TSE2101
Final Report

For

HomeWoW System

Version 3.0

Tutorial Section: TT9L
Group No.: 4

SADMAN ZULFIQUER	**1221301874**
TAN TENG HUI	**1211102289**
TENG WEI JOE	**1211102797**
HO TECK FUNG	**1211102399**

Date: 12/02/2024

Contents

Revisions

Version	Primary Author	Description of version	Date Completed
2.0	Teck Fung & Joe	- Updated Interface Design	5/02/2024
2.0	Wei Joe	- Added a new case for admin. Update Property Information	11/02/2024
2.0	Sadman	Data Dictionary - Improved	12/02/2024

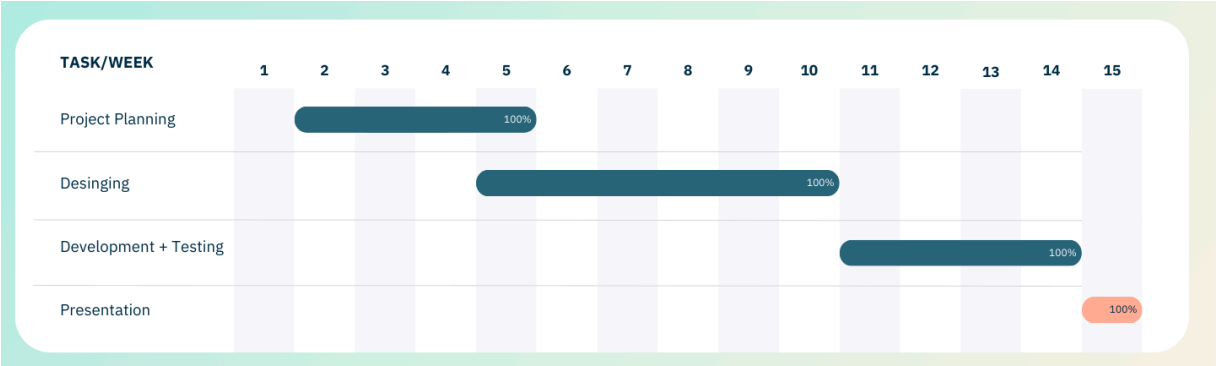
1 Project Management

1.1 Team Members

Name	Actor
SADMAN ZULFIQUER	**Owner/Agent**
TAN TENG HUI	**Tenant**
TENG WEI JOE	**Admin**
HO TECK FUNG	**Searcher & Admin**

1.2 Project Plan

The project is divided into four phases: Project Planning, Designing, Development + Testing, and Presentation. The Project Planning, Designing, and Development + Testing phases are completed. The Presentation phase is scheduled to start in week 15 and is expected to be completed by week 15.



2 System Overview

2.1 Description

The HomeWoW web app is a comprehensive property management platform designed to enhance the user experience through a robust verification system. Ensuring the integrity of the rental process, the system implements a thorough verification mechanism, confirming the legitimacy of property owners and agents by validating their submitted property documents. This feature instills trust and confidence in users seeking rental properties. The platform caters to diverse rental needs by accommodating both short and long-term tenancies, offering flexibility to users based on their preferences. A key functionality of the system is the provision of a seamless Tenancy Agreement feature, allowing property owners and tenants to create, review, and sign agreements efficiently. This makes a more efficient rental process, minimizing paperwork and ensuring a secure promised foundation for both parties. Additionally, the system prioritizes transparency by providing accurate and up-to-date market prices, empowering users to make informed decisions. Altogether this revolutionizes the property rental landscape by prioritizing security, flexibility, efficiency, and transparency for property owners, agents, and tenants.

2.2 Actors

Actor	Use Cases
Owner/Agent	1. Login 2. Sign Up 3. Edit Profile 4. Change Password 5. Add Property 6. Remove Property 7. Update Property Information 8. Upload Property Documents 9. View Property 10. Updates from Payment API 11. Agreement Request 12. Contact Support 13. Dashboard 14. Appointment 15. Logout

Actor	**Use Cases**
Tenant	1. Sign Up 2. Log In 3. Edit Profile 4. Change Password 5. Search Properties 6. View property 7. Make an Appointment 8. Sent Agreement request 9. View Outstanding Balance 10. Pay bill 11. View Invoices 12. Dashboard 13. Contact Support 14. View Agreement 15. View Receipt 16. Logout

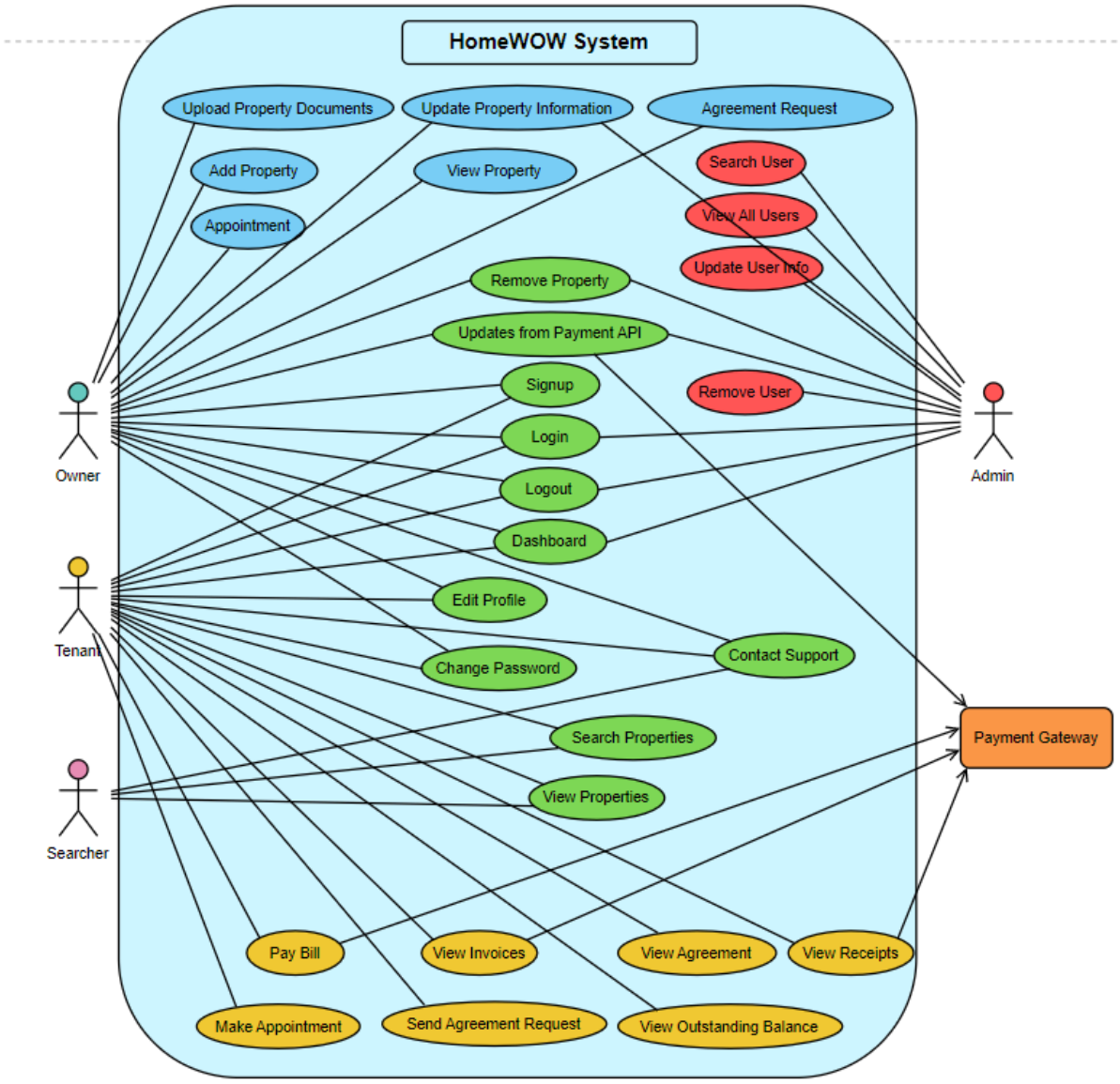
Actor	**Use Cases**
Admin	1. Login 2. Dashboard 3. Updates from Payment API 4. View All Users (Tenant & Owner/Agent) 5. Search User (Tenant & Owner/Agent) 6. Update User Info (Tenant & Owner/Agent) 7. Remove User 8. Remove Property 9. Contact Support 10. Logout 11. Update Property Information

Actor	**Use Cases**
Searcher (User is not Logged In the System)	1. View Properties 2. Search Properties 3. Contact Support

