Design Document

for

Roomble

Version <1.0>

Prepared by

Group #: 7 **Group Name: Marauders**

Aarsh Jain	**230015**	**aarshjain23@iitk.ac.in**
Aritra Ambudh Dutta	**230191**	**aritraad23@iitk.ac.in**
Aritra Ray	**230193**	**aritrar23@iitk.ac.in**
Bhukya Vaishnavi	**230295**	**bhukyav23@iitk.ac.in**
Bikramjeet Singh	**230298**	**bsingh23@iitk.ac.in**
Hitarth Makawana	**230479**	**hitarthkm23@iitk.ac.in**
Shlok Jain	**230493**	**jainshlok23@iitk.ac.in**
Ronav Puri	**230815**	**ronavg23@iitk.ac.in**
Rathod Ayushi	**230844**	**rathoday23@iitk.ac.in**
Saksham Verma	**230899**	**sakshamv23@iitk.ac.in**
Surepally Pranaysriharsha	**231057**	**surepally23@iitk.ac.in**

Course: **CS253**

Mentor TA: **Nij Bharatkumar Padariya**

Date: **07/02/2025**

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.0	Aarsh Jain Aritra Ambudh Dutta Aritra Ray Bhukya Vaishnavi Bikramjeet Singh Hitarth Makawana Shlok Jain Ronav Puri Rathod Ayushi Saksham Verma Surepally Pranaysriharsha	First Version of the Software Design Document	07/02/2025

1 Context Design

1.1 Context Model

The system works smoothly with its different subsystems, all of which come together to create the best possible networking experience for users. These subsystems are linked to each other and the server, handling necessary fetch and update requests. Often, using one subsystem naturally leads to using another as a follow-up.

The model and its subsystems are as follows:

- **User Authentication System:** Using the user's email, the system authenticates the signup for first-time login using OTP.

- **User Profile Management:** The system maintains the user profiles of the tenants and landlords using databases and the various associated attributes while allowing them to add or remove several features.

- **Property Listing System:** The landlord is given the privilege of listing the property for renting out, where he/she can provide details about the same, including but not limited to price, amenities, etc.

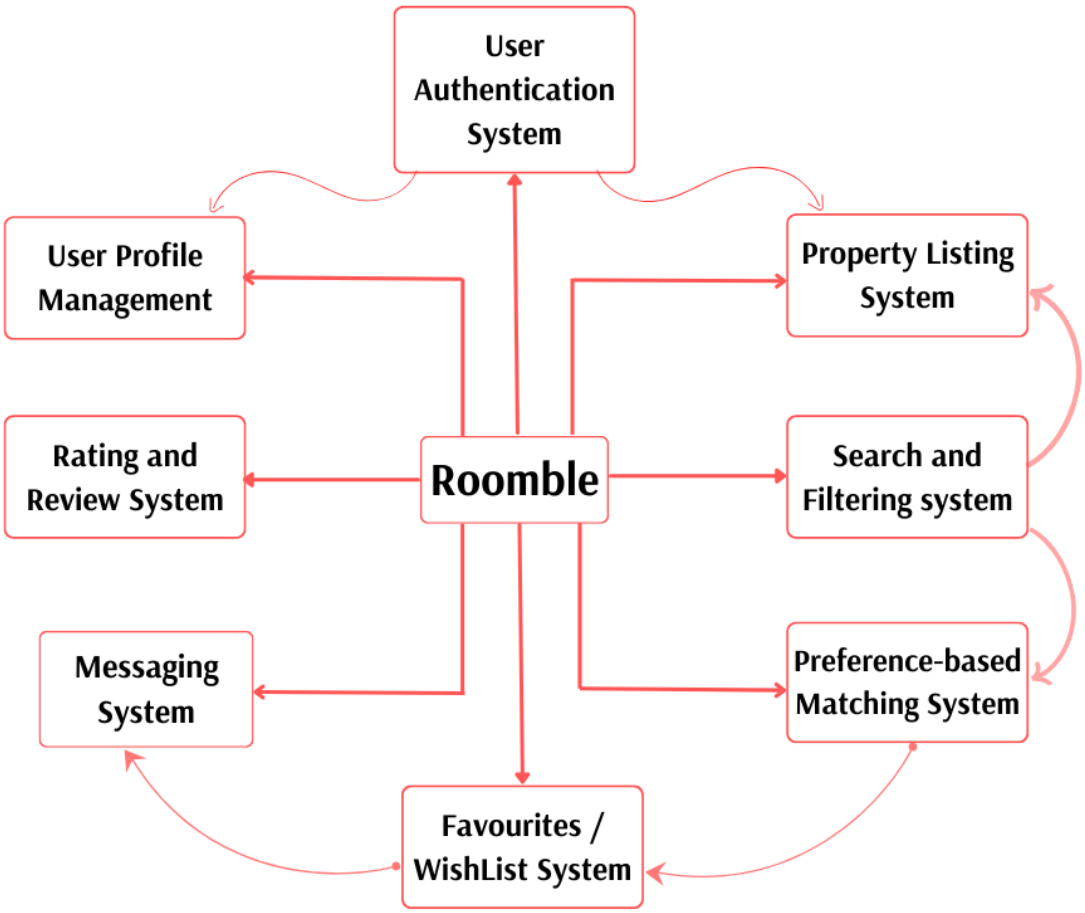
- **Rating and Review System:** One of the system's key features, it allows the tenants to review and rate the properties they have lived on as well as the flatmates they have shared a property with. The landlords also have the option to do the same for any tenant, hence providing a two-fold review system, ensuring impartiality for both sides.

- **Searching and Filtering System:** The system provides a simple yet efficient search and filtering system, allowing users to search for properties or flatmates according to their preferences/choices. This eases their process of making a choice.

- **Messaging System:** Allows users to connect easily with each other if they are interested in the concerned property or find each other's potential flatmates. One can easily chat over any concerns before visiting the property or meeting the potential flatmate in person.

- **Favorites/Wishlist System:** A list of marked properties or persons that a user might be interested in renting or sharing respectively, but is not sure of the time. Allows users to save the details for future reference.

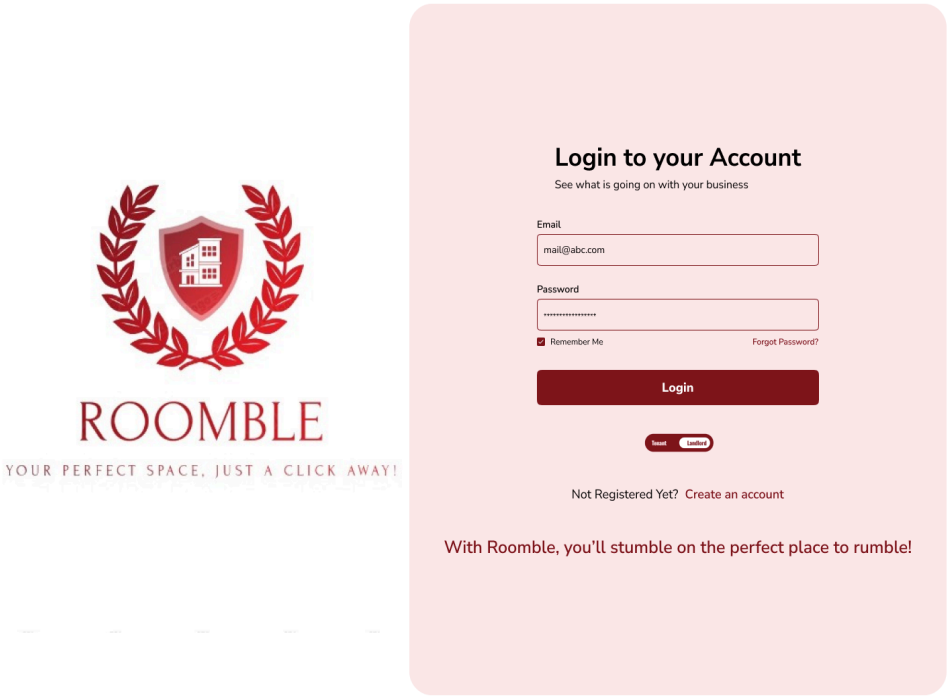
- **Preference-based Matching System:** The searching filter allows users to set their preferences on essential questions and then provides them with the best possible matches. This will enable them to navigate and make an efficient choice easily.



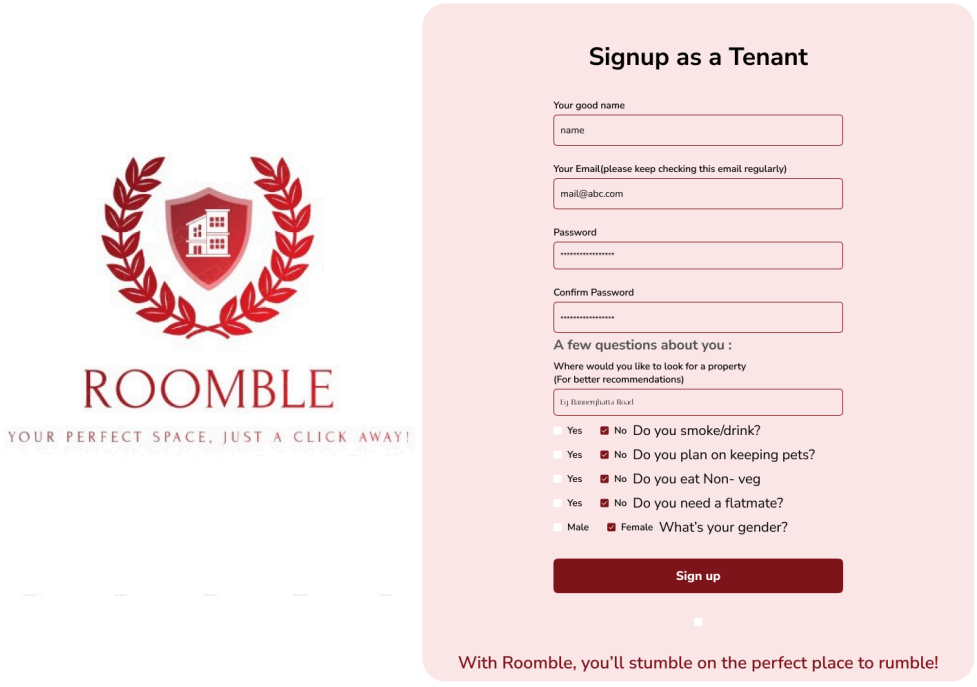
1.2 Human Interface Design

This section includes early drafts that lay out the interfaces for different software parts. These drafts act as a handy guide for developers and system designers, giving a clear breakdown of how the software is structured and how users will interact with it. They cover key features, provide visual examples of screen layouts, and outline the different options available to users. The goal is to make everything feel smooth and intuitive, ensuring a user-friendly experience. Using these drafts, developers can keep things consistent and well-organized throughout development.

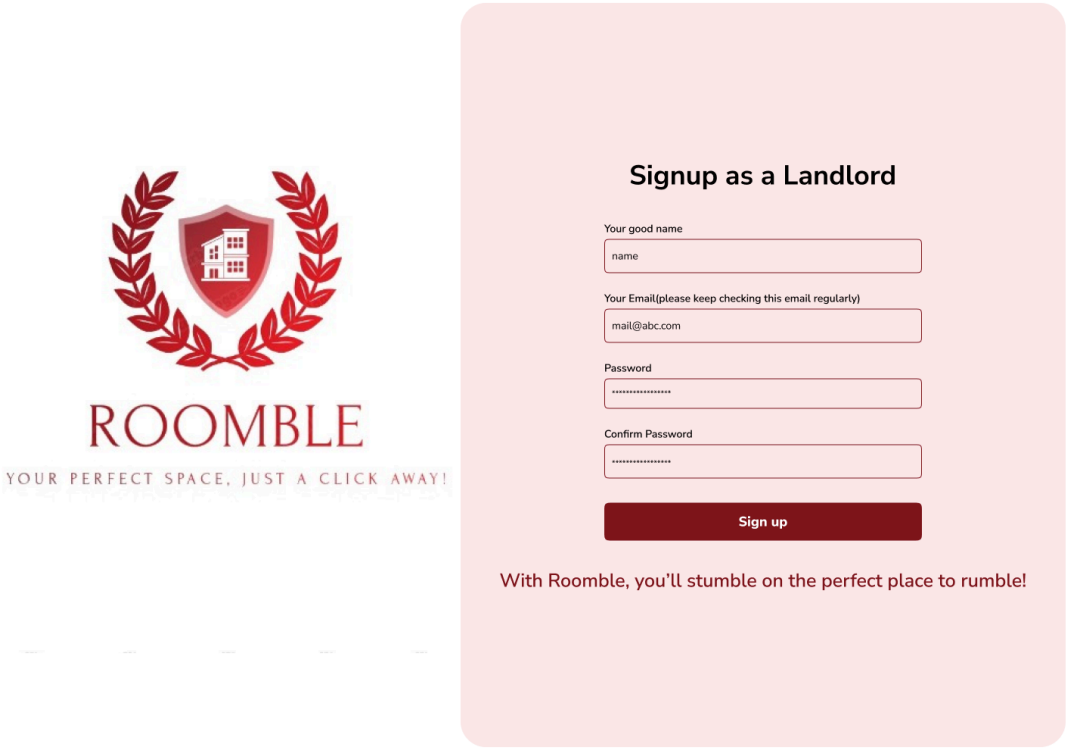
1.2.1 Authentication Designs



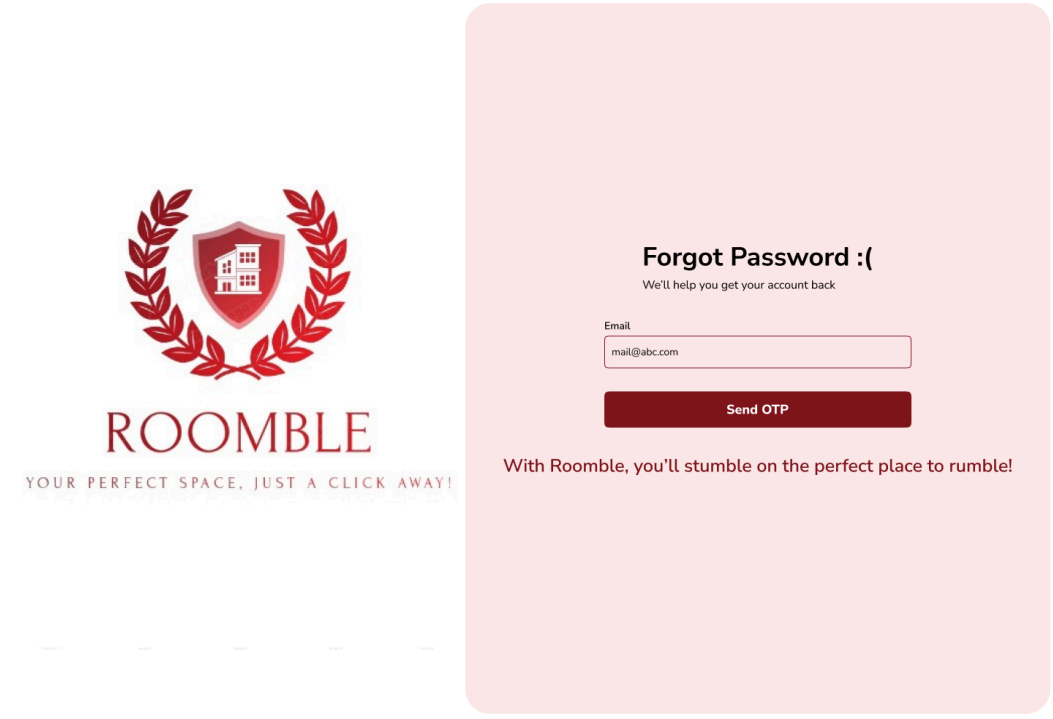
Login page where the user can log in according to their role (Tenant/ Landlord)



