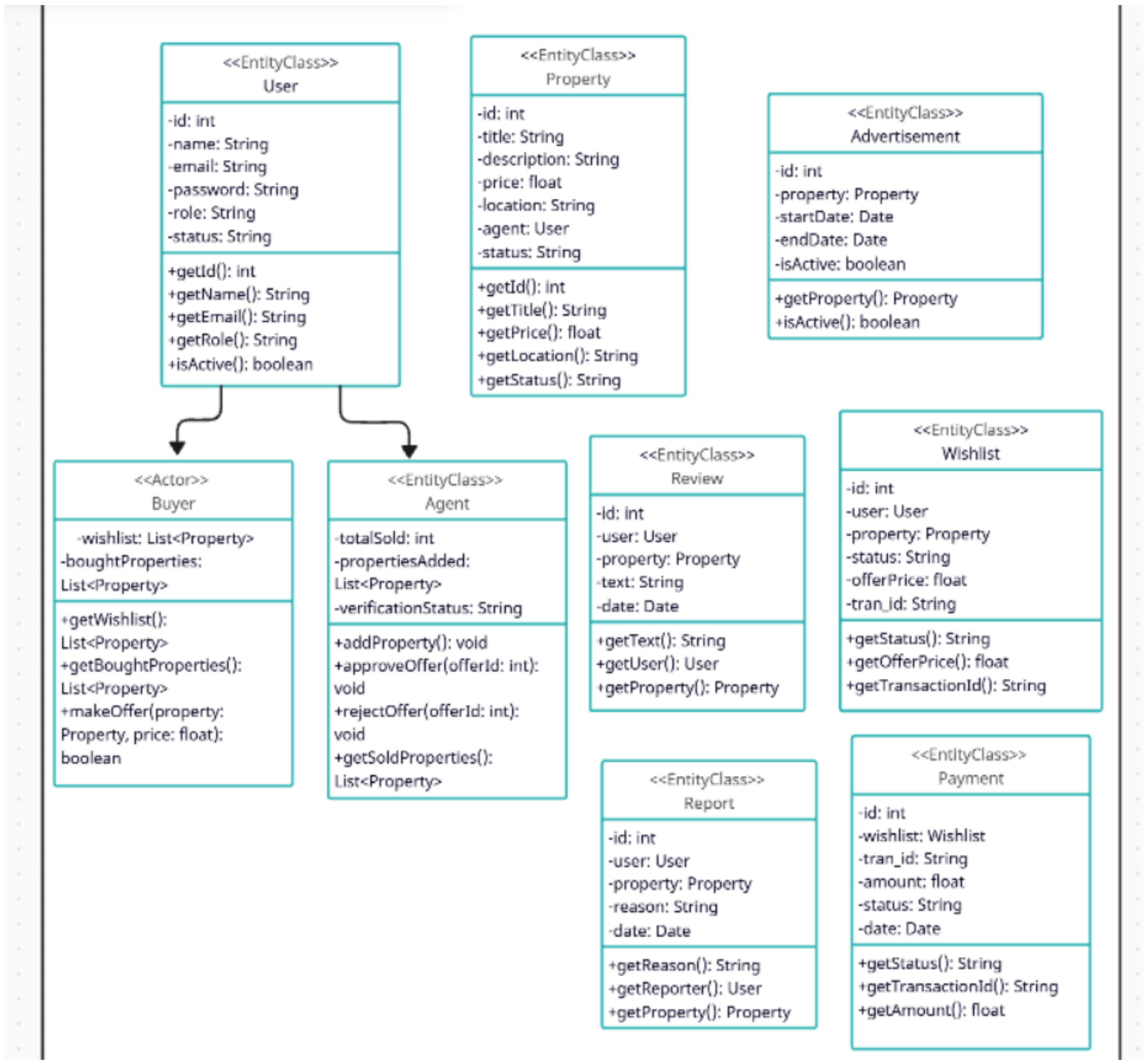

Fig:10-Entity Class Diagram

## Access Classes:

This diagram shows two important classes: LocalDataHandler and RemoteDataHandler. Both are access classes, but they manage data in different ways. LocalDataHandler manages data stored locally on the device. It keeps the database name and version private and provides methods to open the database for reading or writing, create tables, and run queries. This class helps the app handle offline data efficiently. RemoteDataHandler works with data stored on a remote server. It stores the server's address privately and offers various functions such as user login and registration, fetching property details, submitting offers, processing payments, fetching reviews, and uploading property data. These methods allow the app to communicate with the server and manage online data smoothly.