2.3 Assumptions & Dependencies

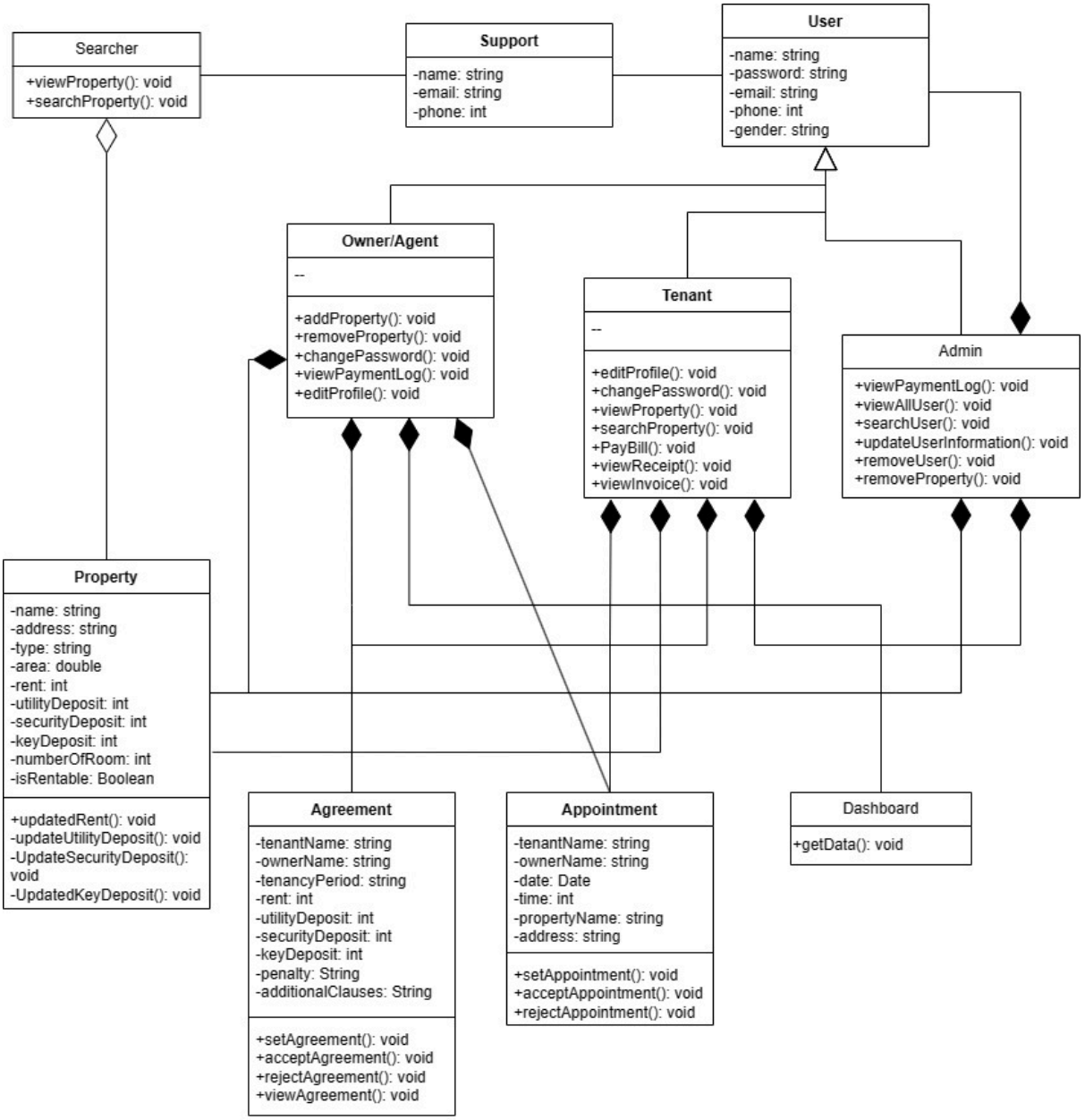
- All property listings submitted by owners or agents will be successfully added to the platform without technical issues.
- The platform is designed for use exclusively within Malaysia, with all users (owners, agents, and tenants) located within the country.
- The assumption is that all property documents submitted by owners or agents are legally valid and compliant with local regulations.
- It is assumed that all verification procedures are done instantly.
- The system assumes the payment gateway works without any technical failures.
- It is assumed that the customer support system is responsive and capable of promptly addressing user inquiries and issues.
- Owners or agents are expected to consistently update the availability status of their properties based on occupancy and other factors.
- The system depends on a single database to store and manage all relevant data.
- The system assumes that the database has sufficient storage capacity to accommodate the expected volume of property listings and user data.
- The system is assumed to work when accessed simultaneously by all actors.
- It is assumed that there are already a few pre-existing users & properties in the system.

2.4 Use Case Diagram



3 Requirements

3.1 Design Class Diagram



Class	Description
User	The User class serves as the superclass, encapsulating common attributes and behaviors shared by all users within the system. This includes essential information such as username, email, and contact details. It forms the foundation for more specialized user types.
Tenant	The Tenant subclass extends the User class and represents individuals seeking to rent properties. Tenants have

	attributes associated with their tenancy, including Agreement class through composition. They utilize the system to search for and apply to rent properties listed by OwnerAgents.
OwnerAgent	Extending the User class, the OwnerAgent subclass represents property owners or agents. OwnerAgents not only manage their profiles but also utilize the Agreement class through composition. This composition implies that OwnerAgents have an association with Agreement instances, allowing them to manage agreements with Tenants.
Admin	Admins, extending the User class, have elevated privileges for system management. They remove user accounts, properties in the system, etc.
Property	The Property class encapsulates essential attributes and functionalities related to properties within the system. This class includes details such as property type, location, size, amenities, and rental terms. Owners or agents (OwnerAgents) utilize instances of this class to create and manage property listings, providing a comprehensive view for potential tenants. The class facilitates essential operations like property addition, editing, and status updates. Through composition, instances of the Property class are associated with OwnerAgent, Tenant & Admin class.
Agreement	The Agreement class is composed of both OwnerAgent and Tenant subclasses. It encapsulates the details of agreements between property owners (OwnerAgents) and tenants. This class includes attributes such as terms, conditions, and duration, facilitating an organized approach to managing agreements within the system.
Dashboard	The Dashboard class is designed to enhance user experience. Implemented by OwnerAgents, Tenants, and Admins. It offers valuable insights and functionalities tailored to each user's role within the system.
Support	The Support class establishes a vital association with the User class, facilitating efficient communication and issue resolution within the system. It serves as the backbone for user assistance, ensuring that queries, concerns, and concerns are effectively addressed. Users, including OwnerAgents, Tenants, and Admins, can leverage the Support class to seek help, report problems, and receive timely responses from the support team.
Searcher	The Searcher class is a specialized user class designed for individuals who focus on exploring available properties within the system. By utilizing an aggregation relationship with the Property class, it is also associated with the