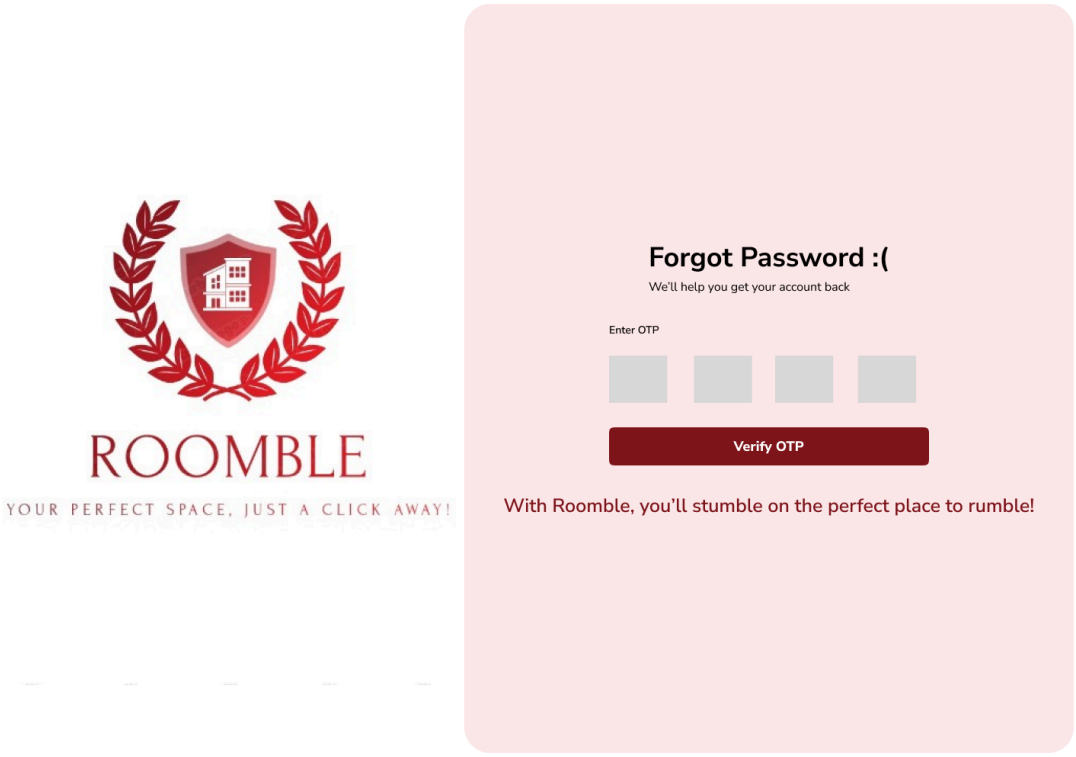
Tenants will be prompted about their personal preferences before creating an account



Landlord's sign-up page

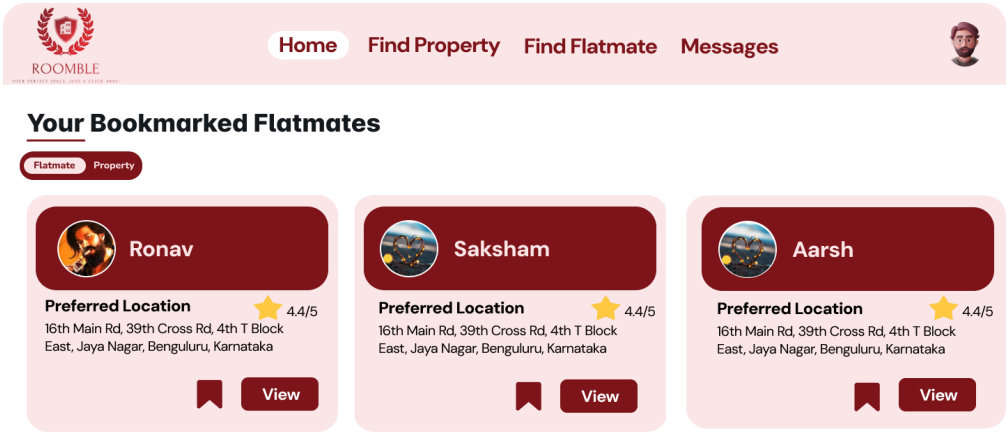


In case someone forgets their password, we help them recover it by verifying their email

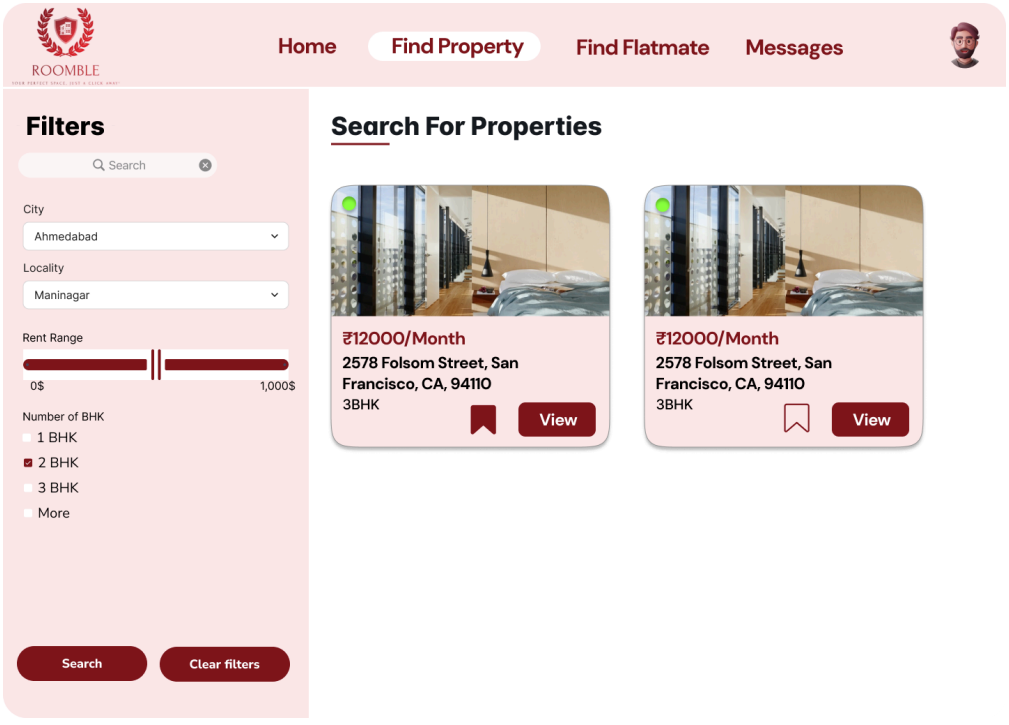


The user is sent an OTP on their email to recover their password

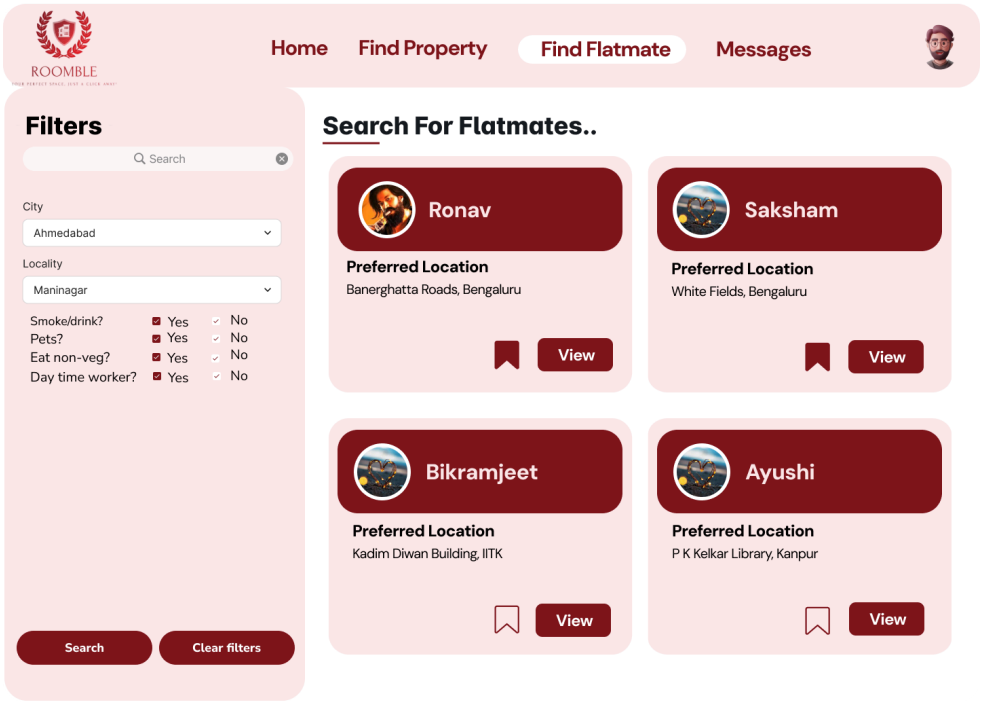
1.2.2 Tenant Designs



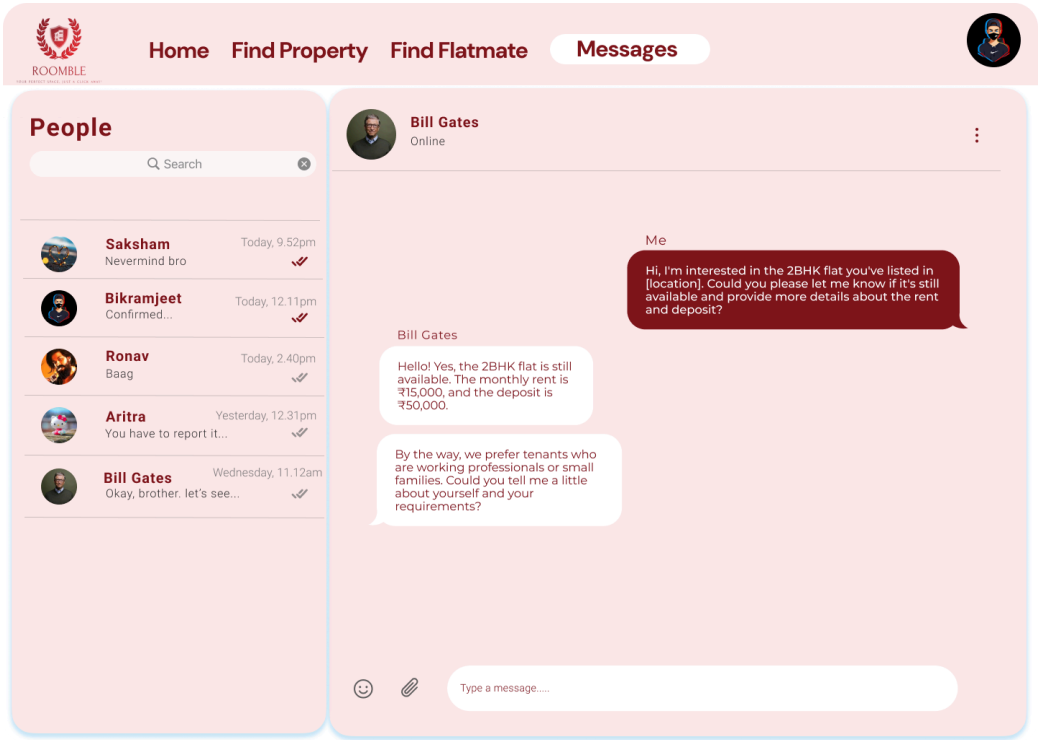
Tenant's Homepage



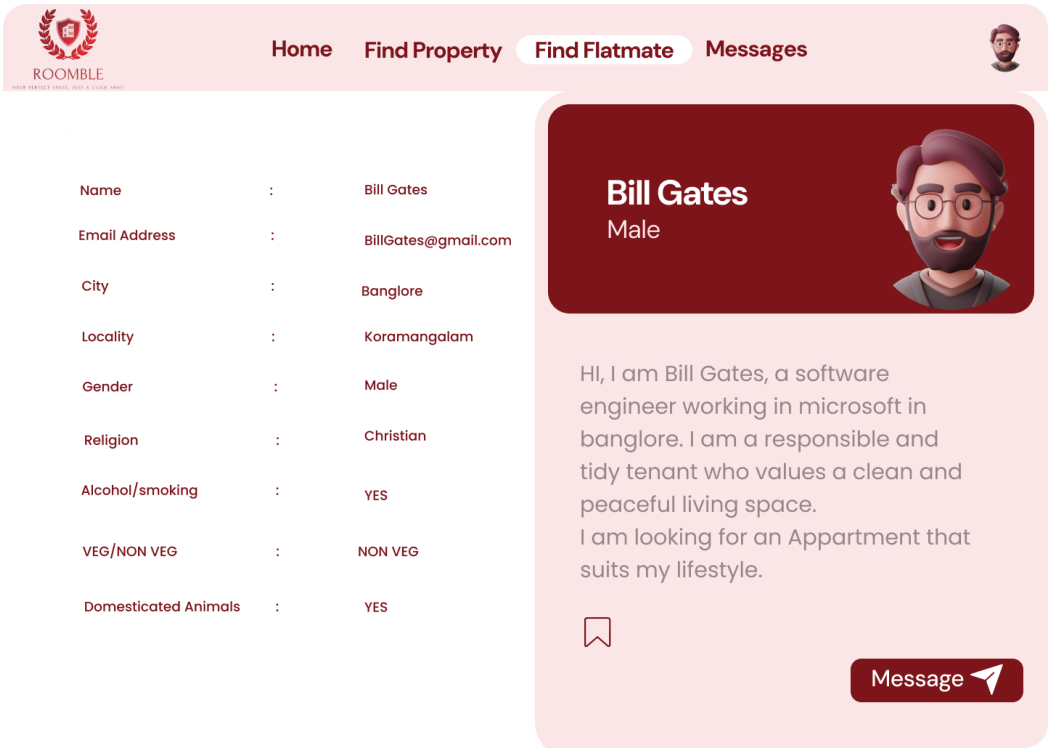
Tenants can browse properties listed by Landlords.



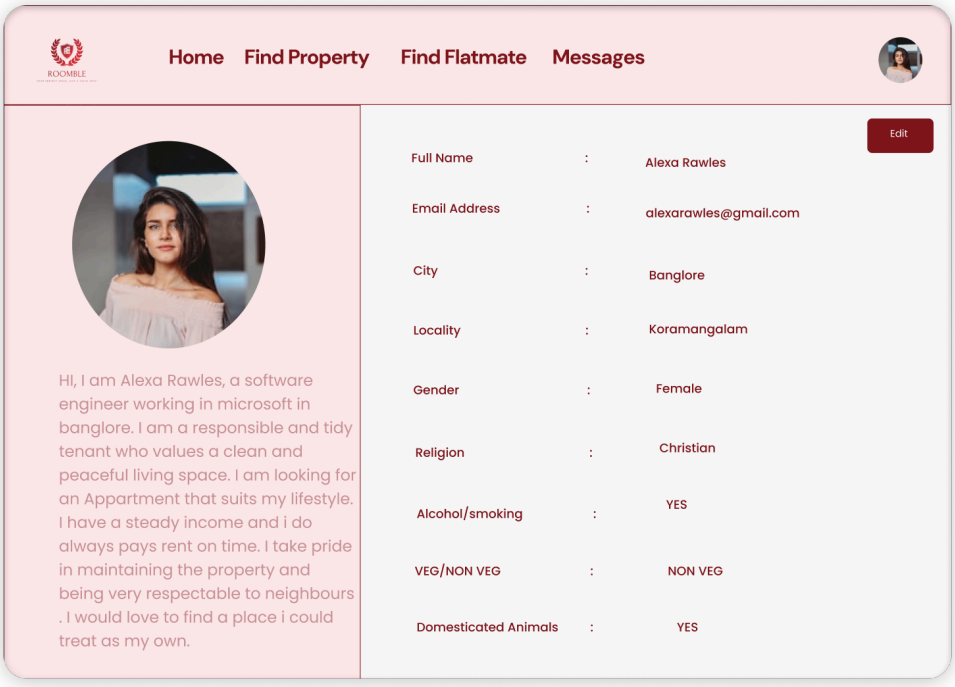
Tenants can look for other people looking for flatmates



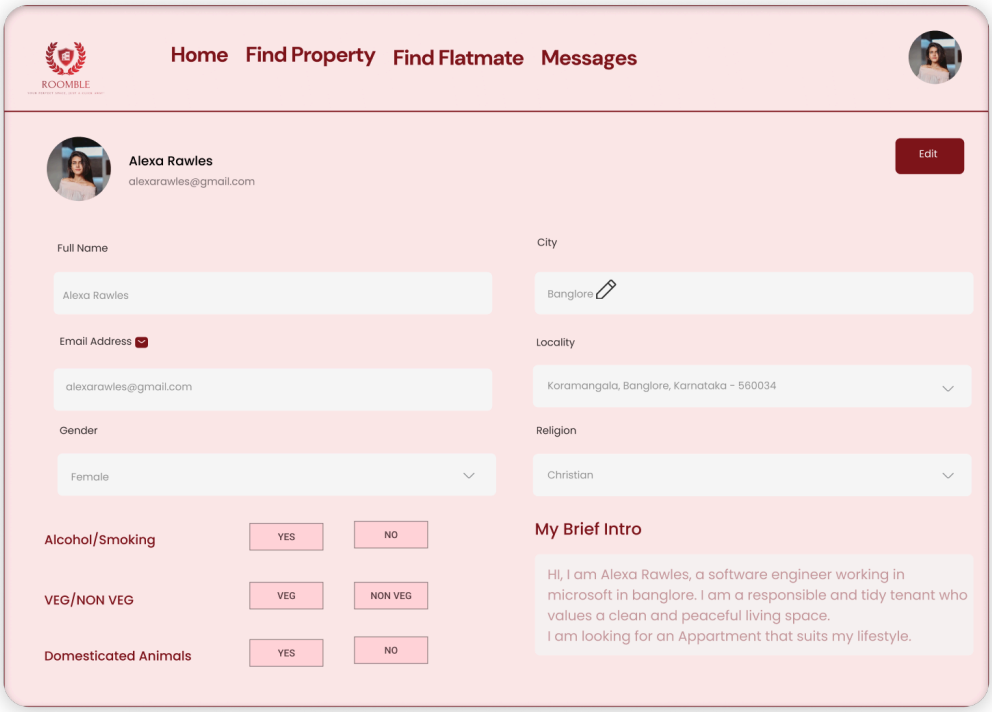
Tenants can talk via DM with Landlords/Flatmates to confirm compatibility.