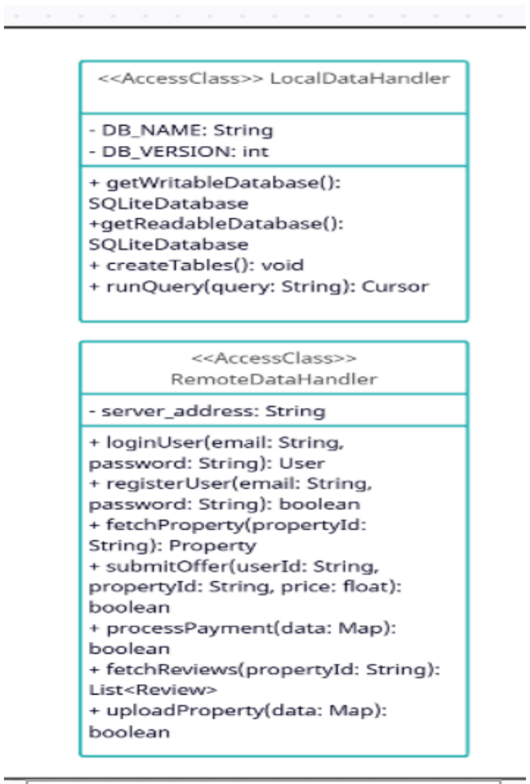
<<AccessClass>> LocalDataHandler
- DB_NAME: String
- DB_VERSION: int
+ getWritableDatabase(): SQLiteDatabase
+getReadableDatabase(): SQLiteDatabase
+ createTables(): void
+ runQuery(query: String): Cursor
<<AccessClass>> RemoteDataHandler
- server_address: String
+ loginUser(email: String, password: String): User
+ registerUser(email: String, password: String): boolean
+ fetchProperty(propertyId: String): Property
+ submitOffer(userId: String, propertyId: String, price: float): boolean
+ processPayment(data: Map): boolean
+ fetchReviews(propertyId: String): List<Review>
+ uploadProperty(data: Map): boolean

Fig:11-Access Class Diagram

## Logic Classes:

This class diagram shows the main parts of a real estate platform's business logic. Each box is a class that has a specific job. The AuthLogic class handles user login, logout, and checking the user's role. People can log in with email/password or Google. The PropertyLogic class manages all property data. It helps to add new properties, see an agent's properties, view approved properties, and mark a property as sold. If a user likes a property, they can save it using the WishlistLogic class and even make an offer by giving a price. These offers are handled by the OfferLogic class, which helps agents to see pending offers, approve or reject them, and mark them as paid. Users can also write reviews about properties using the ReviewLogic class. They can add, view, or delete their reviews. The AdminLogic class is for admins to approve or reject properties and delete any review if needed. All these classes together make the system work smoothly.

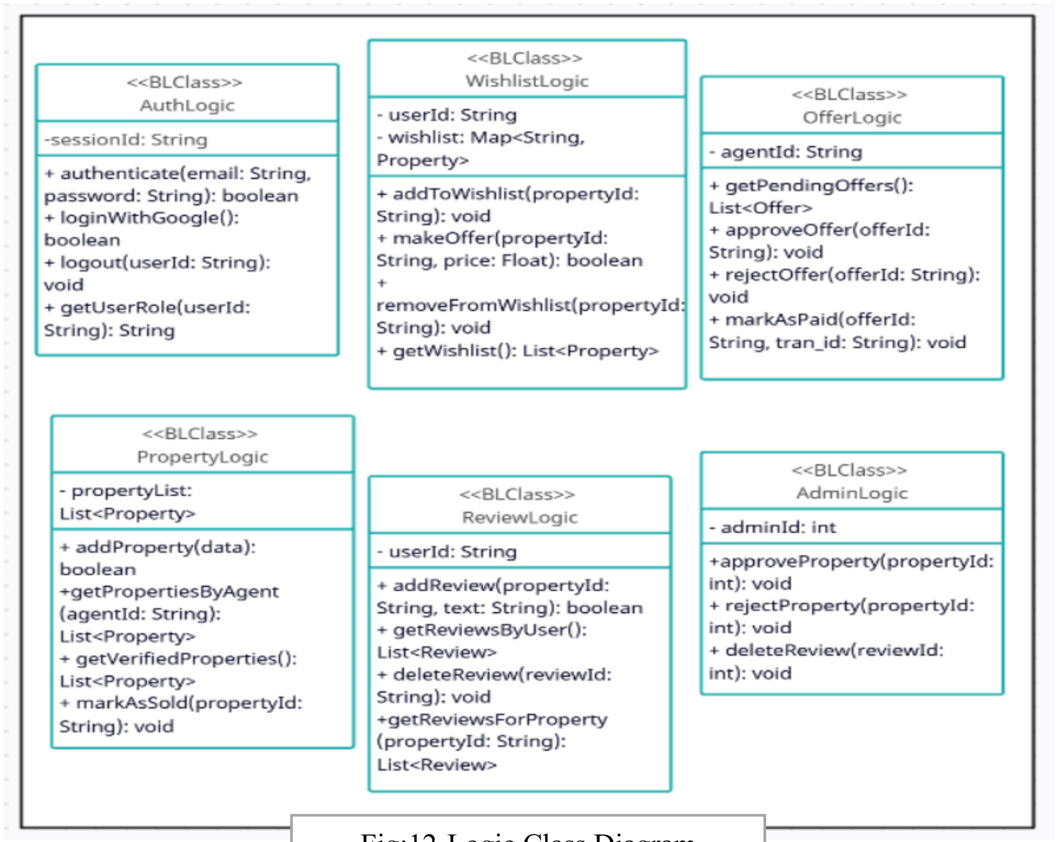
<<BLClass>>
AuthLogic
-sessionId: String
+ authenticate(email: String, password: String): boolean
+ loginWithGoogle(): boolean
+ logout(userId: String): void
+ getUserRole(userId: String): String
<<BLClass>>
WishlistLogic
- userId: String
- wishlist: Map<String, Property>
+ addToWishlist(propertyId: String): void
+ makeOffer(propertyId: String, price: Float): boolean
+ removeFromWishlist(propertyId: String): void
+ getWishlist(): List<Property>
<<BLClass>>
OfferLogic
- agentId: String
+ getPendingOffers(): List<Offer>
+ approveOffer(offerId: String): void
+ rejectOffer(offerId: String): void
+ markAsPaid(offerId: String, tran_id: String): void
<<BLClass>>
PropertyLogic
- propertyList: List<Property>
+ addProperty(data): boolean
+getPropertiesByAgent (agentId: String): List<Property>
+ getVerifiedProperties(): List<Property>
+ markAsSold(propertyId: String): void
<<BLClass>>
ReviewLogic
- userId: String
+ addReview(propertyId: String, text: String): boolean
+ getReviewsByUser(): List<Review>
+ deleteReview(reviewId: String): void
+getReviewsForProperty (propertyId: String): List<Review>
<<BLClass>>
AdminLogic
- adminId: int
+approveProperty(propertyId: int): void
+ rejectProperty(propertyId: int): void
+ deleteReview(reviewId: int): void