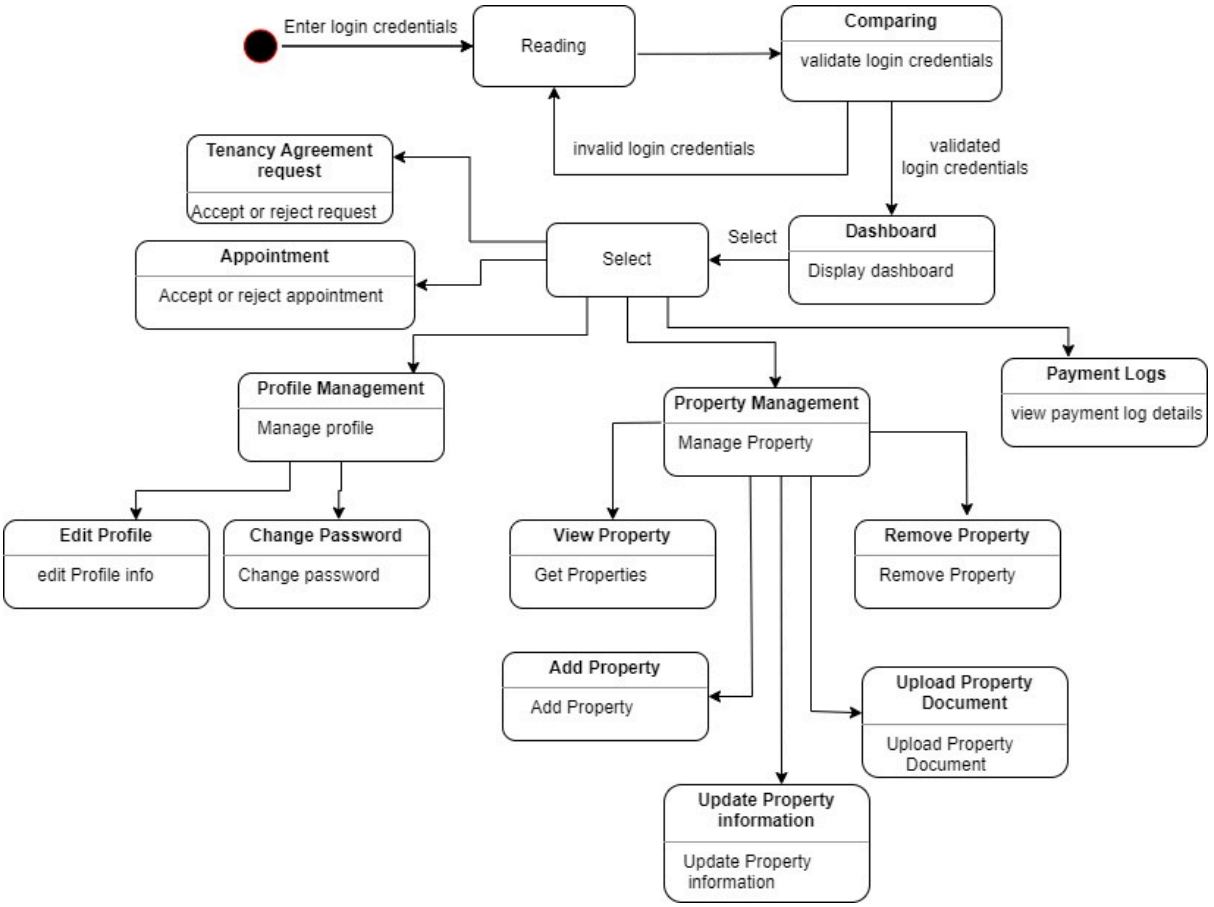
	Support class, providing users with the capability to easily contact support for assistance, report issues, or seek clarification during their property exploration journey.
Appointment	The Appointment class represents scheduled appointments between tenants and property owners (OwnerAgents). It encapsulates the details of each appointment, including the tenant involved, the corresponding property, and the scheduled date. This class facilitates an organized approach to managing appointments within the system.

3.2 State Diagrams

3.2.1 Owner/Agent

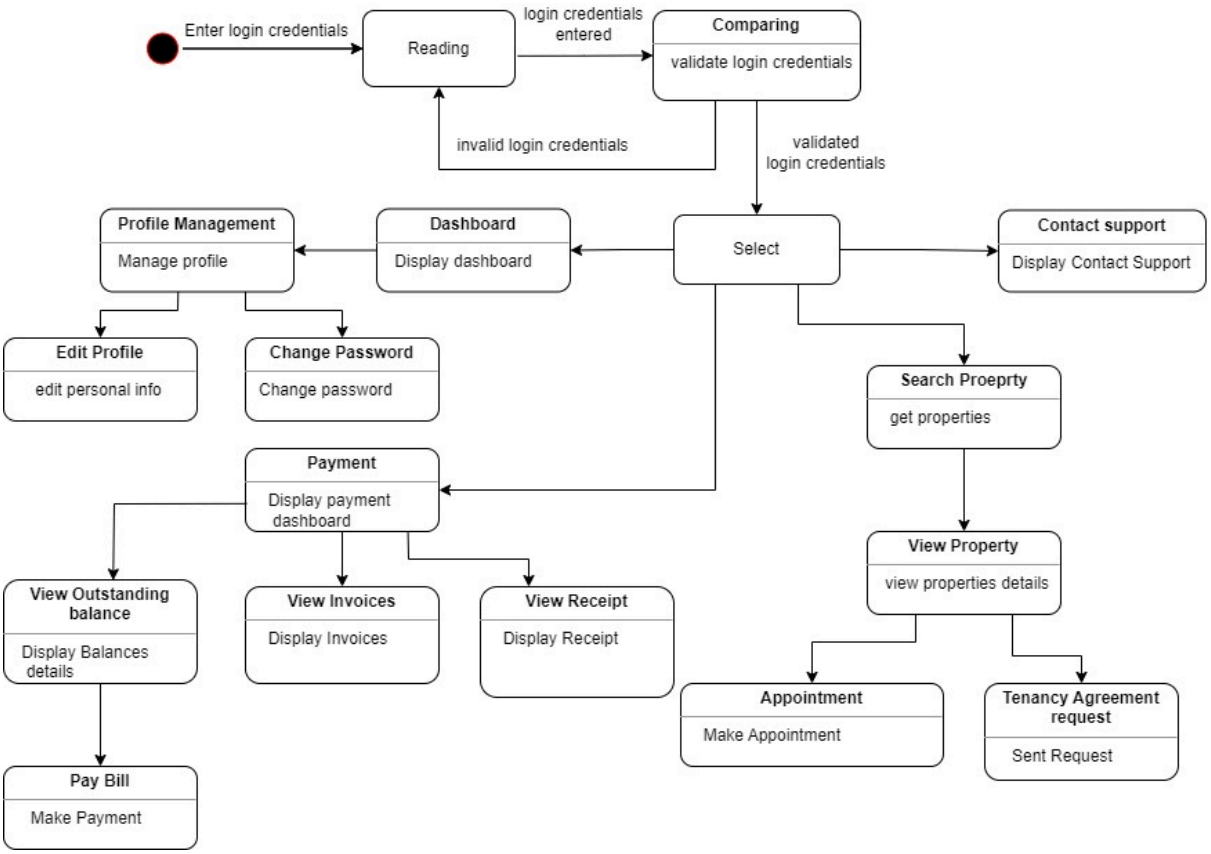
The initial interaction within the system commences with the login process, where the system reads and verifies the entered user data. Upon successful authentication, users gain access to the Dashboard. Owners can opt to review tenancy agreement requests, providing the flexibility to either accept or reject these requests. Furthermore, owners/agent have the ability to manage appointments, allowing them to accept or reject scheduled property viewings. In addition, owners can navigate to the Profile Management section for editing personal information and changing passwords, ensuring account security and relevance. To track financial transactions, owners can access Payment Logs, providing a detailed overview of payment histories. Lastly, the Property Management section offers a comprehensive set of functionalities, allowing owners to view property details, add new properties, remove

property, upload essential property documents, and update property information.