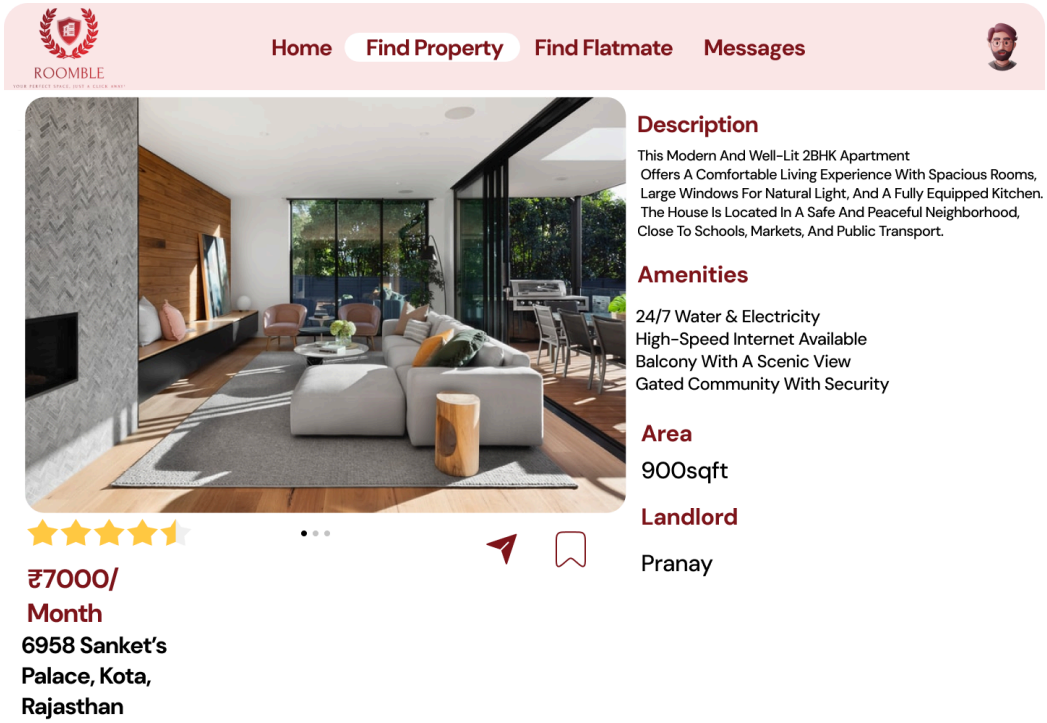
Tenants can see precise details about other people who are looking for flatmates and contact them via messaging service



*Tenant's complete profile according to the details submitted at the time of profile, which can be edited by clicking the **Edit** option*

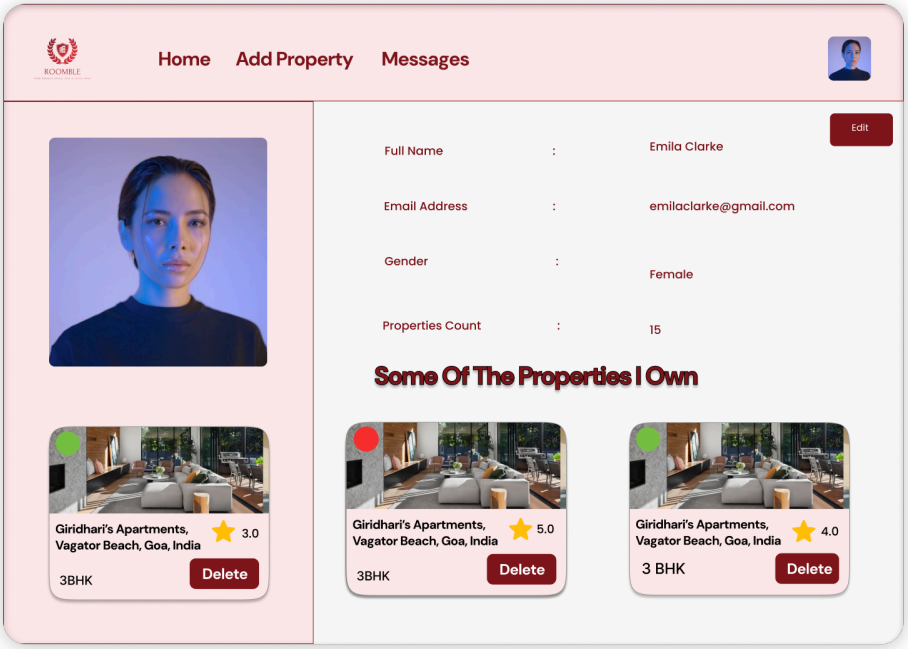


*The tenant can edit his/her details by clicking on the **Edit** option.*

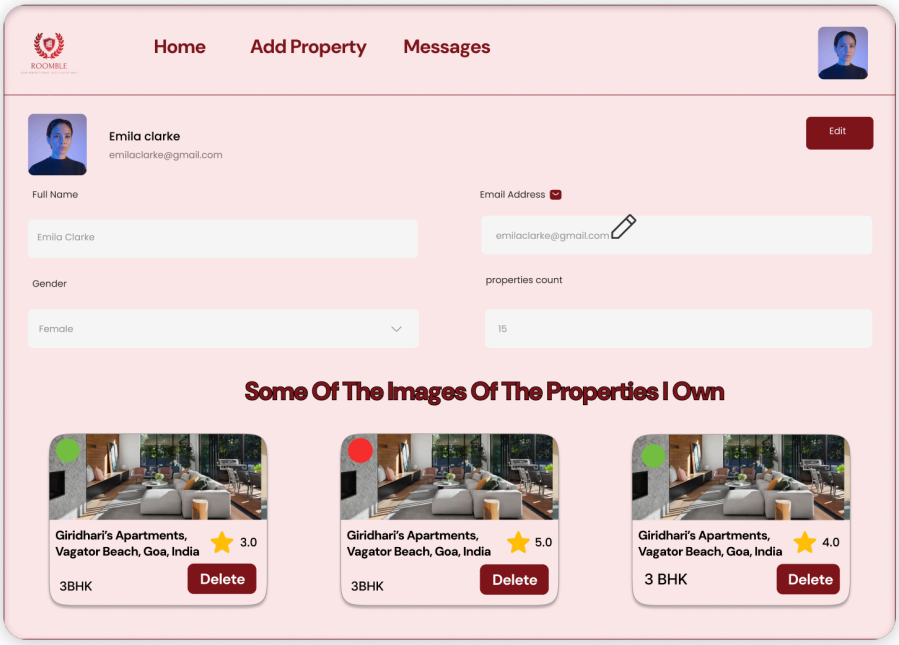


Precise property details appear when the tenant clicks on the property card.

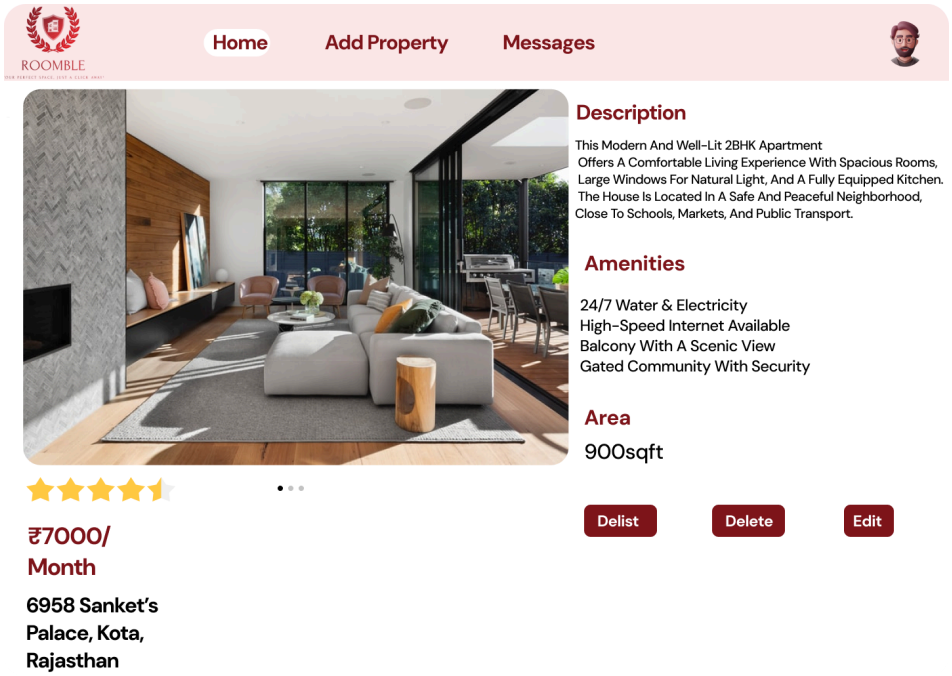
1.2.3 Landlord designs



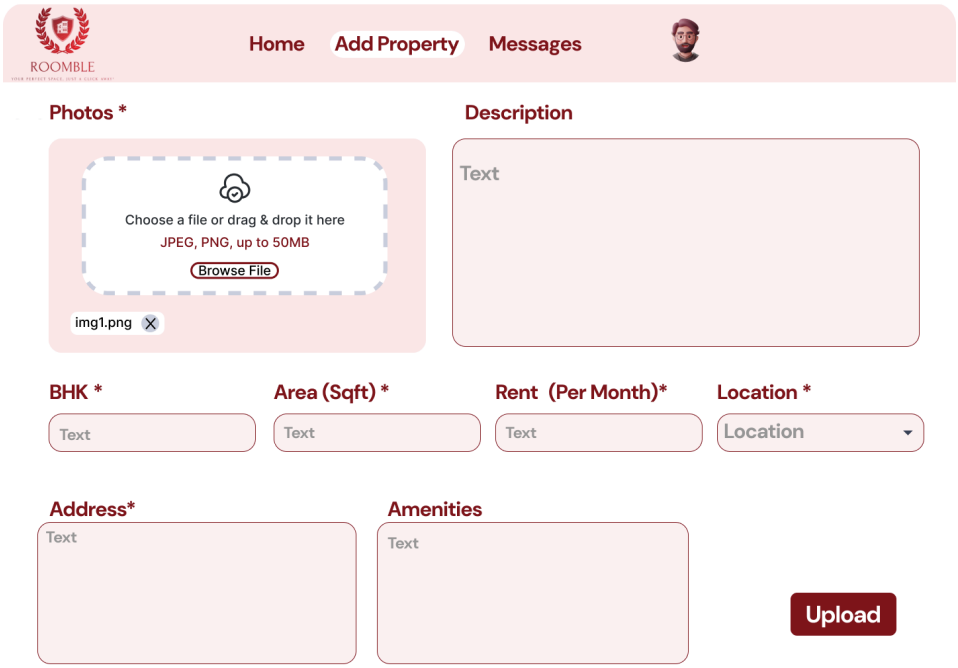
The landlord profile page which appears when he clicks on the profile icon on the navigation bar



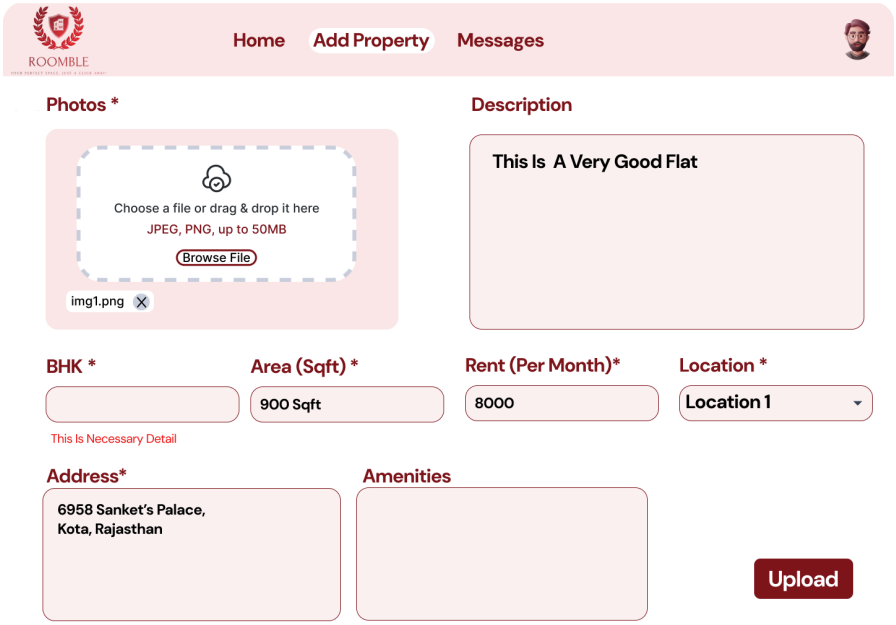
The landlord can edit his profile by clicking on the edit option.



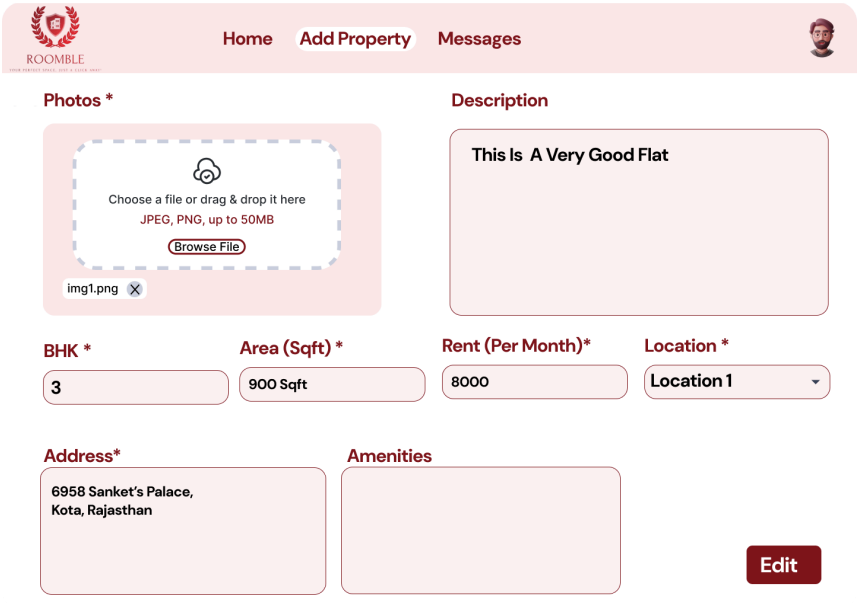
The landlord can view his property details like this after uploading the property



Form for adding/uploading a new property



Necessary details must be uploaded before the final upload.



The Landlord can edit the details of his property at any time

2 Architecture Design

Model

The model is responsible for managing all the data in the rental platform. It bridges the application and the database, ensuring that data is stored, retrieved, and updated correctly.