Fig:12-Logic Class Diagram

## Entity-Classes Relationship:

This diagram shows the relationship between different parts of a real estate platform. The main entities are User, Property, Agent, and Admin. A User can have many Wishlists, make Payments, and give Reviews. Each Property is linked to one User, one Agent, and one Payment, but one property can also have many Reviews and be part of Advertisements. Agents are responsible for multiple Properties and can post Advertisements. Admins manage the system and are linked to Agents, Advertisements, and Reviews. The Wishlist is connected to both User and Property, and Payments are linked to User and Property as well. Advertisement is managed by an Agent and can have many Reviews linked to it. This diagram helps to understand how all the system parts are connected and work together.

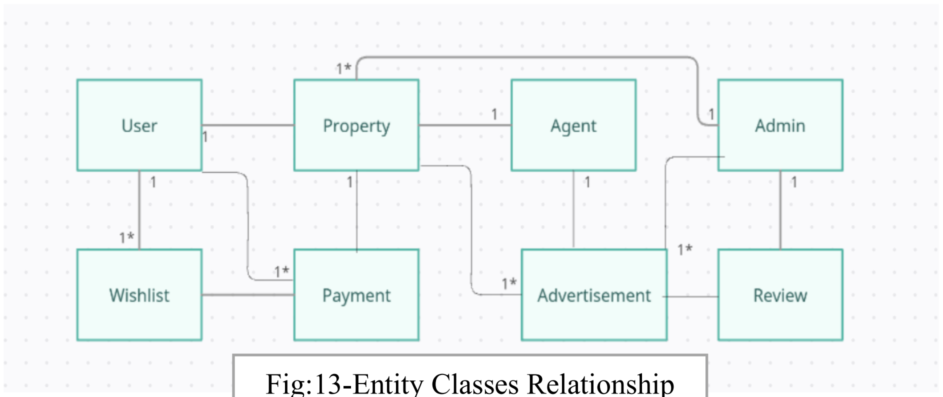

Fig:13-Entity Classes Relationship

## Boundary and Logic Classes Relationship:

This is a Boundary and Logic Classes Relationship diagram that shows how different parts of the system are linked. The system starts with the LoginActivity, which works with AuthLogic to handle user login and registration. After logging in, the user moves to the DashboardActivity, which is the main screen of the system.From the dashboard, the user can go to different features such as viewing property details through PropertyDetailsActivity, checking or managing their wishlist using WishlistLogic, viewing or posting ads with AdvertisementLogic, and reporting issues through ReportActivity. When a user opens PropertyDetailsActivity, they can make an offer using OfferLogic or give a review through ReviewLogic. If payment is needed for offers or reviews, the system uses PaymentLogic to handle it.The solid arrows in the diagram show the main flow of actions, while the dotted arrows show optional or indirect paths. This diagram separates boundary classes (which deal with user screens) from logic classes (which handle the system's background work). It helps to understand how users move through the system and how each part works together.