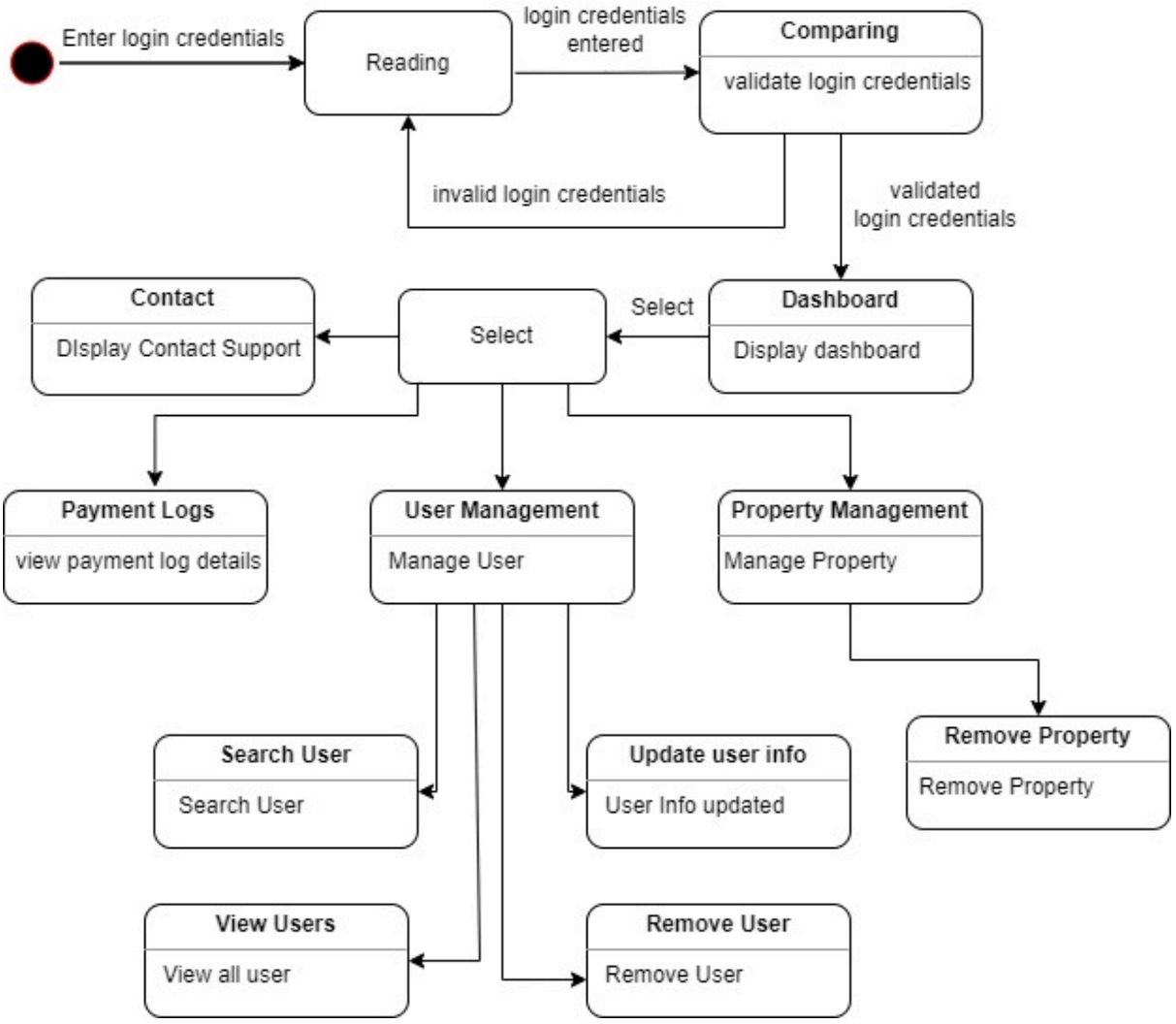
3.2.2 Tenant

The initial interaction within the system commences with the login process, the system reads and verifies user-entered data. Upon successful verification, granting access to the main use case, tenants are presented with a myriad of choices. Within the dashboard, tenants can seamlessly navigate to Profile Management, allowing them to edit their profiles and enhance account security by changing passwords. Tenants have the option to access the Payment section, where they can effortlessly view outstanding balances, view invoices, view receipts, and pay bills through the payment gateway. To explore available properties, tenants can initiate a search and delve into the details of properties, followed by the ability to schedule appointments with property owners or agents. Additionally, tenants can streamline the rental process by sending tenancy agreement requests. For any assistance or queries, tenants can opt to contact the support team through the Contact Support option.



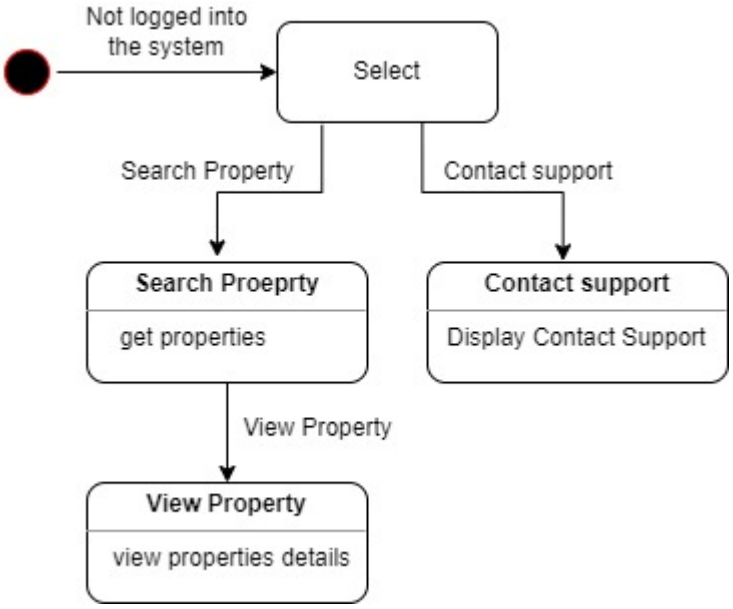
3.2.3 Admin

The initial interaction within the system commences with the login process, the system meticulously reads and verifies user-entered data. Upon successful verification, granting access to the dashboard, administrators are prompted to select their next action.
Within the dashboard, administrators can seamlessly choose Property Management, allowing them to efficiently remove properties. Alternatively, they have the option to delve into User Management, where they can search for users, view a comprehensive list of all users, remove users, and update user information as needed. To gain insights into financial transactions, administrators can access the Payment Logs section, enabling them to peruse detailed payment log information. Lastly, administrators can opt to utilize the Contact Support feature to contact the support team.



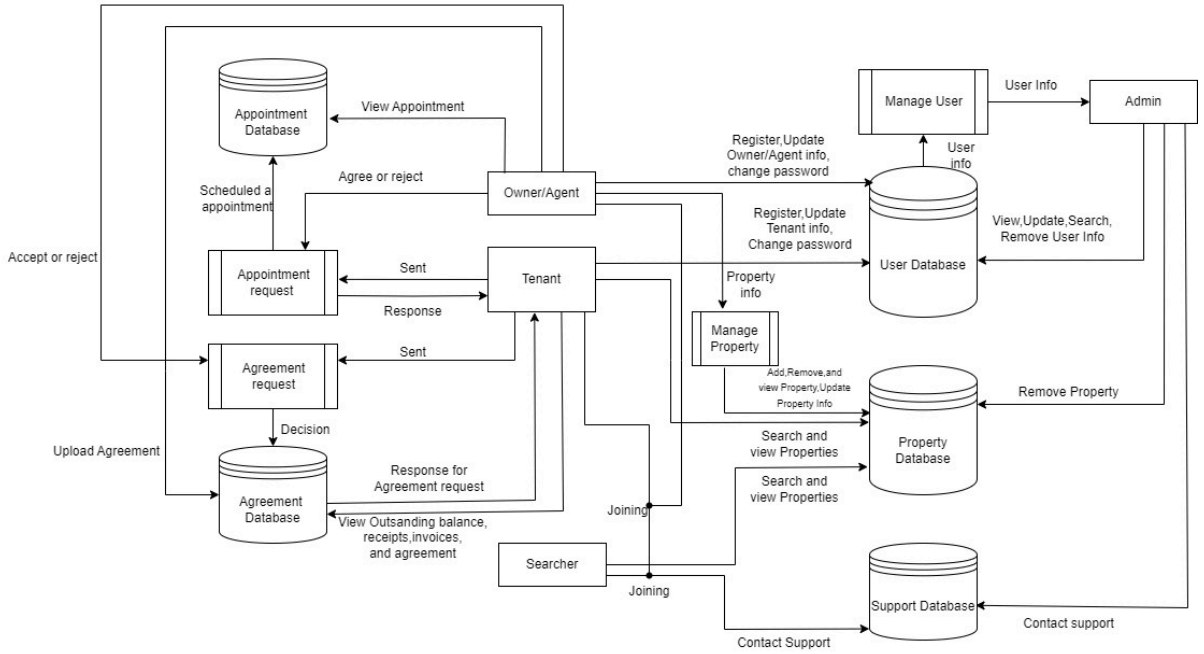
3.2.4 Searcher

The entry point for searchers doesn't require login credentials, providing immediate access to valuable functionalities. Searchers can seamlessly explore the system by initiating property searches and gaining insights into property details. Additionally, searchers have the option to engage with the support team by selecting the Contact Support feature.



3.3 Data Flow Diagrams

This Data Flow Diagram (DFD) intricately outlines the interactions and processes within the Housepooling system, featuring four key entities: Owner/Agent, Tenant, Admin, and Searcher. The system revolves around four pivotal processes: Appointment Request, Agreement Request, Manage User, and Manage Property. The heart of the system lies in five databases, each serving a distinct purpose. The User Database accommodates registrations, updates, and password changes for both Owners/Agents and Tenants. Admins leverage the Manage User process to retrieve, view, update, search, and manage user information within this database. Owners/Agents play a crucial role in property management through the Manage Property process, allowing them to add, remove, view, and update property information stored in the Property Database. Admins have the authority to remove properties from this database. Tenants and Searchers interact with the Property Database to search and view available properties. The Support Database acts as a central repository for support team information, accessible by all entities. Appointment requests from Tenants are processed and stored in the Appointment Database. Owners/Agents can then review and respond to these requests, influencing the status stored within the database.
Tenancy Agreement Requests, initiated by Tenants and responded to by Owners/Agents, are recorded in the Agreement Database, capturing the decisions made during the acceptance or rejection process. Tenants can subsequently access details related to outstanding balances, receipts, invoices, and agreements stored in this database. Owners/Agents also contribute to the Agreement Database by uploading agreements, enriching the system with comprehensive tenancy information.