- **Data Storage & Retrieval:** It saves information such as property listings, user profiles, messages, and search filters in the database.
- **Database Management:** We are using **MongoDB**, a flexible and scalable database, to store all platform data.
- **Backend Interaction:** The model communicates with the backend, built using **Node.js and Express.js**, to process data requests efficiently.
- **Ensuring Data Consistency:** It makes sure that when a landlord uploads a property or a tenant searches for a listing, the correct information is fetched and displayed accurately.

The **model** is where all the platform's essential data is handled, making sure everything runs smoothly and efficiently.

View

The **view** is responsible for how users interact with the rental platform. It ensures that information is displayed clearly and that users can navigate the website easily.

- **User Interface & Experience:** The view creates a visually appealing and user-friendly interface for tenants, landlords, and administrators.
- **Frontend Development:** We use **React.js**, allowing fast, interactive, and efficient rendering of property listings, search filters, and user actions.
- **Handling User Interactions:** The view captures user inputs, such as property searches, messages, and listing uploads, and sends them to the backend for processing.

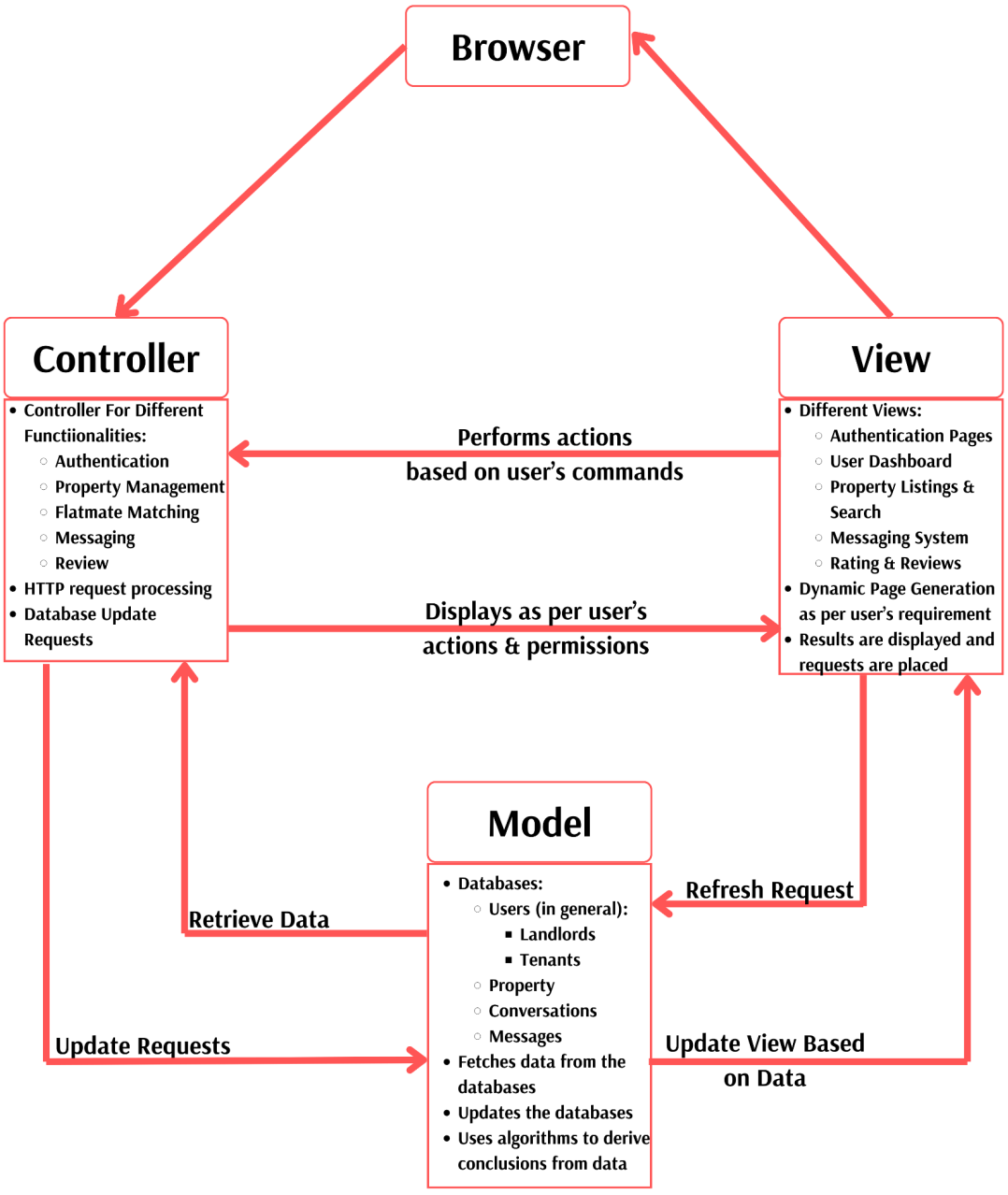
The **view** makes the rental platform easy to use, visually appealing, and interactive, ensuring a smooth experience for all users.

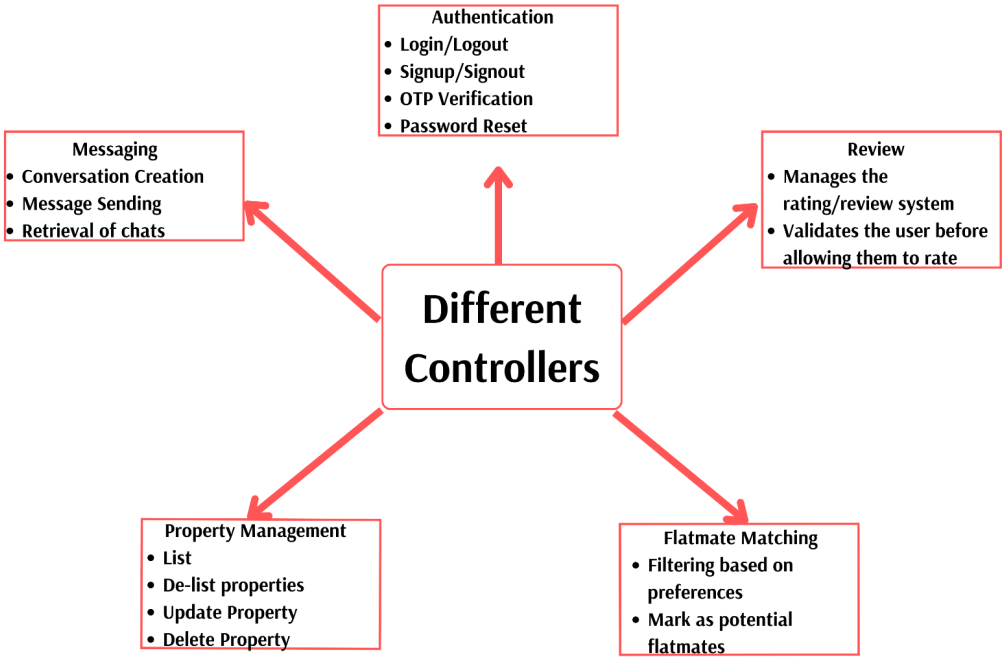
Controller

The **Controller** manages user requests and processes data within the rental platform. It ensures smooth communication between the front end and the database while handling business logic efficiently.

- **Request Handling:** The Controller processes user actions, such as searching for properties, booking listings, and sending messages.
- **Backend Development:** We use **Node.js and Express.js**, which provide a fast, scalable, and efficient way to manage API requests and responses.
- **Data Processing:** The Controller validates inputs, retrieves or updates data from the database, and sends responses back to the front end.
- **Security & Authentication:** It ensures secure user authentication, role-based access control, and data protection.

The Controller acts as the platform's backbone, ensuring smooth operations, secure transactions, and a seamless connection between the user interface and the database.

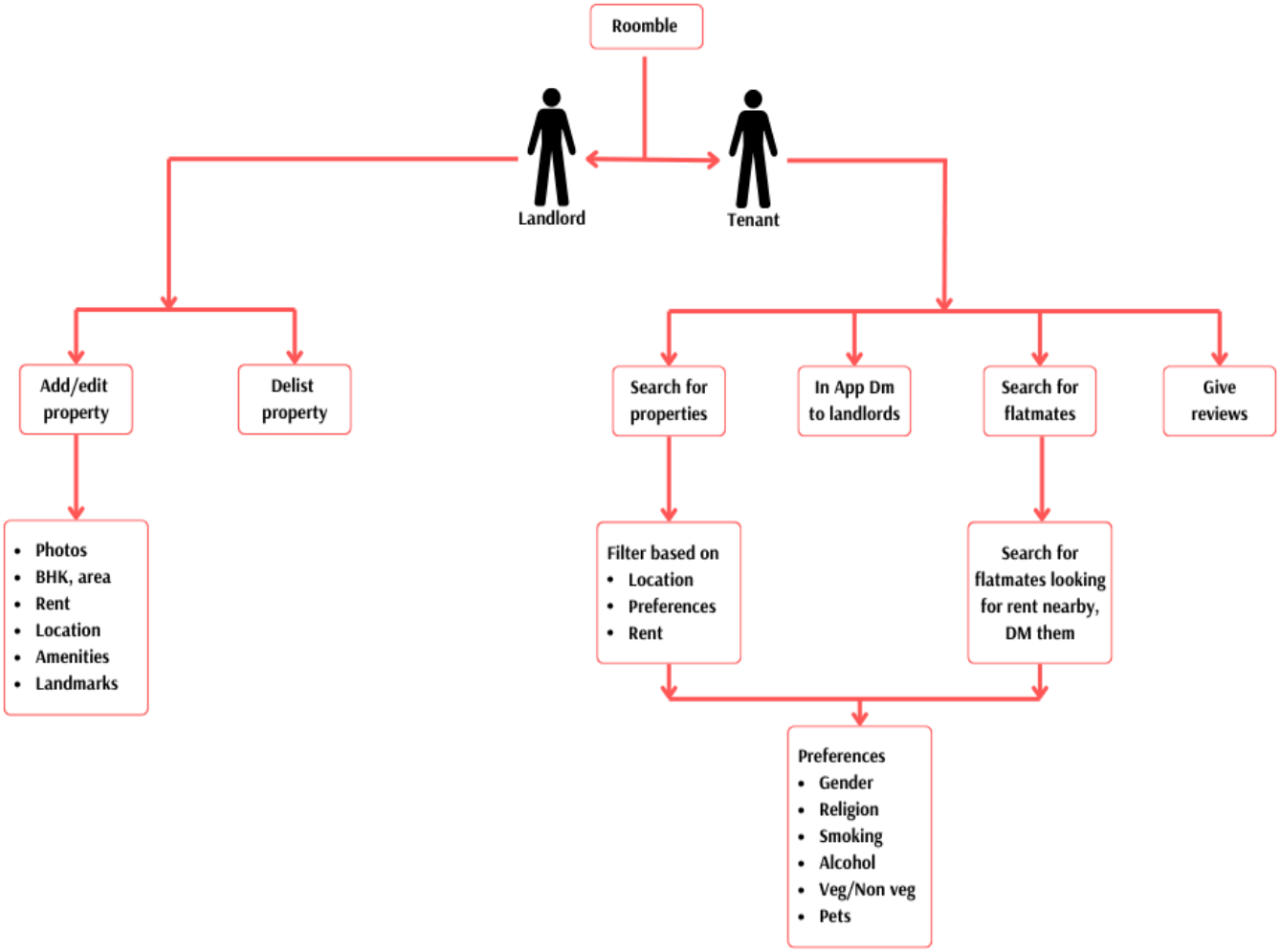

Browser
Controller
• Controller For Different Functiionalities:
◦ Authentication
◦ Property Management
◦ Flatmate Matching
◦ Messaging
◦ Review
• HTTP request processing
• Database Update Requests
View
• Different Views:
◦ Authentication Pages
◦ User Dashboard
◦ Property Listings & Search
◦ Messaging System
◦ Rating & Reviews
• Dynamic Page Generation as per user's requirement
• Results are displayed and requests are placed
Performs actions based on user's commands
Displays as per user's actions & permissions
Model
• Databases:
◦ Users (in general):
▪ Landlords
▪ Tenants
◦ Property
◦ Conversations
◦ Messages
• Fetches data from the databases
• Updates the databases
• Uses algorithms to derive conclusions from data
Retrieve Data
Refresh Request
Update Requests
Update View Based on Data



We chose the **MVC (Model-View-Controller)** architecture for our rental platform because it helps organize the code efficiently, making development, maintenance, and scaling easier.
This structure improves **code reusability, scalability, and maintainability**, allowing different platform parts to be updated independently without affecting the entire system. It also ensures a **clear workflow**, making debugging and feature enhancements more manageable.

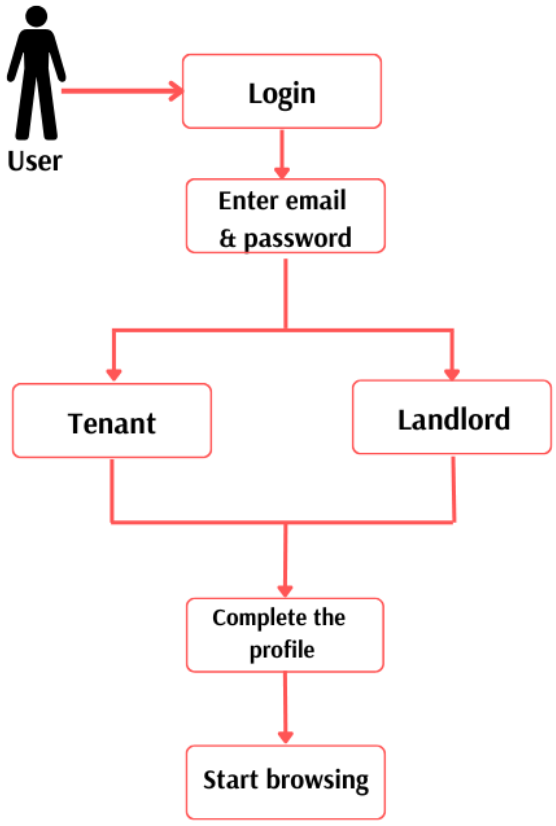
3 Object Oriented Design

3.1 Use Case Diagrams



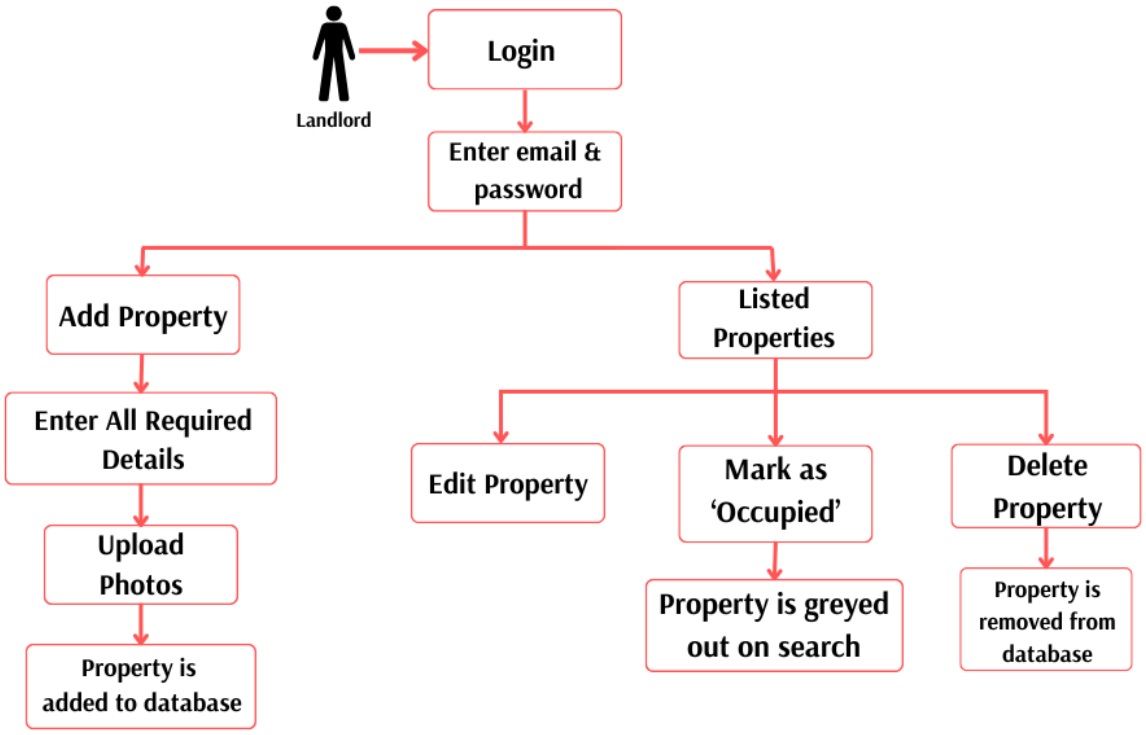
A brief overview of the complete model and what is expected to be done with its help