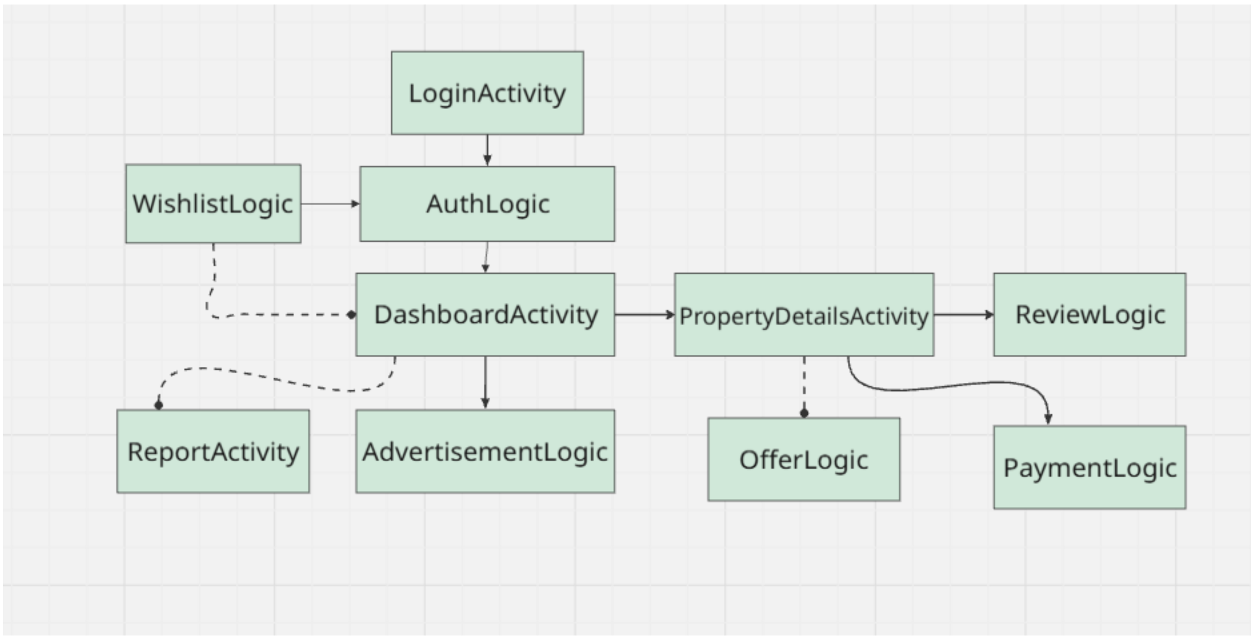

Fig:14- Boundary and Logic Classes Relationship

## Sequence Diagram

The sequence diagrams in this project show how users interact with different components of the real estate platform. They represent step-by-step execution of key use cases like login and property search ,agent offer handling ,admin management ,wishlist and offer flow ,Stripe-based payments, review posting ,and advertisement requests. These diagrams illustrate how data flows between the frontend, backend APIs, and the database, helping users to understand system behavior, verify logic, and ensure smooth integration of components. They are essential for both system design and effective communication.

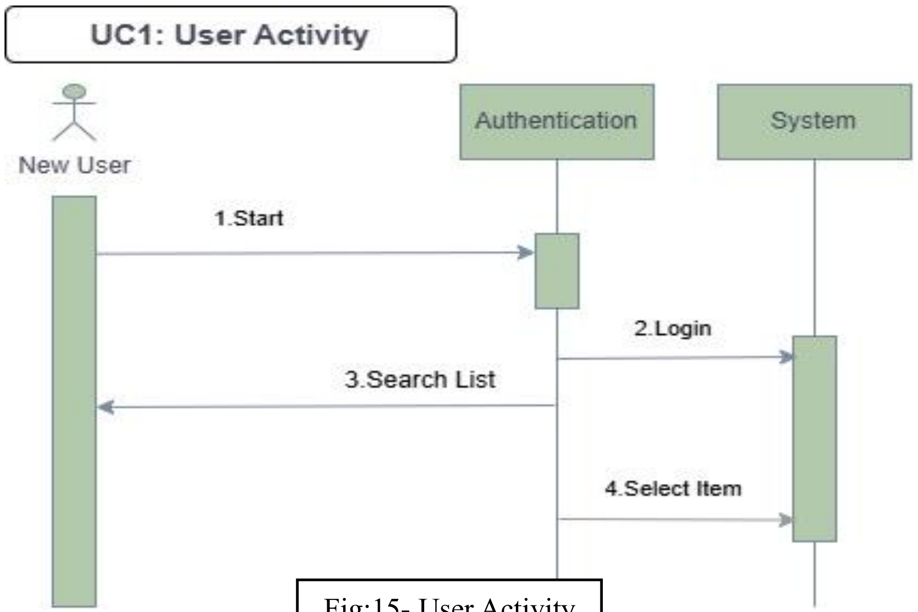

Fig:15-_User Activity

This diagram represents a user interacting with the system to log in, browse items, and select a property. The sequence starts with a "Start" action, then flows through steps like login, search, and selection. Interactions move between User → Authentication → System, ending with the user completing their action.

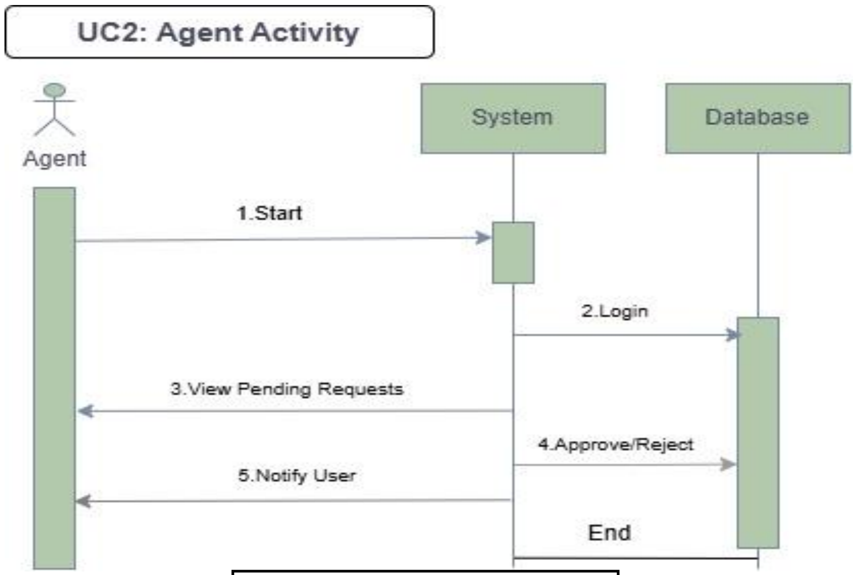

Fig:16-_Agent Activity

This use case illustrates how an agent logs in and manages property requests submitted by users. After logging in, the agent views pending offers and approves or rejects them.
The flow connects Agent → System → Database with clear update and notify steps.