Appointment Database
View Appointment
Manage User
User Info
Admin
Register,Update Owner/Agent info, change password
User info
Agree or reject
Scheduled a appointment
Owner/Agent
Register,Update Tenant info, Change password
View,Update,Search, Remove User Info
Accept or reject
Appointment request
Sent
Tenant
User Database
Property info
Response
Manage Property
Sent
Agreement request
Add,Remove,and view Property,Update Property Info
Remove Property
Decision
Upload Agreement
Search and view Properties
Property Database
Response for Agreement request
Search and view Properties
Agreement Database
Joining
View Outsanding balance, receipts,invoices, and agreement
Searcher
Joining
Support Database
Contact support
Contact Support

4 Design

4.1 Data Dictionary

Our data dictionary serves as a pivotal document ensuring data integrity and promoting a clear understanding of your system's structure. In this report, we present a detailed overview to enhance your comprehension. Organized into sections such as tables, fields, and data types, the data dictionary provides a systematic approach to understanding the intricacies of your database. Each table is introduced with a succinct purpose statement.

User Collection

This collection outlines essential attributes for user information. Each attribute is defined by its name, contents, data type, format, length, and whether it is a required field. The 'User' collection encapsulates details crucial for user identification and classification. Understanding the nature of data in this collection is pivotal for user authentication, access control, and personalized user experiences.

Collection Name	Attribute Name	Contents	Type	Format	Length	Required
User	name	Name of the User	String	Xxxxx	50	Y
	password	Password of the User	String	Xxxxx	20	Y
	email	Email of the user	String	Xxxxx	20	Y
	icNum	IC number of the user	String	Xxxxx	12	Y
	phone	Phone number of the User	Integer	9999999	20	Y
	gender	Gender of the user	String	Xxxxx	6	Y

Property Collection

This collection encompasses essential attributes for property details, including its name, address, type, dimensions, pricing, and deposit information. The 'Property' collection is vital for managing and classifying real estate assets. Understanding these attributes is crucial for property management, tenant interactions, and financial planning related to real estate.

Collection Name	Attribute Name	Contents	Type	Format	Length	Required
Property	name	name of the property	String	Xxxxxxx	50	Y
	address	address of the property	String	Xxxxxxx	50	Y
	type	type of the property	String	Xxxxxxx	20	Y
	area	area of the property	Double	9999.99	20	Y

	rent	price of the property	Integer	99999999	10	Y
	utilitiyDeposit	utility deposit of the property	Integer	99999999	10	Y
	securityDeposit	security deposit of the property	Integer	99999999	10	Y
	keyDeposit	key deposit of the property	Integer	99999999	10	Y
	numberOfRoom	number of room for property	Integer	99999999	10	Y
	numberOfBathroom	number of bathroom for property	Integer	99999999	8	Y
	isRentable	does the property have active tenant	Boolean	True/false	-	Y

Support Collection

This collection encapsulates fundamental attributes for the 'Support' collection, detailing the contact team's name, email, and phone number. It serves as a central point of communication for addressing user inquiries or concerns. Understanding these attributes is critical for maintaining effective communication channels, providing support, and ensuring prompt resolution of user issues.

Collection Name	Attribute Name	Contents	Type	Format	Length	Required
Support	Name	The name of the contact team	String	Xxxxxxxxxx	30	Y
	Email	The email of the contact team	String	Xxxxxxxxxx	20	Y
	Phone	The phone number of contact team	Integer	99999999	11	Y

Agreement Collection

This collection outlines crucial attributes for rental agreements, including tenant and owner details, tenancy period, financial terms, and additional clauses. The 'Agreement' collection is essential for managing and documenting rental agreements effectively. Understanding these attributes is pivotal for legal compliance, financial tracking, and dispute resolution in the context of rental agreements.

Collection Name	Attribute Name	Contents	Type	Format	Length	Required
Agreement	tenantName	Name of tenant	String	Xxxxxxxx	50	Y
	ownerName	Name of owner	String	Xxxxxxxx	50	Y
	tenancyPeriod	Fixed period of time which the tenant is renting the place	String	Xxxxxxxx	10	Y

	Rent	price of the property	Integer	99999999	10	Y
	utilitiyDeposit	utility deposit of the property	Integer	99999999	10	Y
	securityDeposit	security deposit of the property	Integer	99999999	10	Y
	keyDeposit	key deposit of the property	Integer	99999999	10	Y
	Penalty	Include late payment fees and damages	String	Xxxxxxxx	10	Y
	additionalClauses	Extra conditions that are not covered by standard terms.	String	Xxxxxxxx	10	Y

Appointment Collection

This collection delineates key attributes for managing appointments, including tenant and owner details, date and time, property information, and the address where the appointment takes place. The 'Appointment' collection is essential for scheduling and coordinating property-related activities. Understanding these attributes is crucial for organizing appointments efficiently, facilitating communication between tenants and owners, and maintaining a systematic approach to property management.

Collection Name	Attribute Name	Contents	Type	Format	Length	Required
Appointment	tenantName	Name of tenant	String	Xxxxxxxx	30	Y
	OwnerName	Name of owner	String	Xxxxxxxx	30	Y
	Date	Date of the Appointment	date	YYYY-mm-dd	-	Y
	Time	Time of the Appointment	time	HH:MM:ss	-	Y
	propertyName	Name of property	String	Xxxxxxxx	30	Y
	address	Name of address	String	Xxxxxxxx	50	Y

4.2 Data Structures

—

4.3 Software Architecture

The HomeWoW system comprises seven distinct subsystems: Financial, User Management, Authentication, Profile Management, Information display, Property Management, and Communication. Each subsystem plays a specific role in the overall system. All system users must log in to access these respective subsystems, except for the searcher who can navigate without logging in.

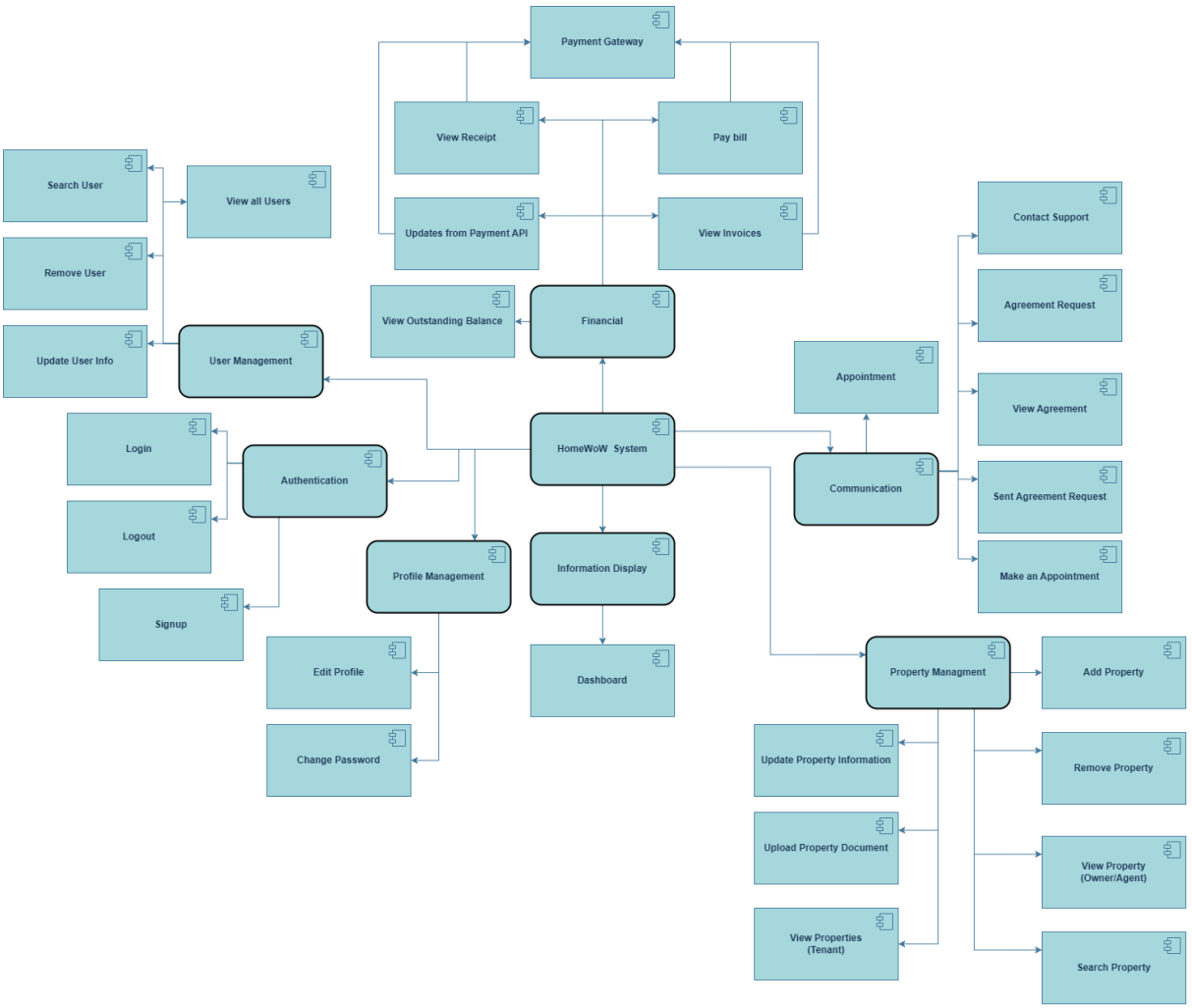
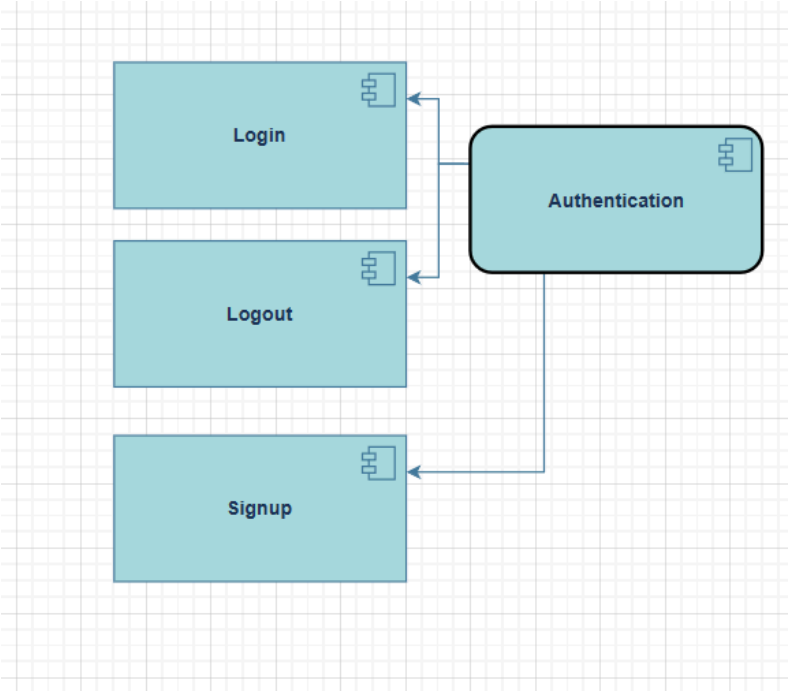