3.1.1 Use Case 1 (UC1): Enabling User Account Creation on the platform:



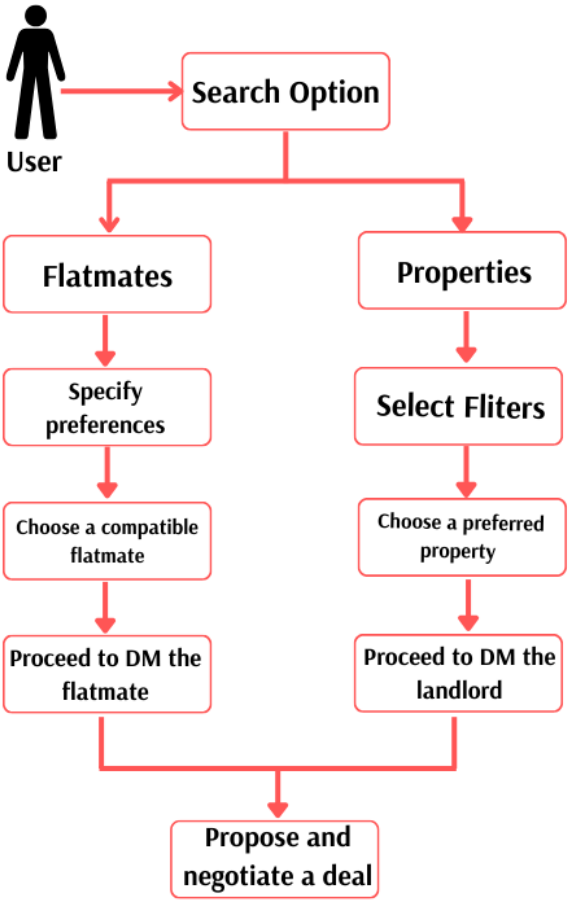
The purpose of this use case is to allow the user to create an account on the web application, enabling them to use it. To create an account a user must have a valid email address and create a unique password so that the admin can authorize the account formation. The priority of this use case is high since, without creating the account, the user won't be able to use the platform. The user must have a valid email ID as a prerequisite to making this account. After account creation, the user's email and password will be updated in the database so that the user can log in and use the platform. The actors involved would be the users looking forward to using the application.

3.1.2 Use Case 2 (UC2): Adding, Editing, Delisting or Deleting Property by Landlord:



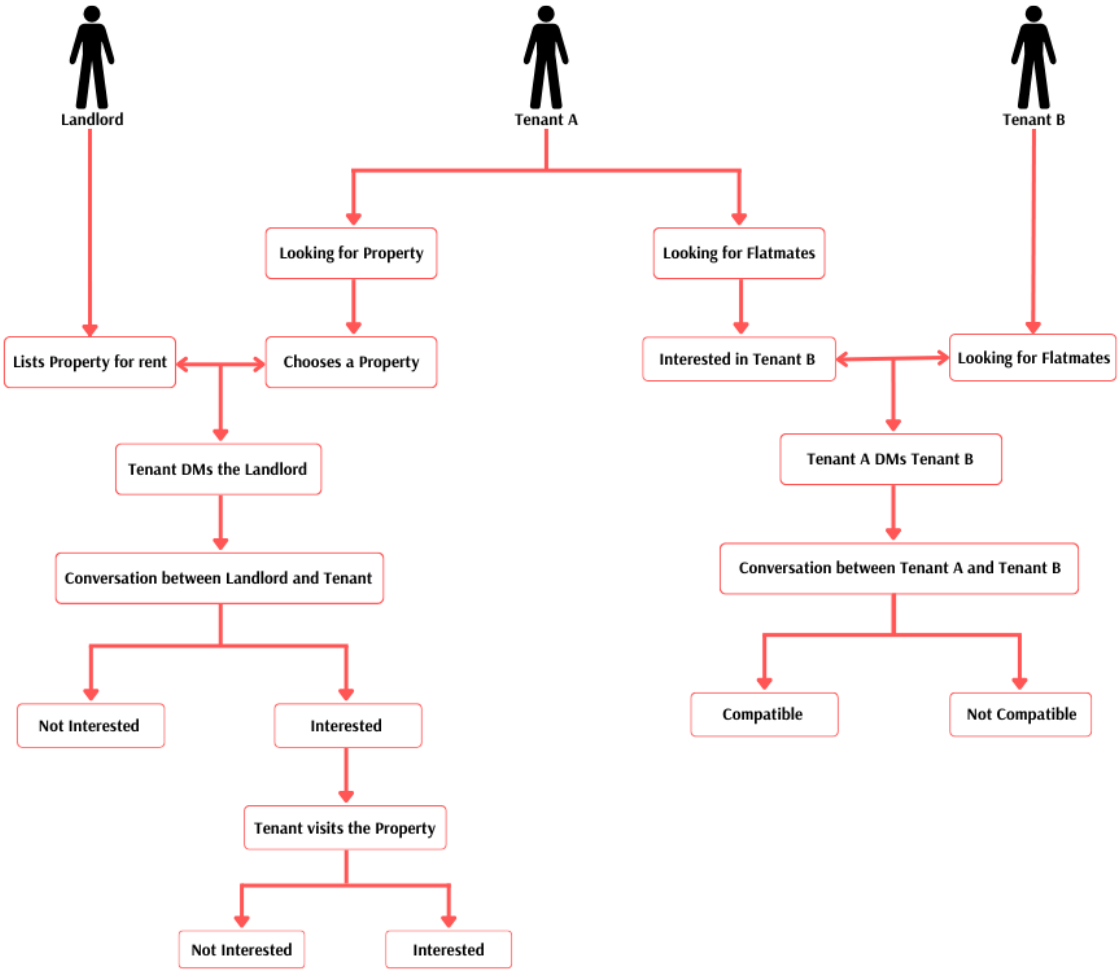
The purpose of this use case is to allow the landlord to add, edit, delist, or delete their properties. To add/edit a property, the landlord must have all the necessary property details available to them. To mark a property as 'Occupied', the property must be temporarily unavailable for rent or already occupied. The priority of this use case is high since one of the platform's main goals is to help tenants find a rental property of their choice. The user must have a valid account as a landlord on the platform to gain access to these operations. After adding a property, it will be added to the database and mapped with the respective landlord. After editing a property, it will be updated in the database. After delisting a property, it will be greyed out and inaccessible to anyone searching for it. After deleting a property, it will be removed from our database. The actors involved would be the users who are registered as landlords on the platform.

3.1.3 Use Case 3 (UC3): Enabling a user to search for a property/flatmate efficiently:



The purpose of this use case is to enable users (specifically the tenants) to search for and find any rentable property in the area of their choice or to find a flatmate to share a rented place. The user must be signed up and logged into the application to search for the property. This use should be a high priority since connecting tenants to rentable properties is one of the app's primary purposes. The user must be signed up and logged into the application to use this feature. If a user is interested in any property or in contacting a particular person as a prospective roommate, he can connect to the concerned persons through the direct messaging feature of the app. The actors in this use case are those using the application and seeking a rentable property or a flatmate/roommate.

3.1.4 Use Case 4 (UC4): Efficient Interaction between the two concerned parties:



The purpose of this use case is to facilitate interaction between tenant and landlord and between prospective flatmates. This is to ensure tenants and landlords can make informed choices and consider many factors before committing to a particular property or flatmate. This use case is of high priority since tenants and landlords cannot make the right choices without interacting with the opposite party if the system malfunctions.

This use case has two pre-conditions –
1. For Tenant-Landlord interaction: The landlord must upload their property correctly, and the tenant must actively search for properties.
2. For Flatmate-Flatmate interaction: Both the prospective flatmates should have completed filling in all their preferences in their profile and should be actively searching for flatmates.

The tenant can go on searching for new landlords or flatmates if he loses interest in the current property or flatmate. If both are compatible after the meeting, the deal can be finalized in the case of property, and the match-up can be confirmed in the case of flatmate.

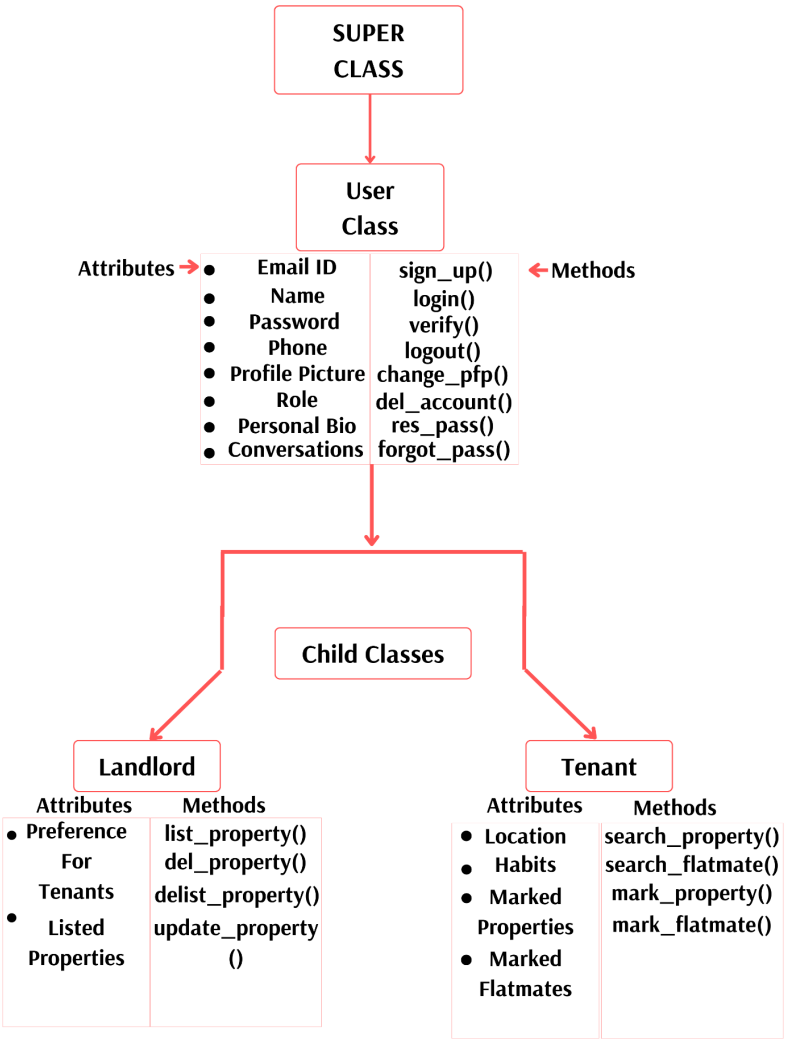
There are three actors in this use case:

1. Tenant A – Searching for property and flatmates
2. Landlord – Potential landlord of Tenant A
3. Tenant B – Potential flatmate for Tenant A

As an exception, if both parties are 100% confident that they are suitable for each other just based on available information without any interaction, then this use-case may not be required.

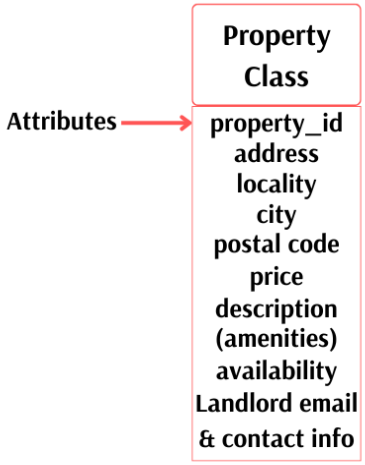
3.2 Class Diagrams

1. Class Diagram 1

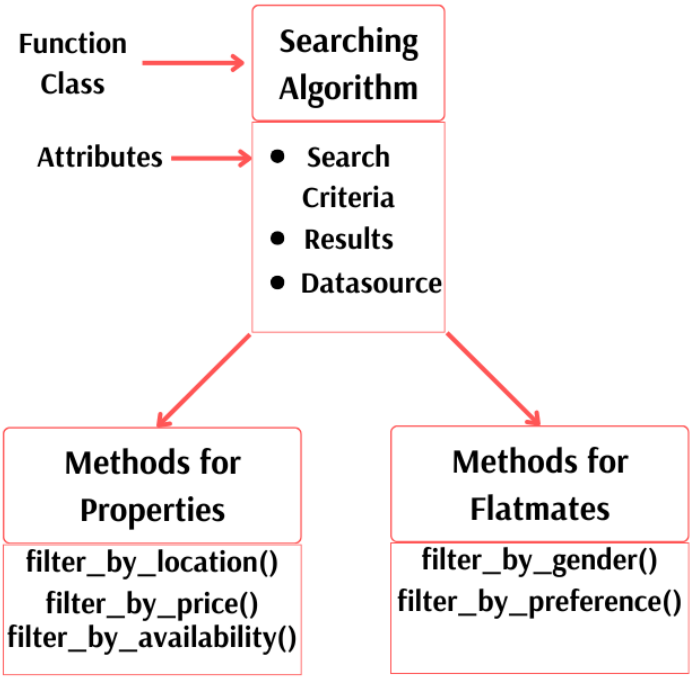


This diagram captures the high-level class description of the User classes which we expect to be a superclass, which is inherited by its two child classes, namely the Landlord and Tenant class, which are the two types of users we have in the platform.

2. Class Diagram 2

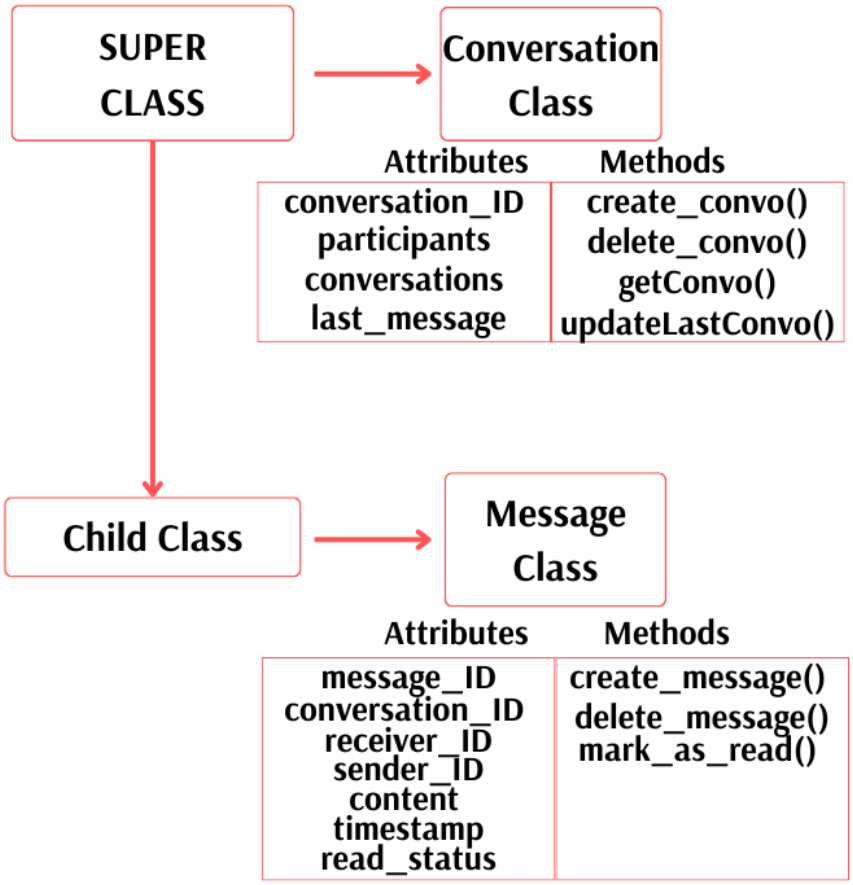


- The property class entails the details of the attributes associated with any property. We expect these details to be filled out by the landlord while listing the property on the platform.
- Complete details aligned with appropriate detailing about the amenities will make the property look more lucrative for any tenant.



This class diagram captures the high-level overview of the function class that we intend to implement, wherein we include the various algorithms related to the platform, namely, searching and filtering based on user preferences.