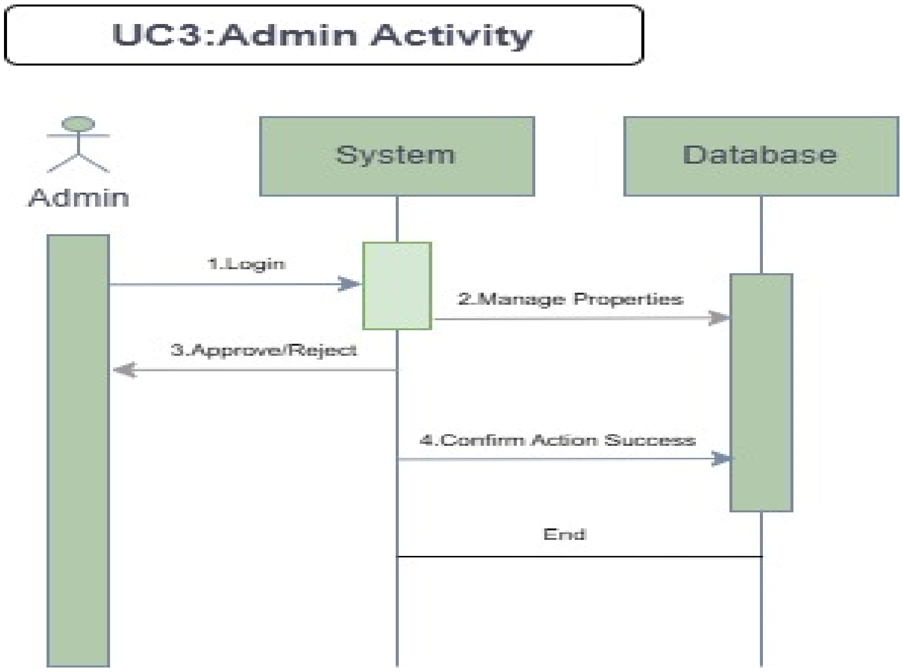

Fig:17-_Admin Activity

This sequence shows basically focuses on admin operations like managing properties. The admin selects actions from a dashboard; the system fetches data and saves updates to the database. It includes multiple branches but follows a consistent Admin → System → Database interaction model.

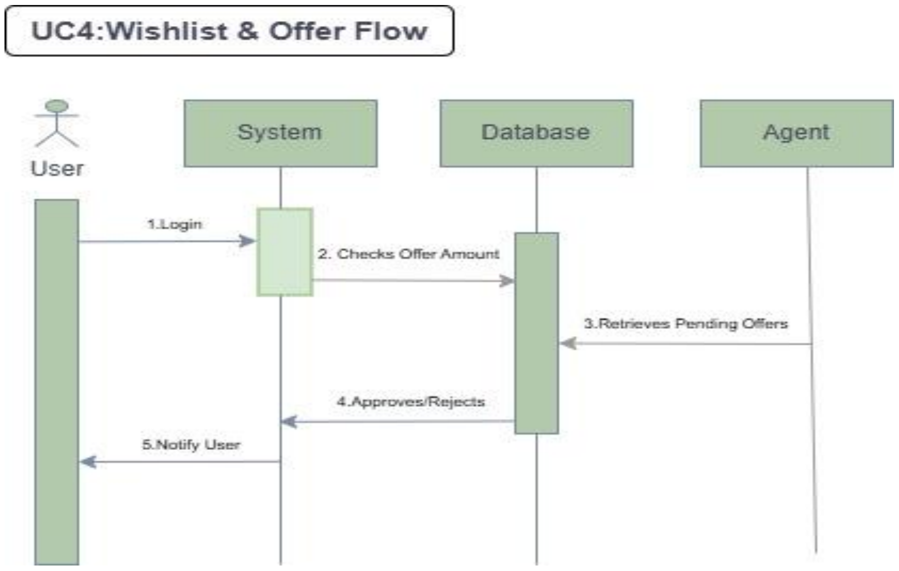

Fig:18-_Wishlist & Offer Flow

In this scenario, user adds a property to their wishlist and sends an offer. The agent views the offer and either approves or rejects it. This sequence shows both User and Agent interacting with the system and database, leading to offer status update.