Table: Assigned Subsystem	
SADMAN ZULFIQUER	Authentication & Property Management
TAN TENG HUI	Profile Management & Communication
TENG WEI JOE	User Management
HO TECK FUNG	Financial & Dashboard

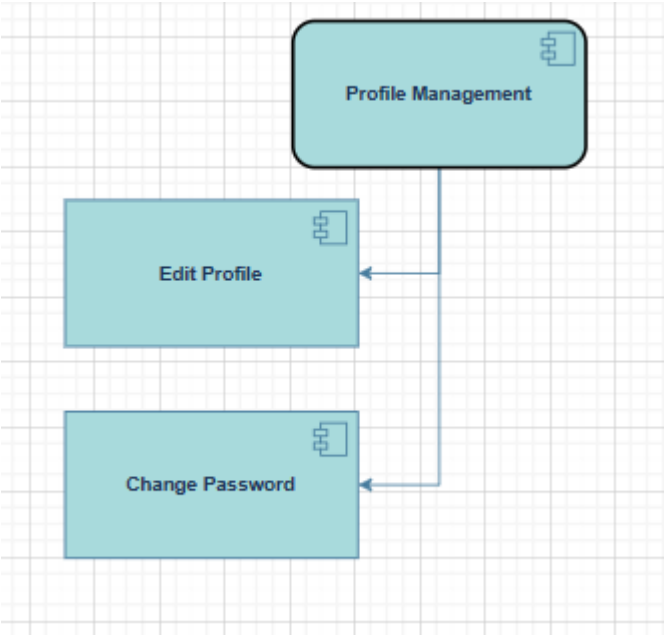
4.3.1 Authentication Subsystem

The Authentication subsystem is used for authenticating users during login, logout, and signup processes.



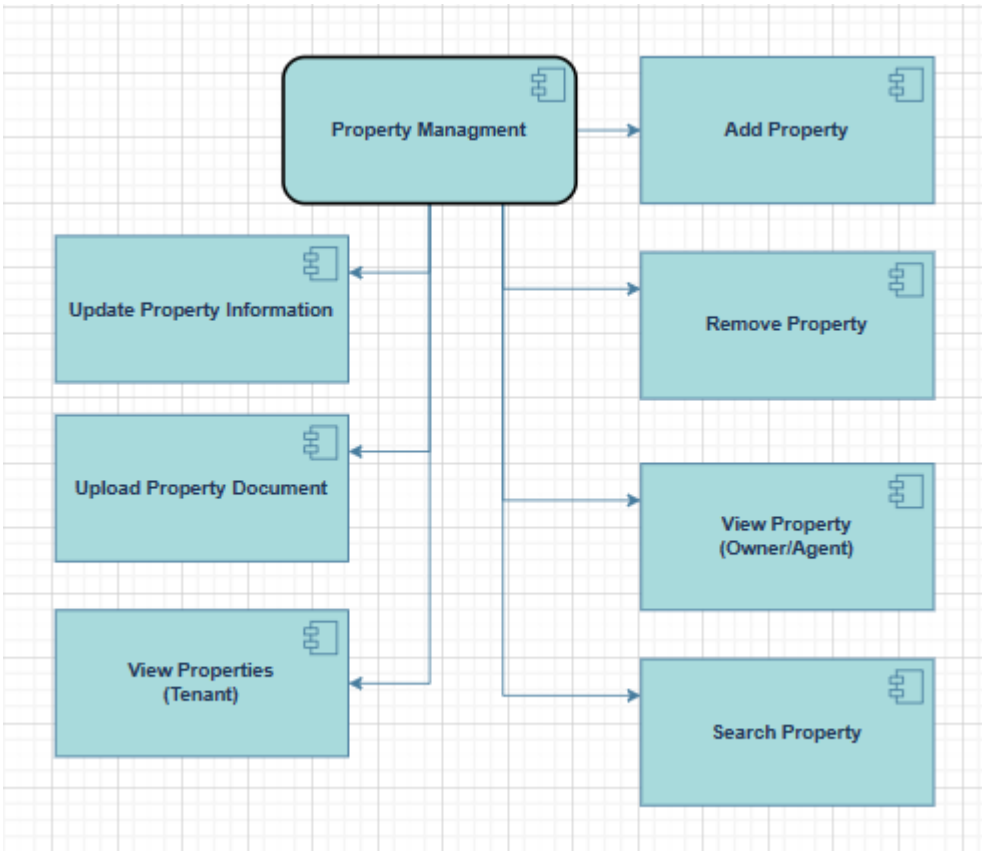
4.3.2 Profile Management Subsystem

The Profile Management Subsystem is used to manage user profiles, preferences, personal information and includes operations related to changing user's passwords for enhanced security.