3. Class Diagram 3



This class diagram captures the conversation and messaging class that we intend to use to implement the direct messaging option.
The various attributes aim to provide the user with an easy method of connecting to other users.

Below, we provide a detailed overview of each class and the associated attributes and methods:

- ## User Class

Attributes

- Email ID – a string that stores the user's email ID.
- Name – a string that stores the name of the user.
- Phone – a string that stores the phone number of the user.
- Password – a string that stores the user’s password.
- Profile Picture – the profile pic of the user.
- Role – stores whether the user is a landlord or tenant.
- Personal Bio – a personal description of the user that he might like to add.
- Conversations – stores the list of all conversation IDs.

Methods

- sign_up() – registers a new user on the platform.
- Login () – logs the user into their account.
- Verify () – handles OTP verification for security.
- Logout () – logs the user out of their account.
- change_pfp() – updates the user's profile picture.
- del_account() – permanently deletes the user’s account.
- res_pass() – changes the user's password.
- forgot_pass() – resets the password via OTP verification.

- ## Landlord Class

Attributes

- Preference For Tenants – list of all the preference choices he would want in a tenant living in his properties. It would be a list of booleans.
- List of properties – list of all the properties the landlord has listed.

Methods

- list_prop() – called when a landlord wants to list a new property on the website, making it available for tenants to view and inquire about.
- delist_prop() – used when a landlord seeks to temporarily remove a property from listings, making it unavailable for tenants without permanently deleting it.
- del_prop() – permanently removes a property from the platform.
- update_prop() – allows a landlord to modify the details of an existing property listing, such as price, availability, or description.

● Tenant Class

Attributes

- Location – stores the location preferred by the tenant.
- Habits – A list containing the tenant's habits, like whether he smokes or has pets, etc.
- Looking for flatmates – stores whether the tenant is looking for flatmates.
- Marked properties – the list of properties the tenant has marked as interested.
- Marked flatmates – the list of flatmates the tenant has marked as interested.

Methods

- search_property() – finds available properties based on filters like location, price, etc.
- search_flatmate() – searches for potential flatmates based on preferences and compatibility.
- mark_property() – saves or marks a property as a favorite for future reference.
- mark_flatmate() – saves a flatmate profile for later consideration

● Property Class

Attributes

- property_id – An integer that stores a unique identifier for the property.
- Address – A string that stores the full address of the property.
- Locality – A string that stores the specific neighborhood or area of the property.
- City – A string that stores the name of the city where the property is situated.
- postal_code – A string that stores the ZIP or postal code of the property’s location.
- Price – A float that stores the rental cost of the property.
- Description (amenities) – A string that stores a brief overview of the property, including its features and amenities.Availability – A boolean that indicates whether the property is currently available for rent.

- landlord_email – A string that stores the landlord's contact email.
- landlord_contact – A string that stores the phone number or other contact details of the landlord.

● Conversation Class

Attributes

- participants – A list of integers that stores the user IDs of both the participants in the conversation.
- Conversations – A list of message objects that stores all messages exchanged in the conversation.
- last_message – A message object that stores the most recent message sent in the conversation.

Methods

- create_convo() – initializes a new conversation between the users and adds them as participants.
- delete_convo() – removes a conversation and all its messages.
- getConvo() – retrieves the details of a specific conversation, including participants and message history.
- update_last_convo() – updates the last message in the conversation when a new message is sent.

● Message Class

Attributes

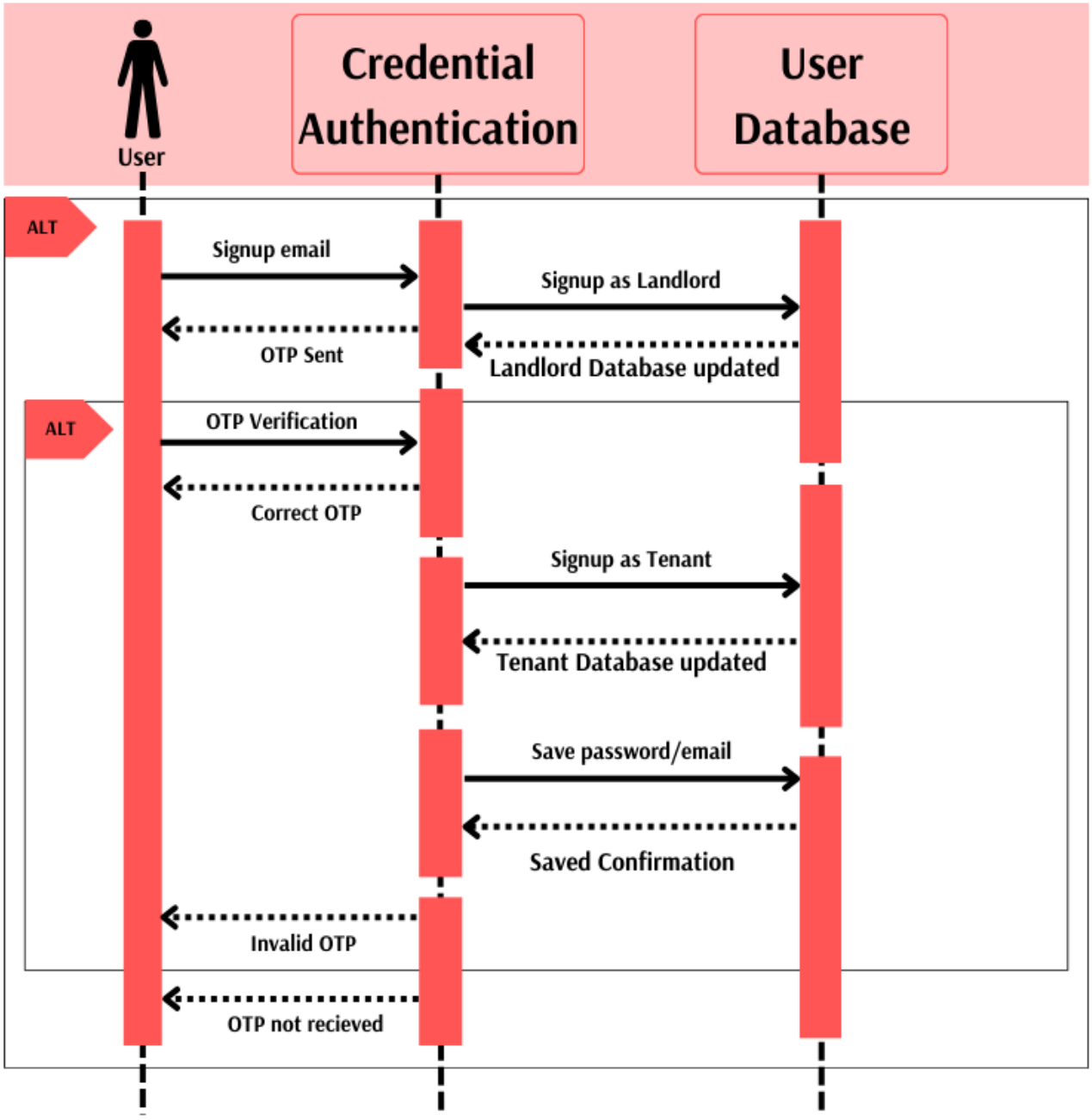
- message_ID – stores a unique identifier for each message.
- conversation_ID – links the message to a specific conversation.
- receiver_ID – An integer that stores the ID of the user receiving the message.
- sender_ID – An integer that stores the ID of the user sending the message.
- Content – A string that stores the text content of the message.
- Timestamp – stores the exact time when the message was sent.
- read_status – A boolean that indicates whether the message has been read by the recipient.

Methods

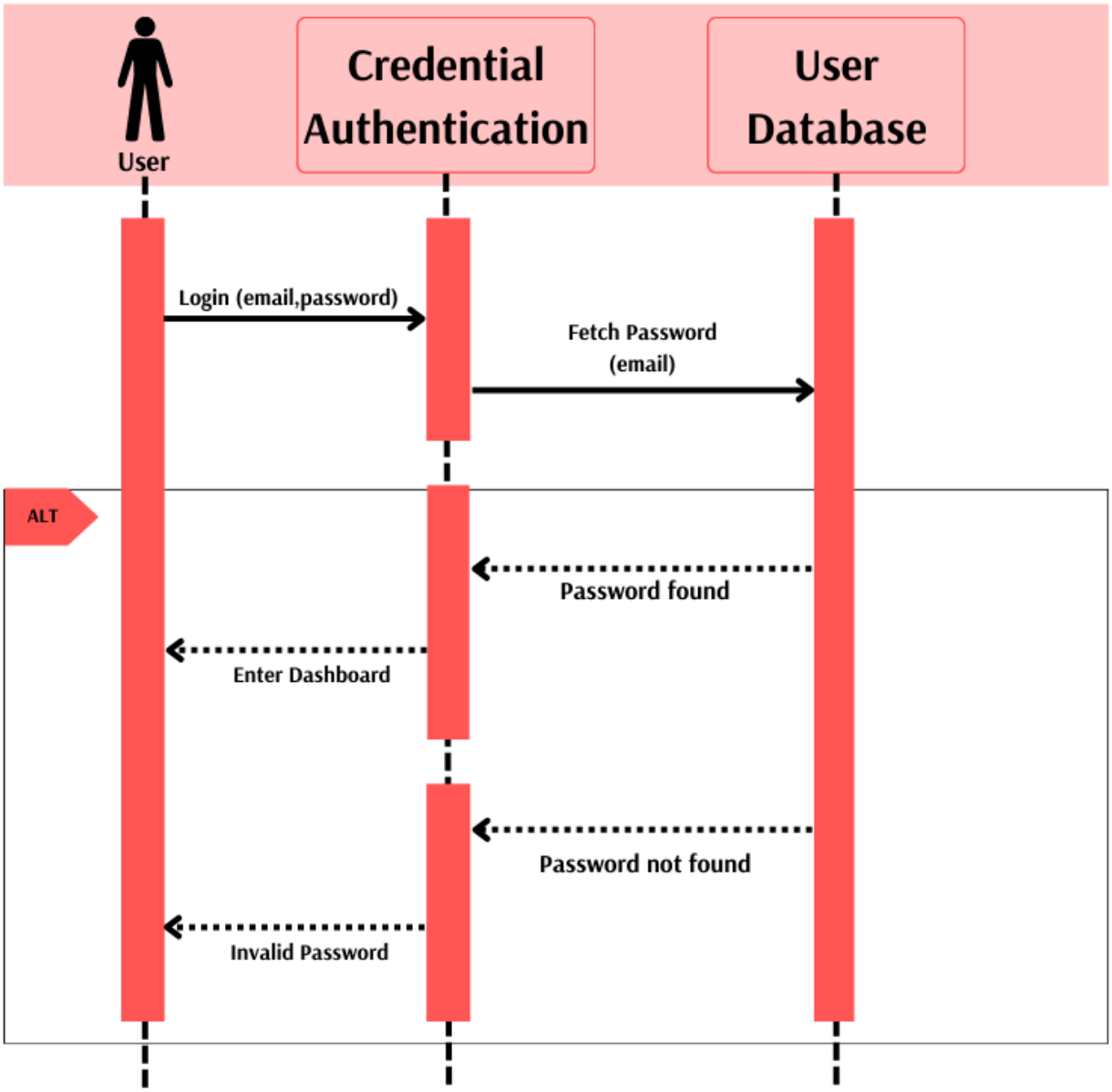
- create_message() – creates a new message, links it to a conversation, and sets the sender, receiver, content, and timestamp.
- delete_message() – removes a message from the conversation.
- mark_as_read() – marks the message as read, updating the read_status to true

3.3 Sequence Diagrams

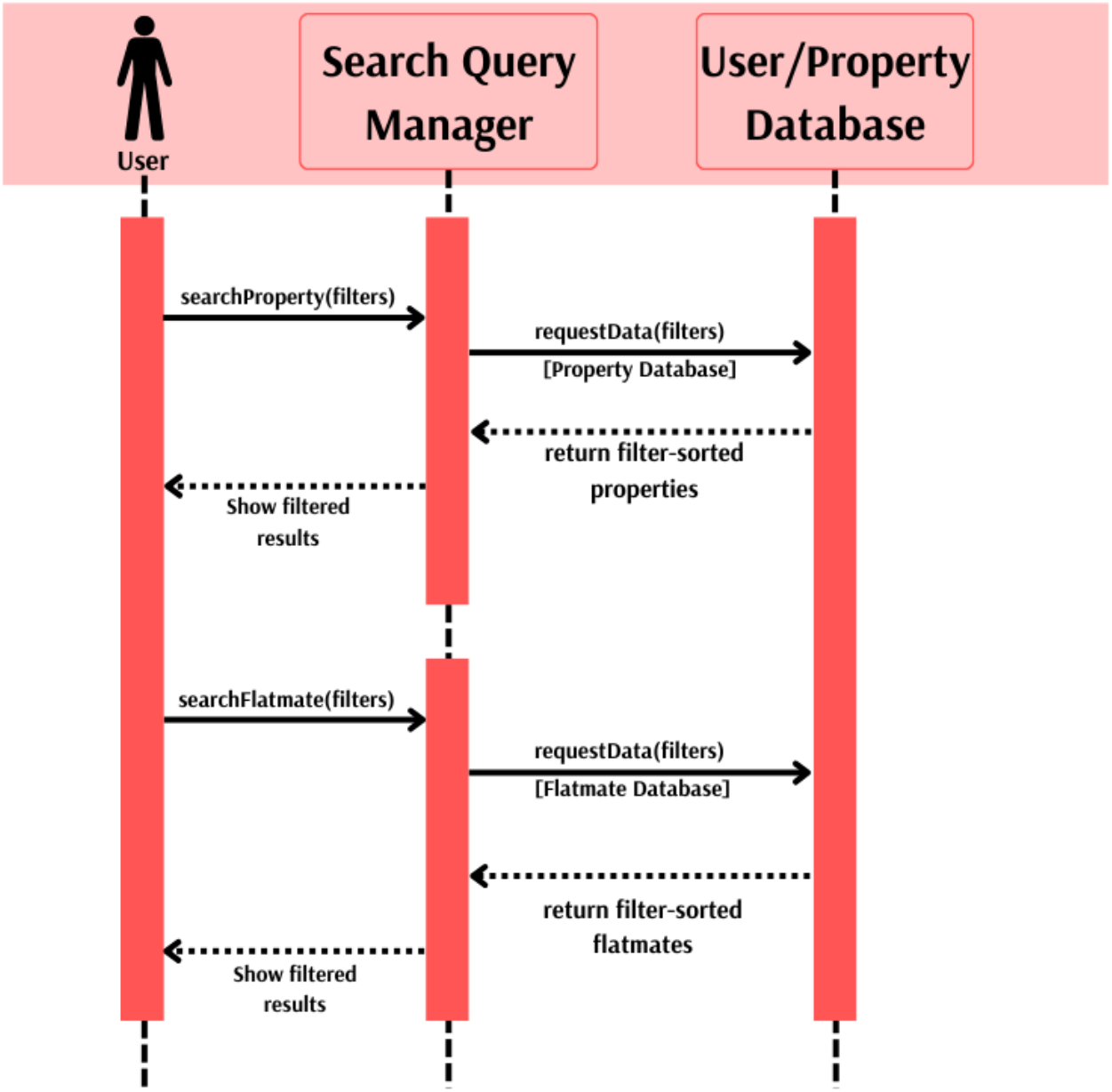
3.3.1 Signup Page:



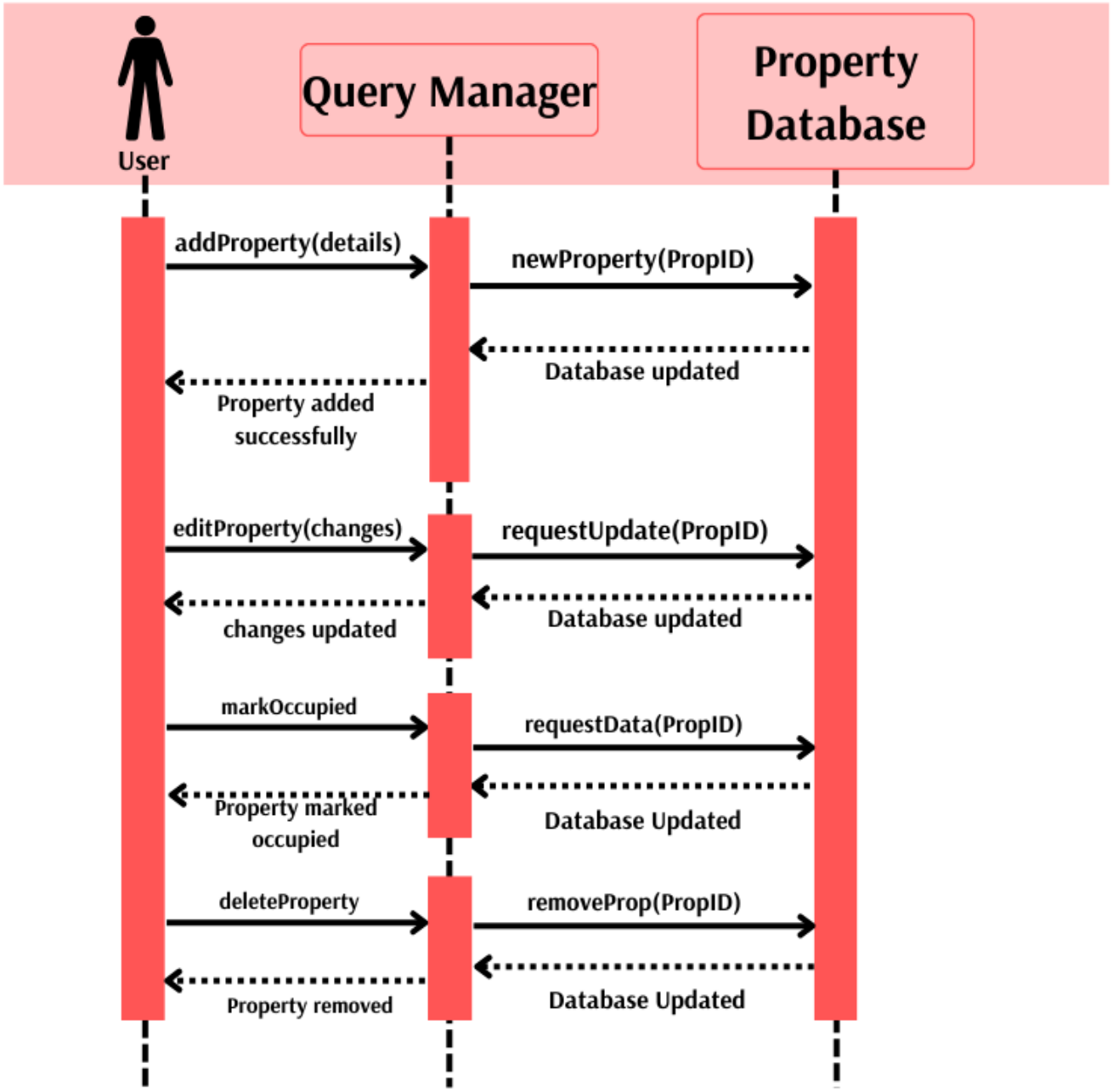
3.3.2 Login Page:



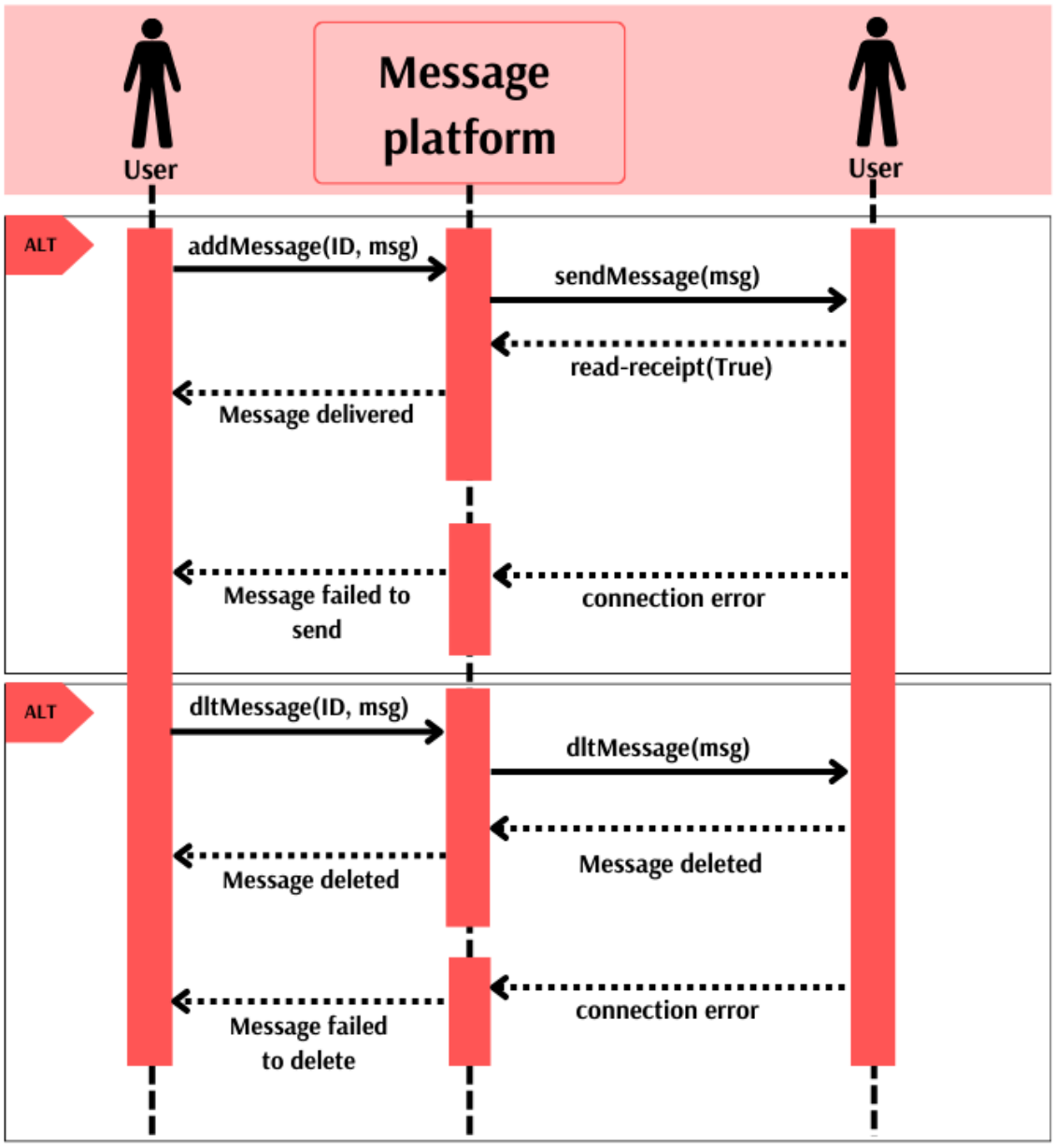
3.3.3 Tenant Dashboard:



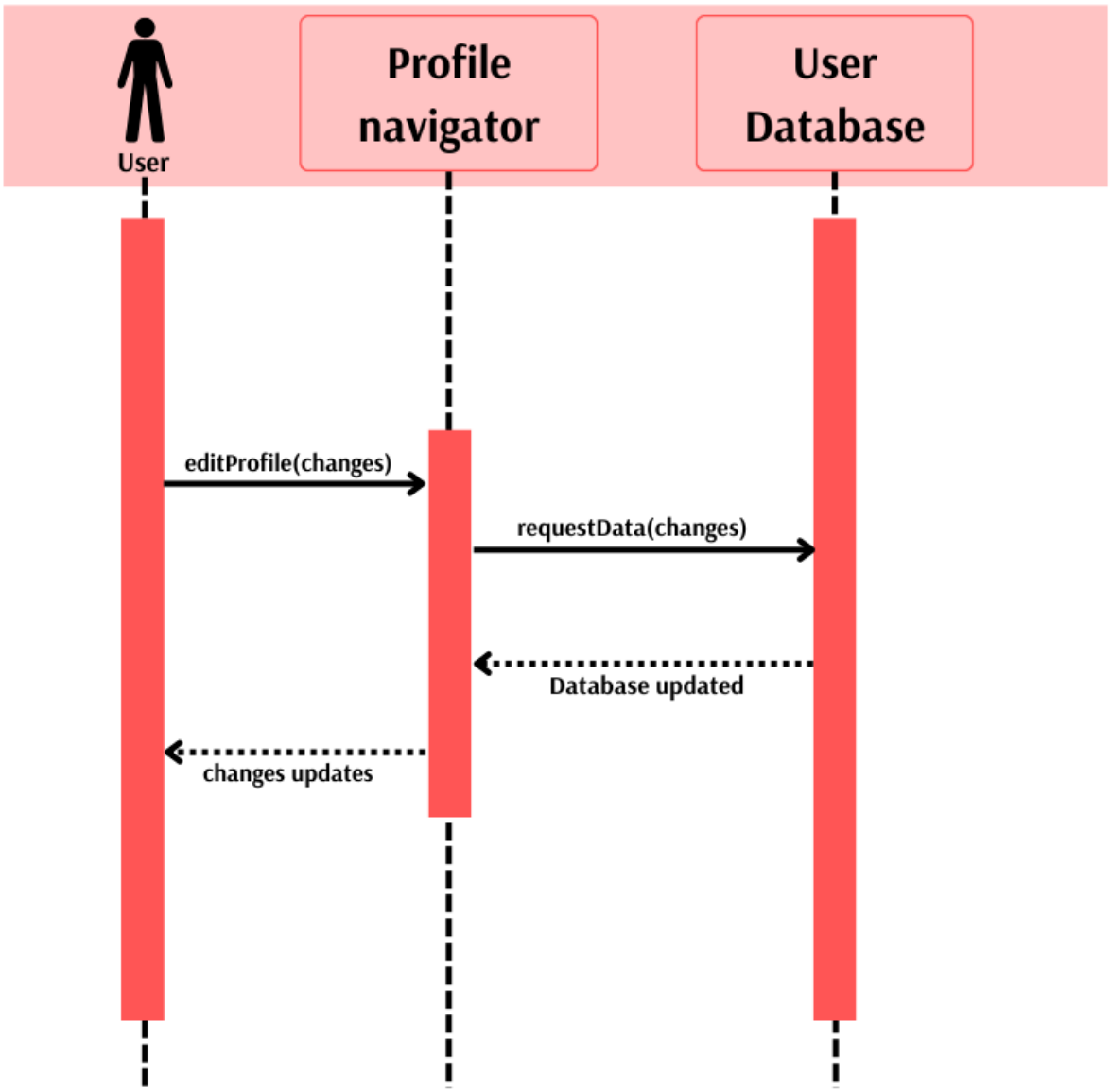
3.3.4 Landlord Dashboard:



3.3.5 Message Users:

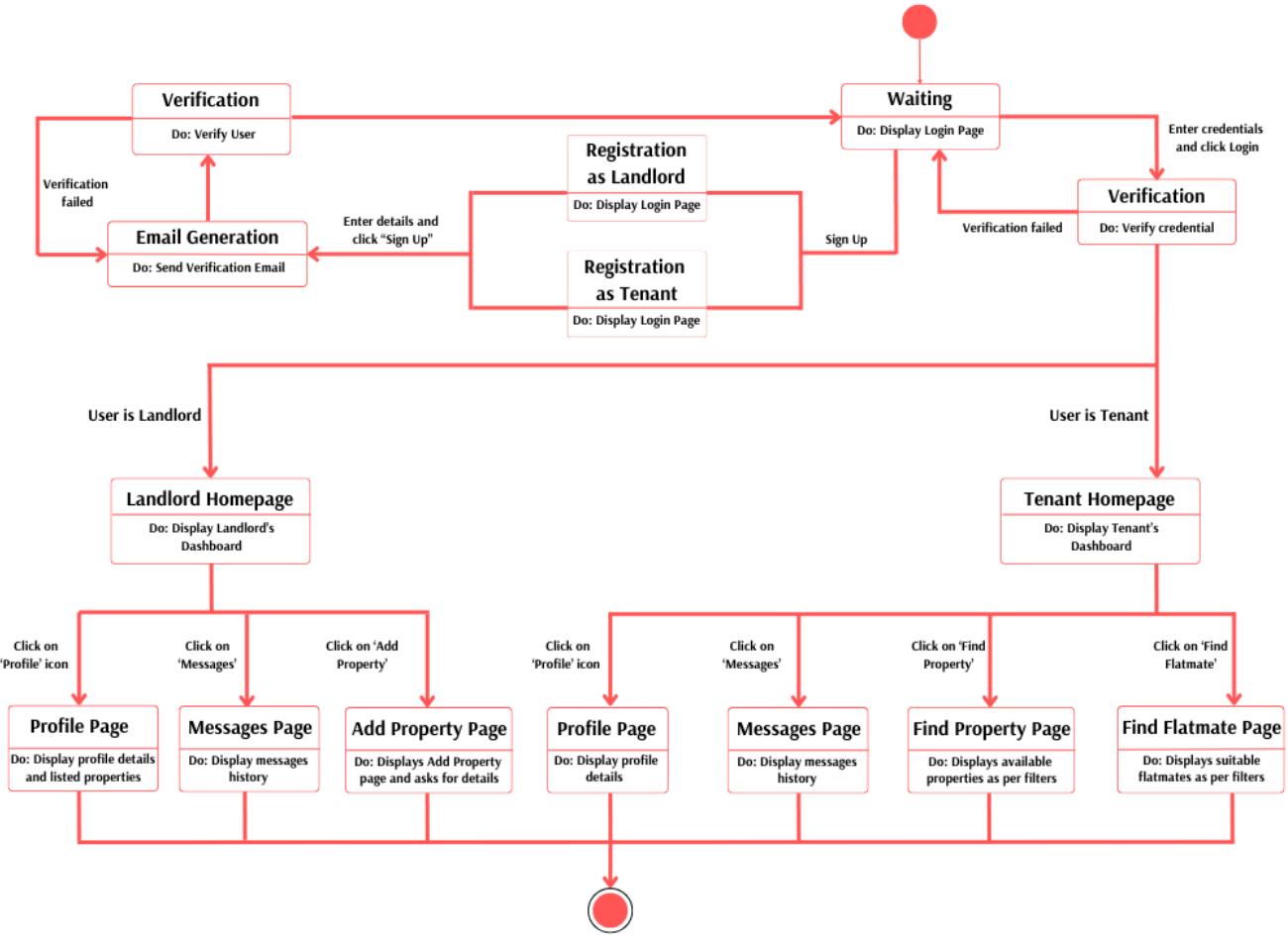


3.3.6 Edit Profile:

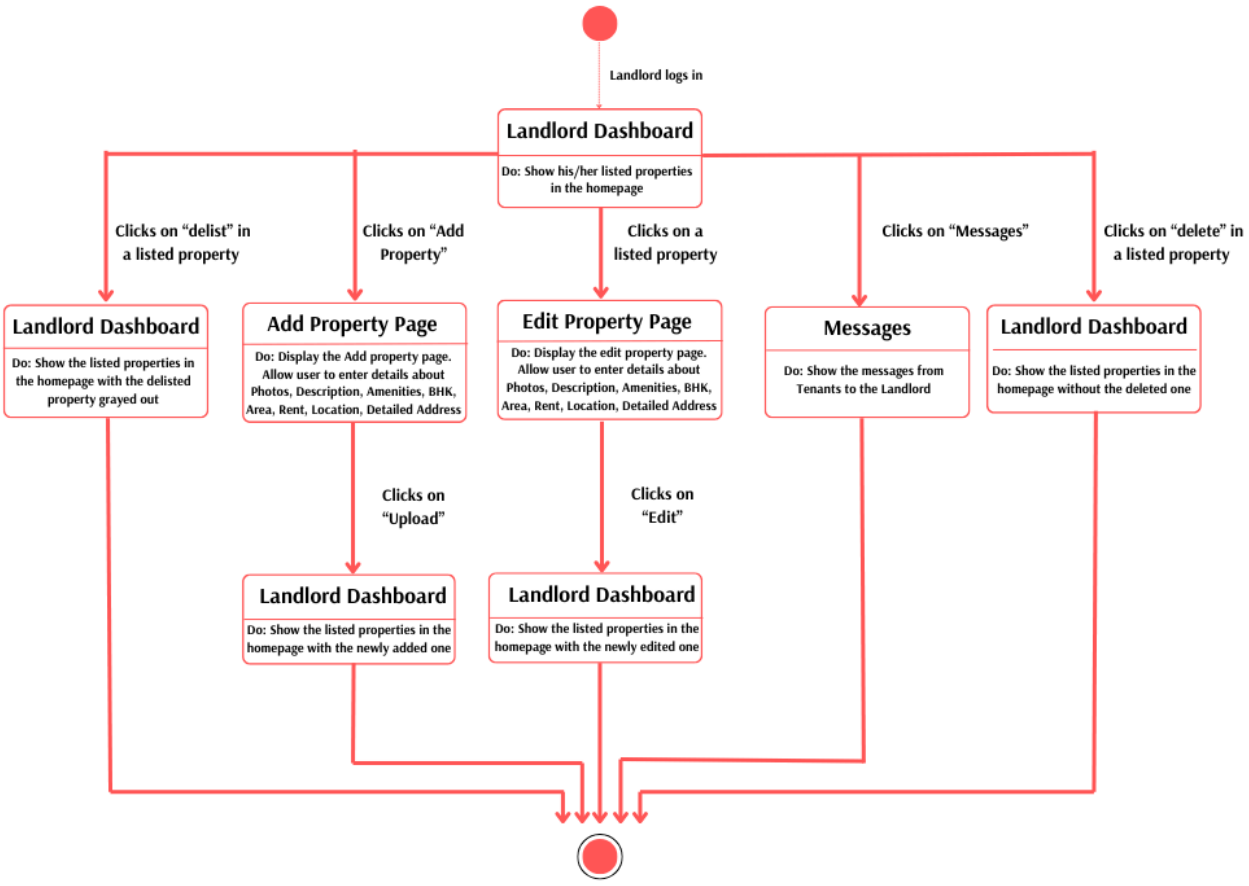


3.4 State Diagrams

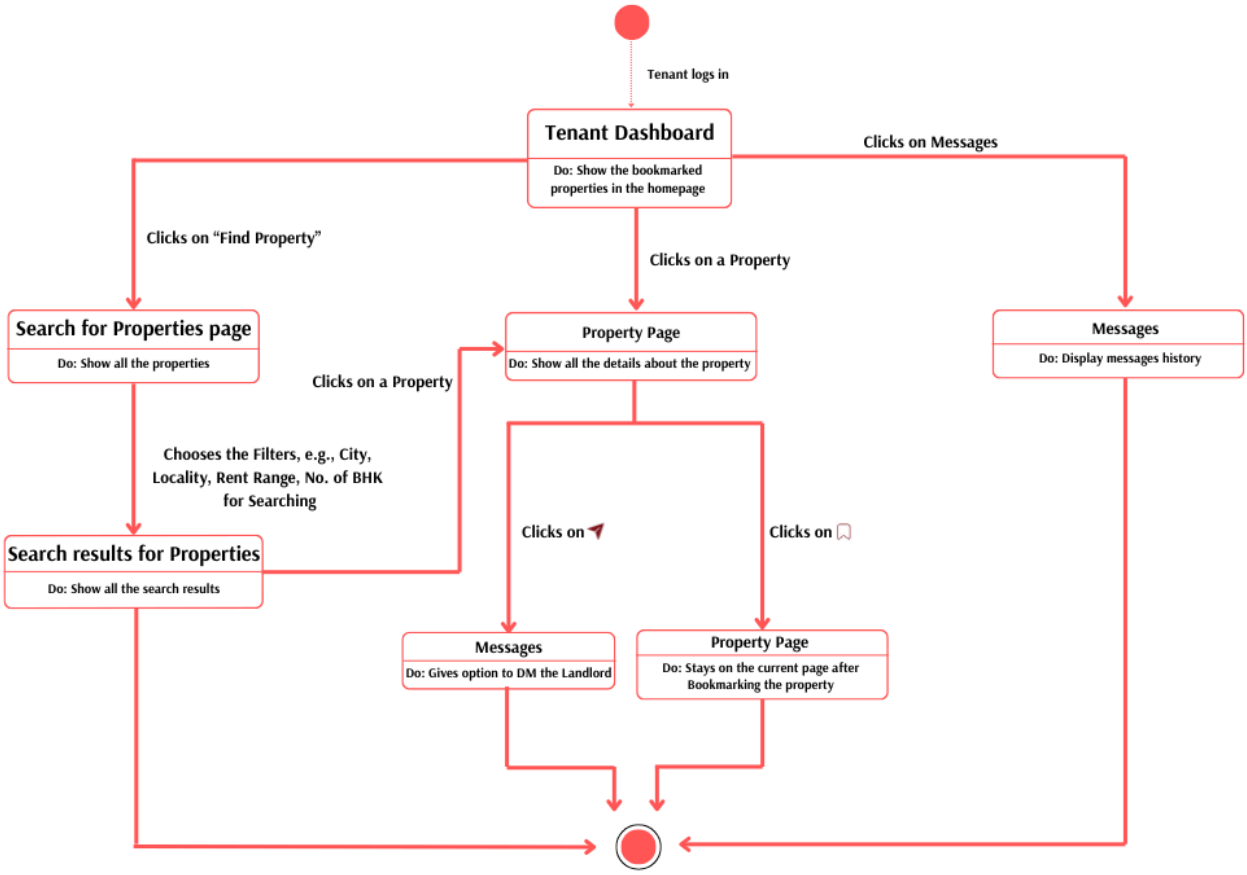
3.4.1 Login and Dashboard



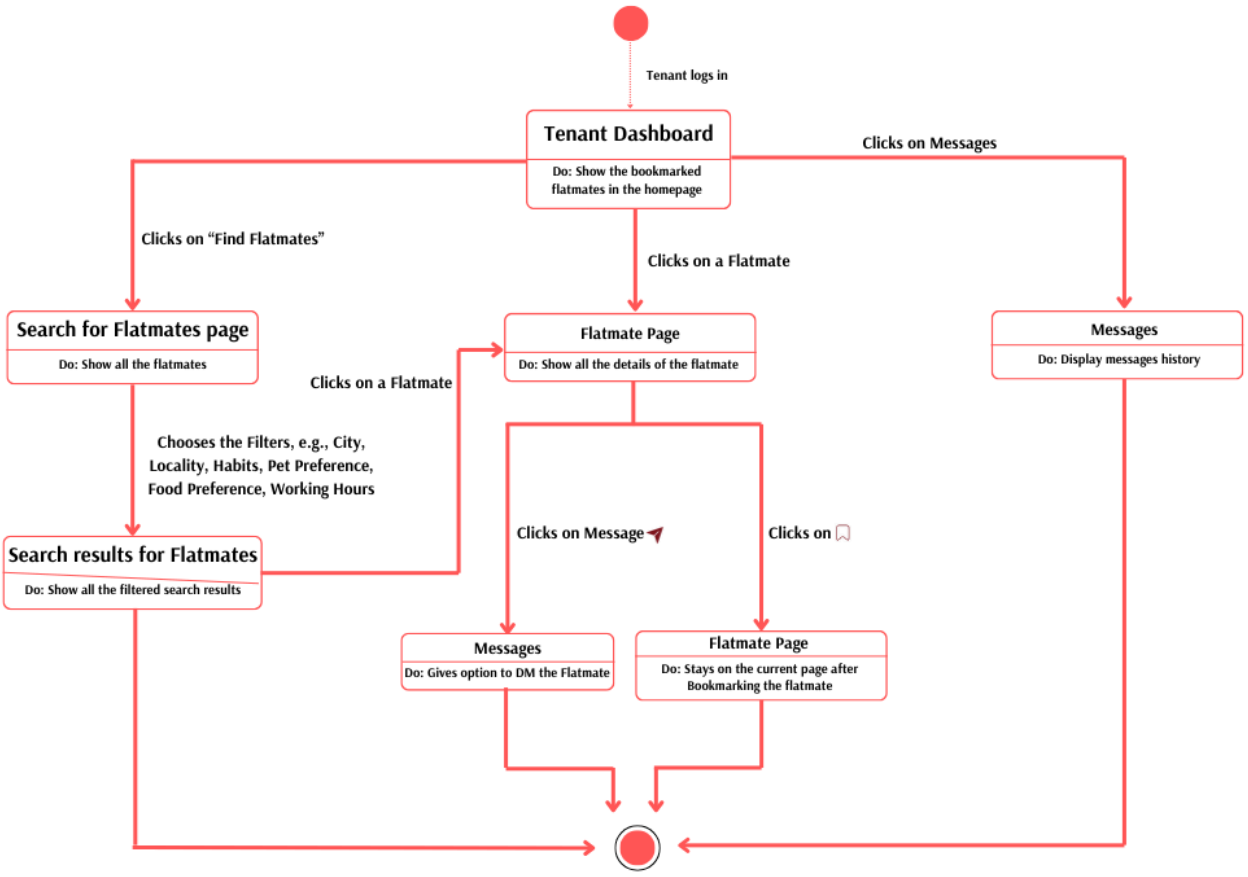
3.4.2 Property Listing, delisting, and deleting as a Landlord



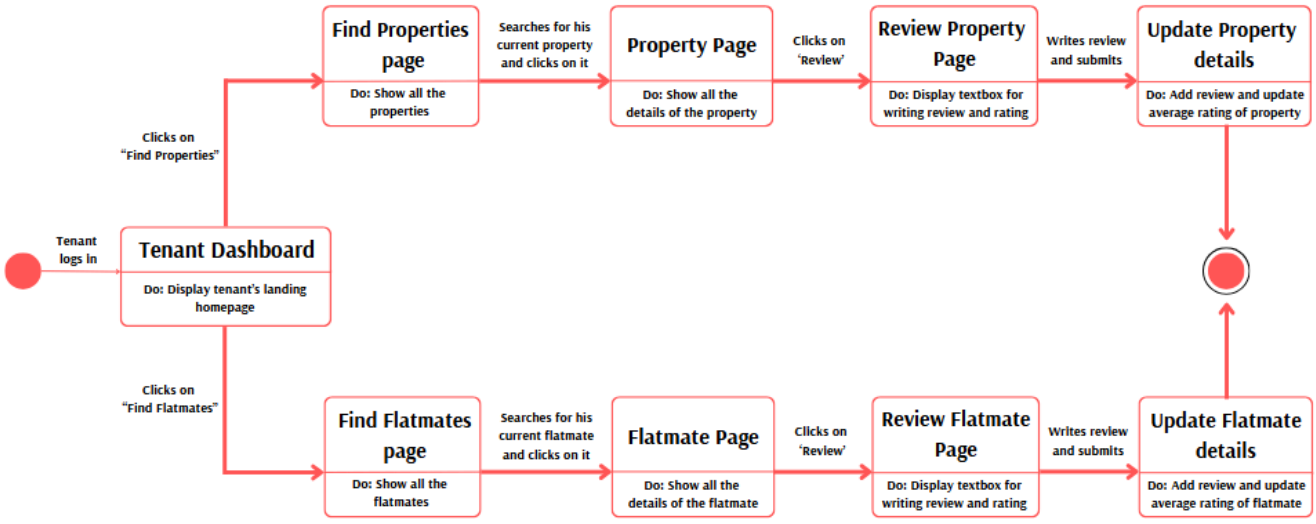
3.4.3 Searching for Property as a Tenant



3.4.4 Searching for Flatmates as a Tenant



3.4.5 Review System

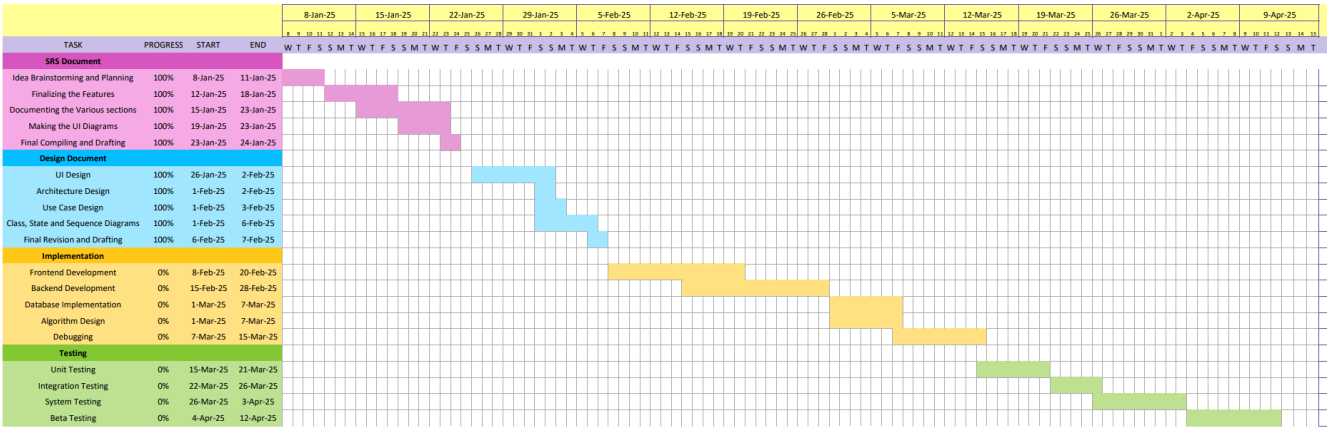


4 Project Plan

Project Roomble

START DATE: 08/01/2025

The following Gantt chart covers our current progress:



TASK	PROGRESS	START	END
SRS Document			
Idea Brainstorming and Planning	100%	8-Jan-25	11-Jan-25
Finalizing the Features	100%	12-Jan-25	18-Jan-25
Documenting the Various sections	100%	15-Jan-25	23-Jan-25
Making the UI Diagrams	100%	19-Jan-25	23-Jan-25
Final Compiling and Drafting	100%	23-Jan-25	24-Jan-25
Design Document			
UI Design	100%	26-Jan-25	2-Feb-25
Architecture Design	100%	1-Feb-25	2-Feb-25
Use Case Design	100%	1-Feb-25	3-Feb-25
Class, State and Sequence Diagrams	100%	1-Feb-25	6-Feb-25
Final Revision and Drafting	100%	6-Feb-25	7-Feb-25
Implementation			
Frontend Development	0%	8-Feb-25	20-Feb-25
Backend Development	0%	15-Feb-25	28-Feb-25
Database Implementation	0%	1-Mar-25	7-Mar-25
Algorithm Design	0%	1-Mar-25	7-Mar-25
Debugging	0%	7-Mar-25	15-Mar-25
Testing			
Unit Testing	0%	15-Mar-25	21-Mar-25
Integration Testing	0%	22-Mar-25	26-Mar-25
System Testing	0%	26-Mar-25	3-Apr-25
Beta Testing	0%	4-Apr-25	12-Apr-25

The following is the current work division that the team intends to follow:

MEMBERS	TASK
AARSH JAIN	FRONTEND DEVELOPMENT, SYSTEM TESTING, ADDRESSING FEEDBACK
ARITRA AMBUDH DUTTA	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, ALPHA TESTING
ARITRA RAY	BACKEND DEVELOPMENT, INTEGRATION TESTING, UI/UX DESIGN
BHUKYA VAISHNAVI	FRONTEND DEVELOPMENT, UI/UX DESIGN, TESTING
BIKRAMJEET SINGH	BACKEND DEVELOPMENT, SYSTEM TESTING, ADDRESSING FEEDBACK
HITARTH MAKAWANA	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, UNIT TESTING
SHLOK JAIN	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, UI/UX DESIGN
RONAV PURI	FULL STACK IMPLEMENTATION, ADDRESSING FEEDBACK, BETA TESTING
RATHOD AYUSHI	FRONTEND DEVELOPMENT, UNIT TESTING, UI/UX DESIGN
SAKSHAM VERMA	BACKEND DEVELOPMENT, CODE IMPROVEMENT, UI/UX DESIGN
SUREPALLY PRANAYSRIHARSHA	FRONTEND DEVELOPMENT, SYSTEM TESTING, ADDRESSING FEEDBACK

Appendix A - Group Log

S. No.	DATE	TIMINGS	VENUE	DESCRIPTION
1.	25-01-2025	19:00-21:00	RM BUILDING	Discussed the UI design and architecture and divided the initial work among the team members.
2.	30-01-2025	20:00-23:00	RM BUILDING	Discussed the document's progress and finalized the various classes, designs, and overview of the algorithms to be used.
3.	03-02-2025	17:30 - 20:00	GOOGLE MEET	Concluded the work on UI, state, and sequence diagrams and validated the progress of the document.
4.	06-02-2025	21:00-22:00	GOOGLE MEET	Finalized all the aspects of the document.
5.	06-02-2025	23:00-23:30	GOOGLE MEET WITH TA	We discussed the progress of the design document with our TA, Mr. Nij, and clarified some doubts regarding it.