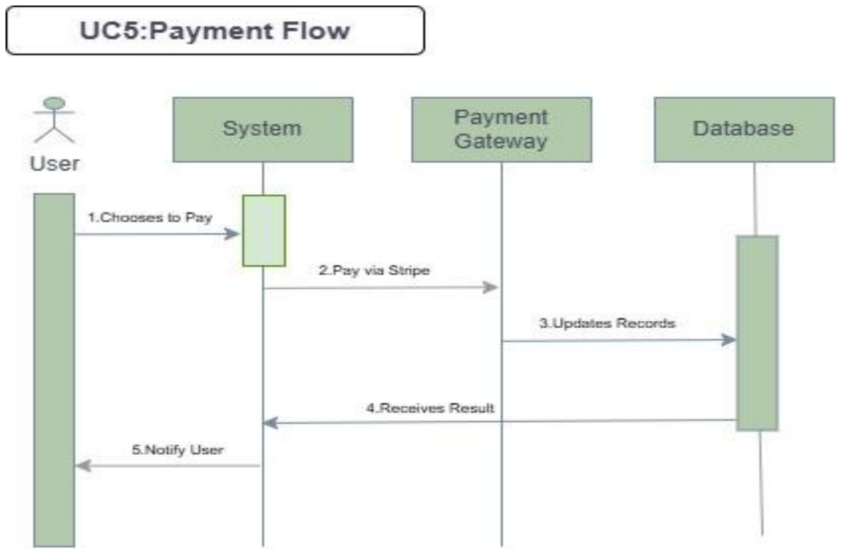

Fig:19- Payment Flow

This diagram begins after offer approval; the user proceeds with payment via Stripe.The system communicates with the payment gateway, and upon success, updates the database. User gets confirmation and sees the property marked as "Purchased."

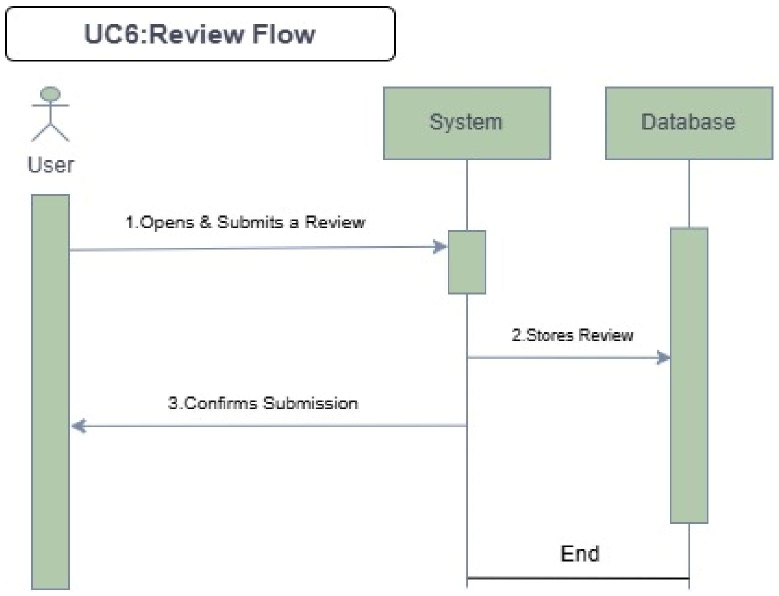

Fig:20- Review Flow

Here, the user adds or deletes a review for a property. The system saves or removes the review in the database accordingly. Flow is with User → System → Database and immediate feedback to the user.

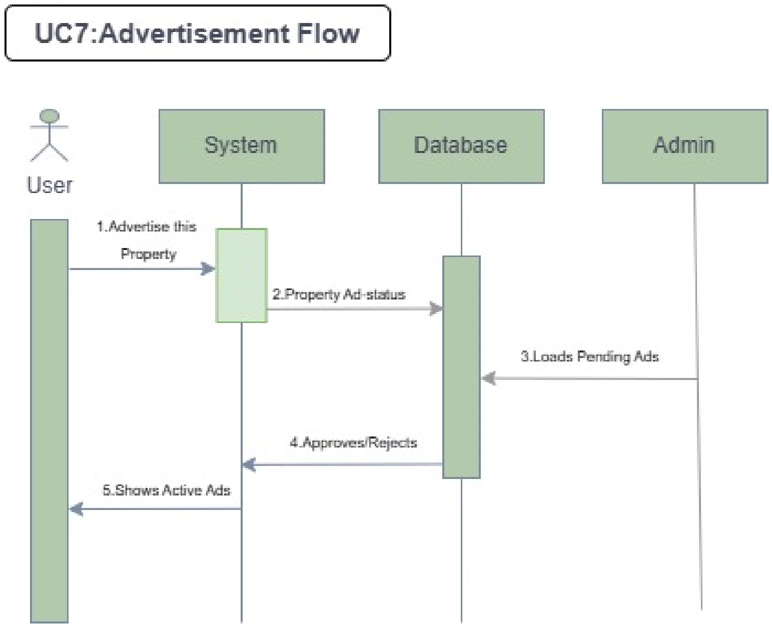

Fig:21-_Advertisement Flow

In this final use case user requests to advertise a property, which is sent for admin approval. Admin reviews and either approves or rejects it. Here, sequence includes User → System → Admin → Database interactions, ending with ad status updates.

## State-Machine Diagram

It represents a single object from one class and tracks how its state changes throughout different events or actions in the system. This diagram helps visualize the behavior of that object over time, showing how it responds to various conditions. It reflects the system from a behavioral perspective.

### Authentication & Role-Based Access:

The flow starts from the Unauthenticated state. After login, the system checks the user role. If the user is an admin, they go to AdminDashboard; if an agent, to AgentDashboard; otherwise, to UserDashboard. On failure, it moves to AuthError. Logout returns to Unauthenticated.