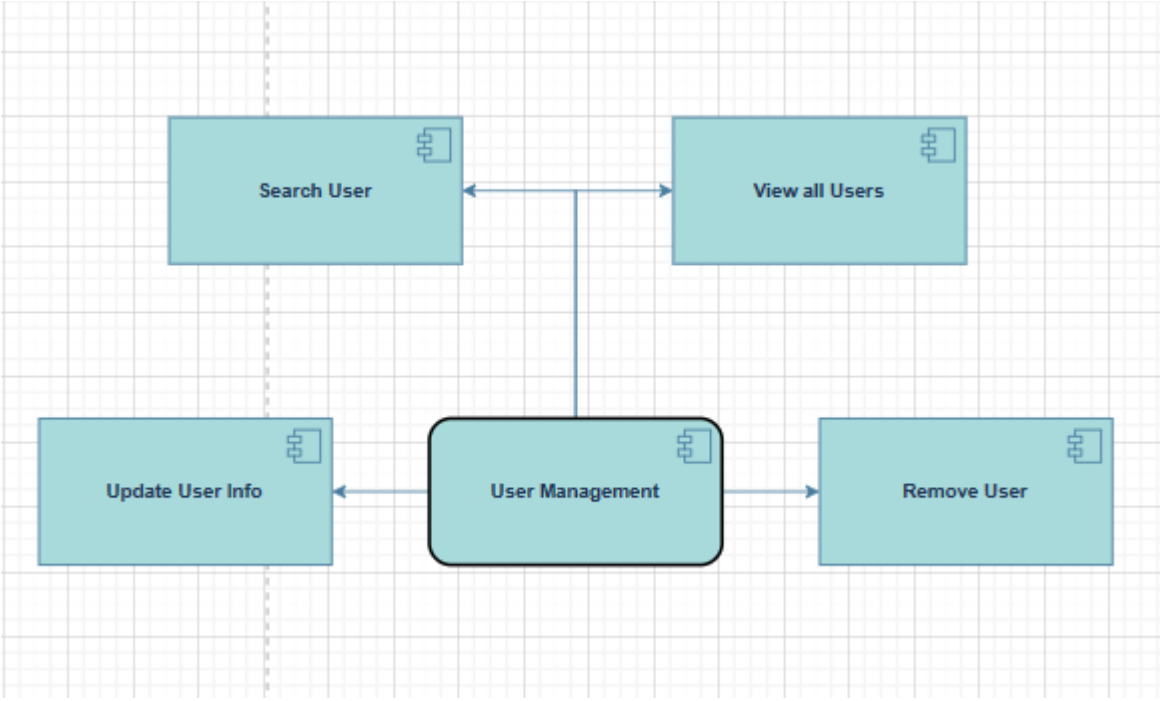
4.3.3 Property Management Subsystem

The Property Management Subsystem allows property owners to add new properties and remove property from the system. It also provides search functionality for users (tenants) to find properties based on specific criteria. Tenants will be able to view available properties within the system and the property owners to view and manage their own properties. The Subsystem also supports the upload of property information, including details like descriptions, images, and amenities, and facilitates the upload of property-related documents.



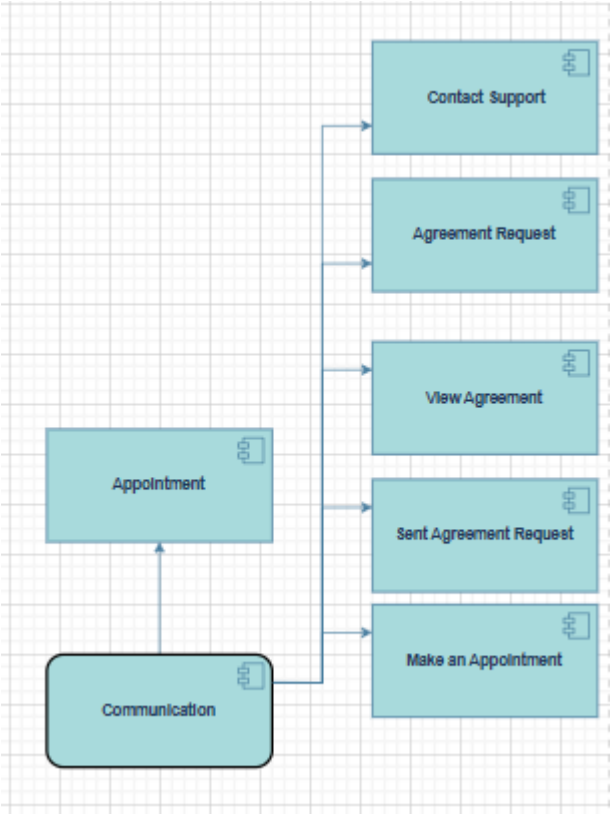
4.3.4 User Management Subsystem

The User Management Subsystem allows the administrators to search for specific users and provides a view of all users within the system.It also supports the update of user information, including profile details and preferences. Furthermore, administrators also can remove user's accounts.



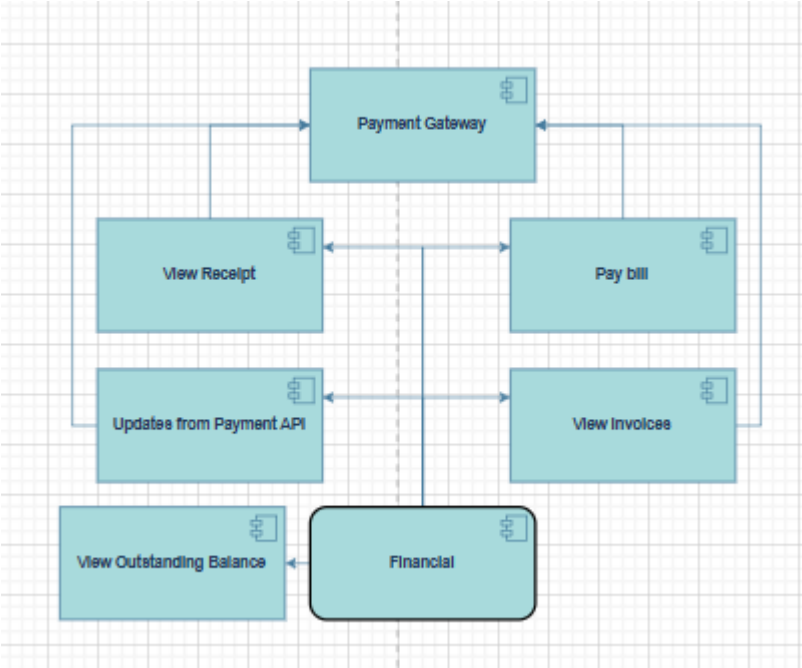
4.3.5 Communication Subsystem

The communication subsystem allows users to contact support for assistance. The tenant can send an agreement request and view the agreement. The Owner can view and choose to accept and send the agreement or reject the request. The subsystem also supports the scheduling of appointments for property viewings and facilitates the confirmation of appointments, typically by property owners.



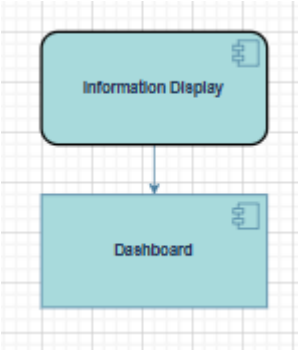
4.3.6 Financial Subsystem

The Financial Subsystem allows users to view detailed receipts of past payments and provides the ability to view invoices, payment history, and records of bills. The subsystem also supports updating financial records from a Payment API and facilitates secure payment of bills through an integrated Payment Gateway. Lastly, users can view details of outstanding balances related to their tenancy.



4.3.7 Information Display Subsystem

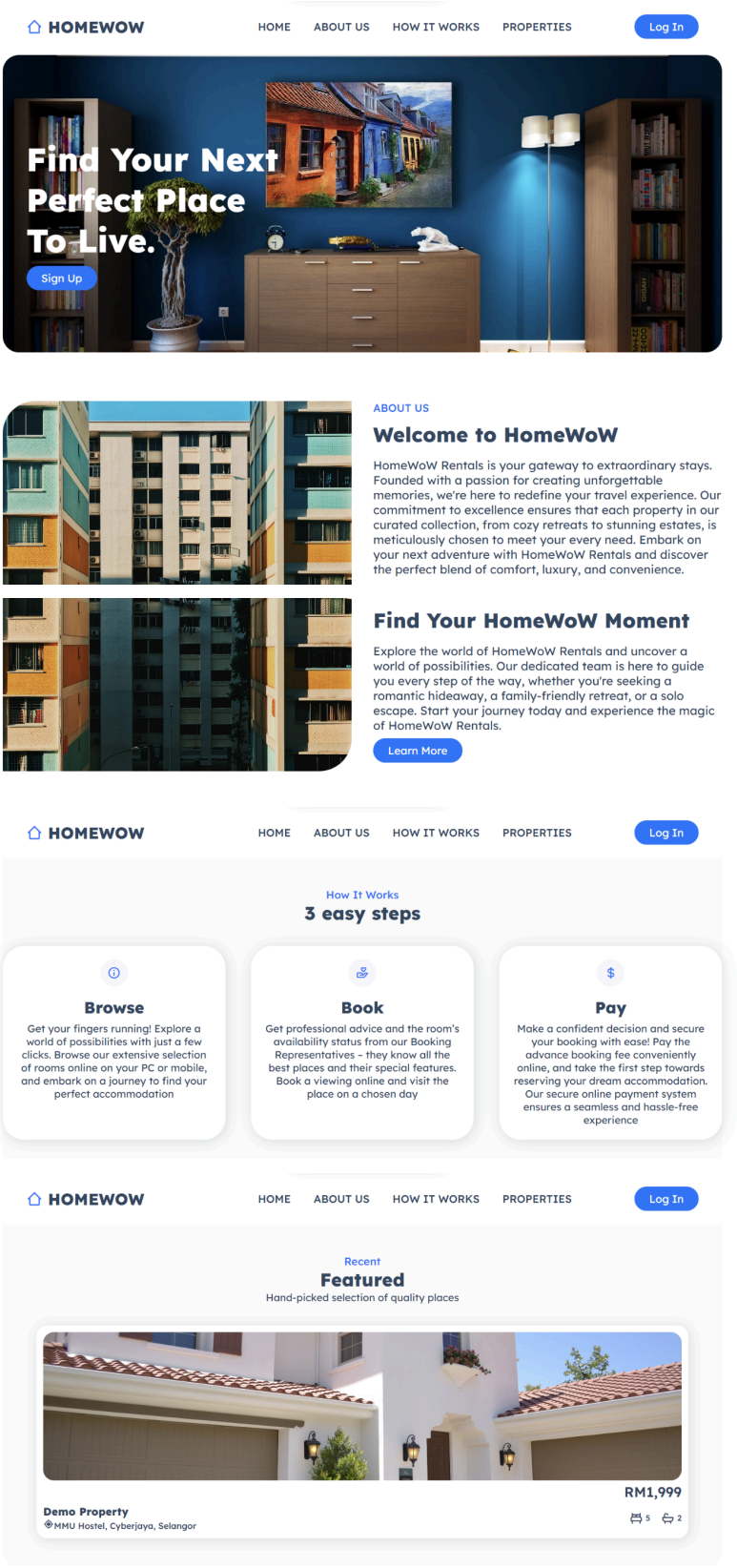
The Information Display Subsystem provides a centralized view for users to access a summarized overview of their account, agreements, payments, and other relevant information.