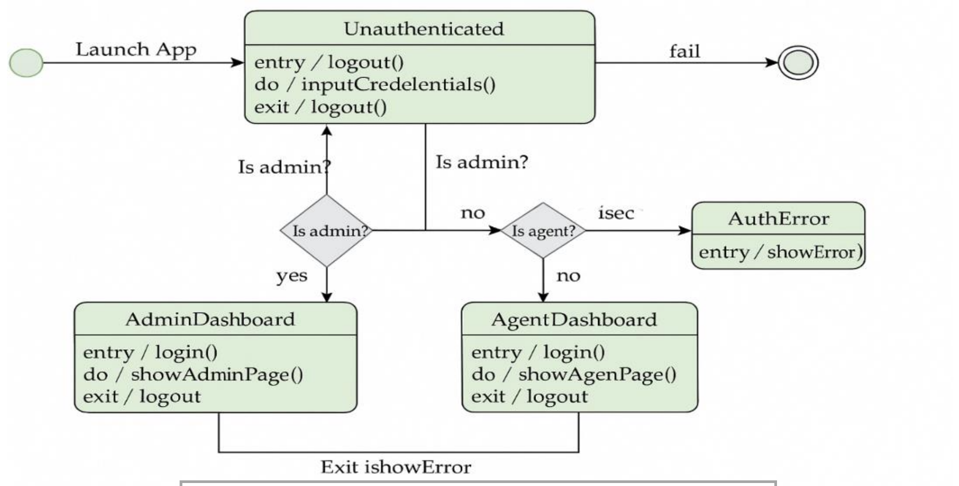

Fig:22-_Authentication & Role-Based Access

### Admin Dashboard Flow:

Once logged in, the admin reaches AdminDashboardHome. From here, they can view users, manage properties, or review reports. Logout from any section brings the state back to Unauthenticated.

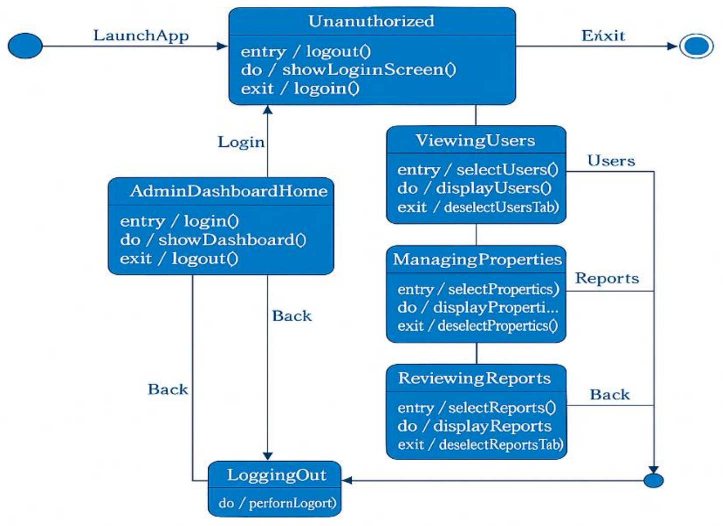

Fig:23-_Admin Dashboard Flow

## Agent Dashboard Flow:

After login, the agent enters AgentDashboardHome. They can add properties, view their listings, check user requests, or review sold items. Logging out resets the state to Unauthenticated.

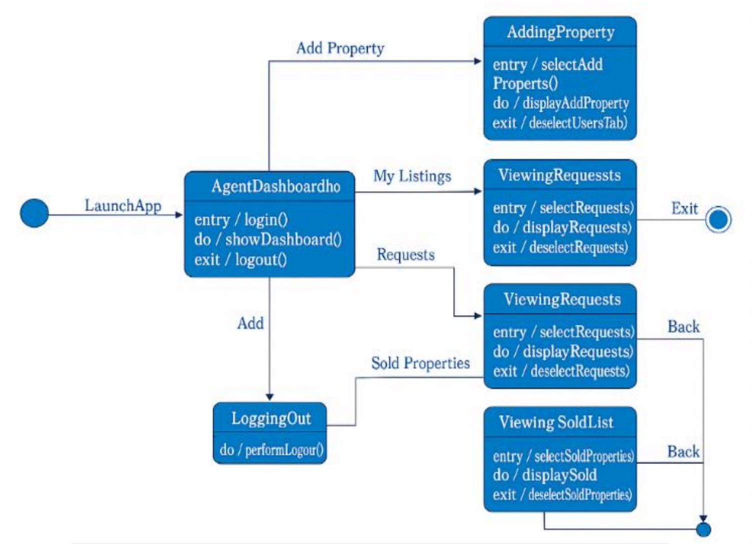

Fig:24-_Agent Dashboard Flow Relationship

## User Dashboard Flow:

A user starts from UserDashboardHome. They can view their wishlist, make offers, check bought properties, write reviews, and make payments. Logging out brings the system back to Unauthenticated.

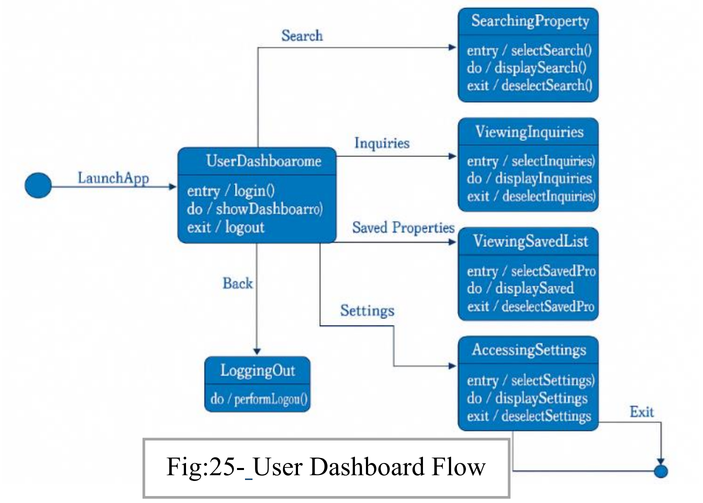

Fig:25-_User Dashboard Flow

## Advertisement Flow:

This diagram shows the advertisement flow. It starts from LaunchApp and moves to Unauthorized, where the user sees the login screen. After login, the user enters ViewingAdvertisements to see ads. From there, they can go to CreatingAdvertisements to create a new ad. After creating, they return to viewing. Logging out leads to LoggingOut, then to Exit.

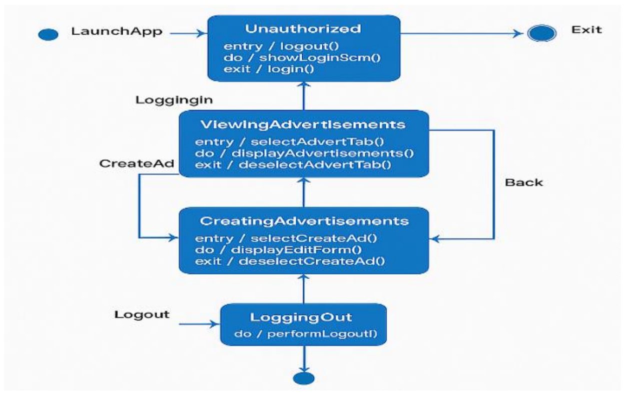

Fig:26-_Advertisement Flow

## Payment / Offer Flow:

The user begins at StartOffer. If the offer is valid, it proceeds to ProcessingPayment. A successful payment leads to PaymentSuccess; failure goes to PaymentError. Invalid offers are sent to OfferInvalid.

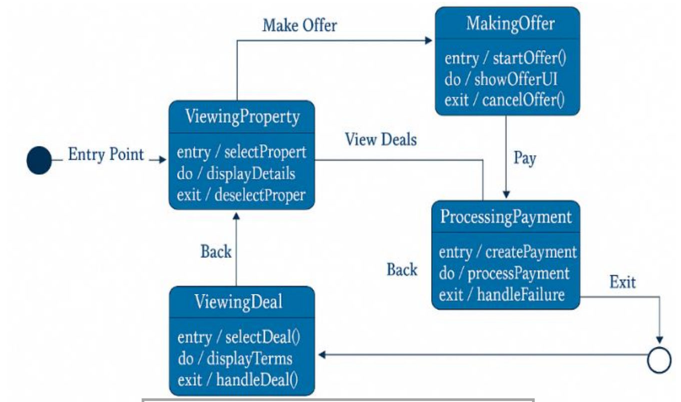

Fig:27-_Payment / Offer Flow

## Review Flow:

This diagram shows a buyer's review flow. After launching the app, the user logs in and reaches BoughtProperties Home. From there, they can write, review, or approve reviews. After actions, they return to the dashboard. Logging out leads to the Exit. Some labels have typos and need fixing.

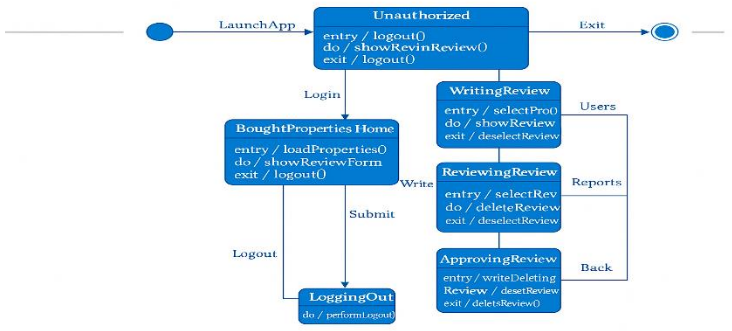

Fig:28-_Review Flow

## Conclusion

Through this design document, we have carefully broken down how our Real Estate Platform will function from the inside. Instead of jumping directly into coding, this planning phase helped us clearly think through the system's structure, data flow, and user interactions. Each diagram and design decision was made to ensure that the platform will be reliable, secure, and easy to use.

By using diagrams like class, activity, sequence, and state diagrams, we could clearly show the structure and behavior of the system. These visual tools made it easier to plan and explain our ideas to others. Every design decision was made to make sure the platform will be easy to use, secure, and efficient for both buyers and agents.

We believe that following this design will reduce future problems and make development smoother. More importantly, this document will serve as a guide for the entire team—so everyone knows how the system works and what their role is. In the end, our goal is not just to build a working system, but to build a smart, well-structured, and user-focused real estate platform.