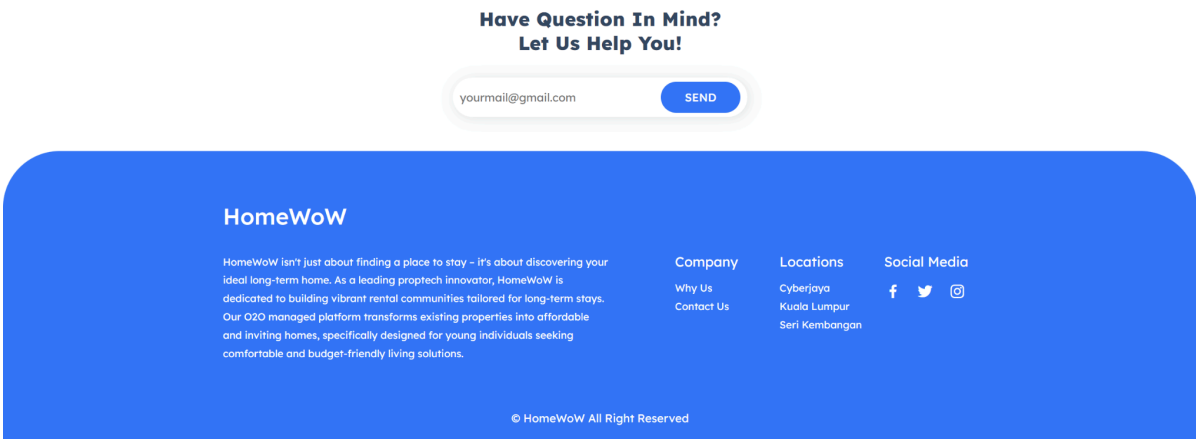


4.4 Main Screens

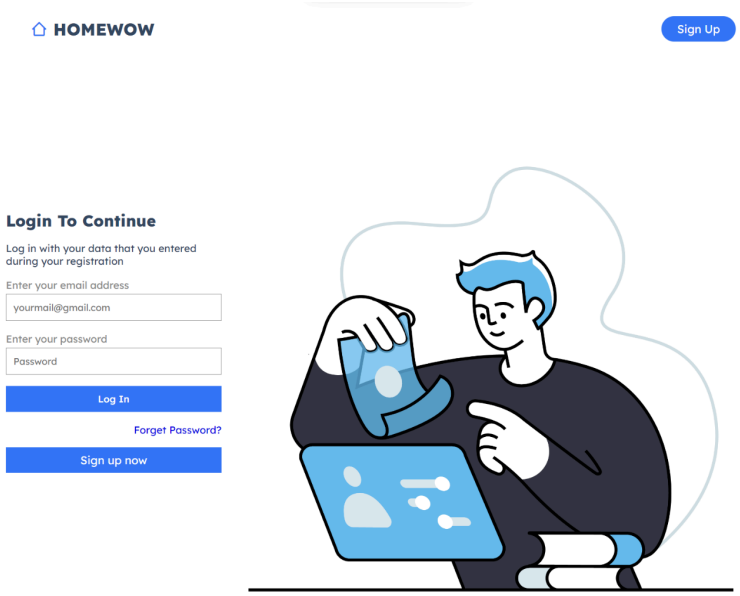
We carefully crafted our website with a warm and inviting color palette, fostering a sense of comfort and familiarity for our users. Our intention was to create a design that is not only visually appealing but also intuitive, ensuring effortless navigation for every visitor.

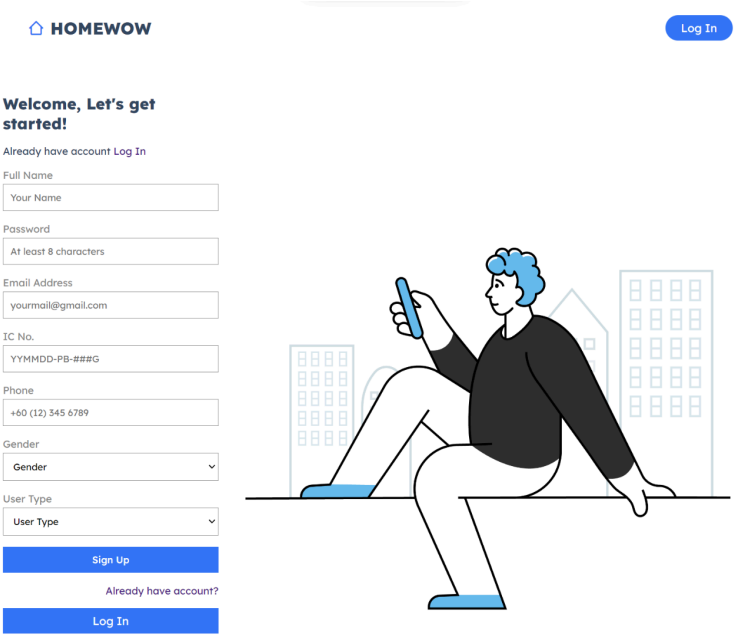
- ***4.4.1 Home page***



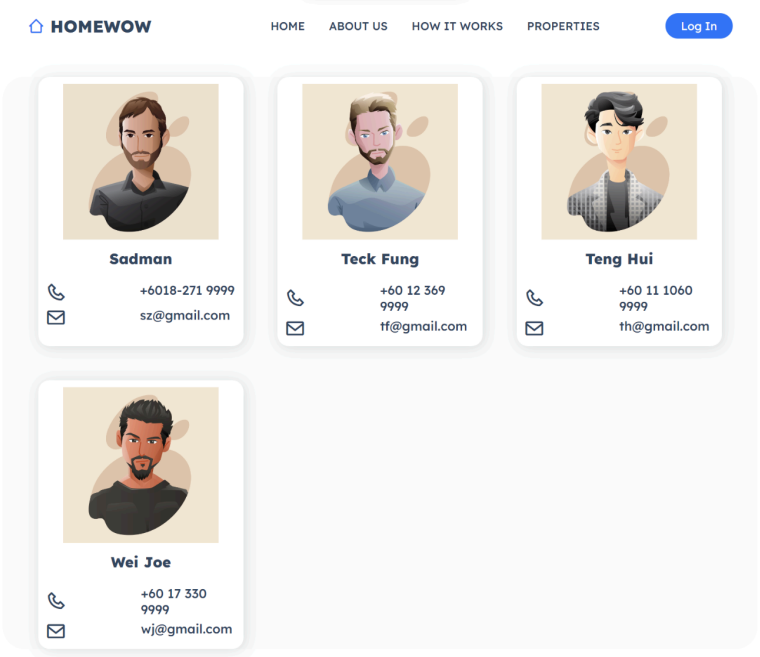


- ***4.4.2 Login / Signup page***





- ***4.4.3 Contact page***



4.5 Main Components

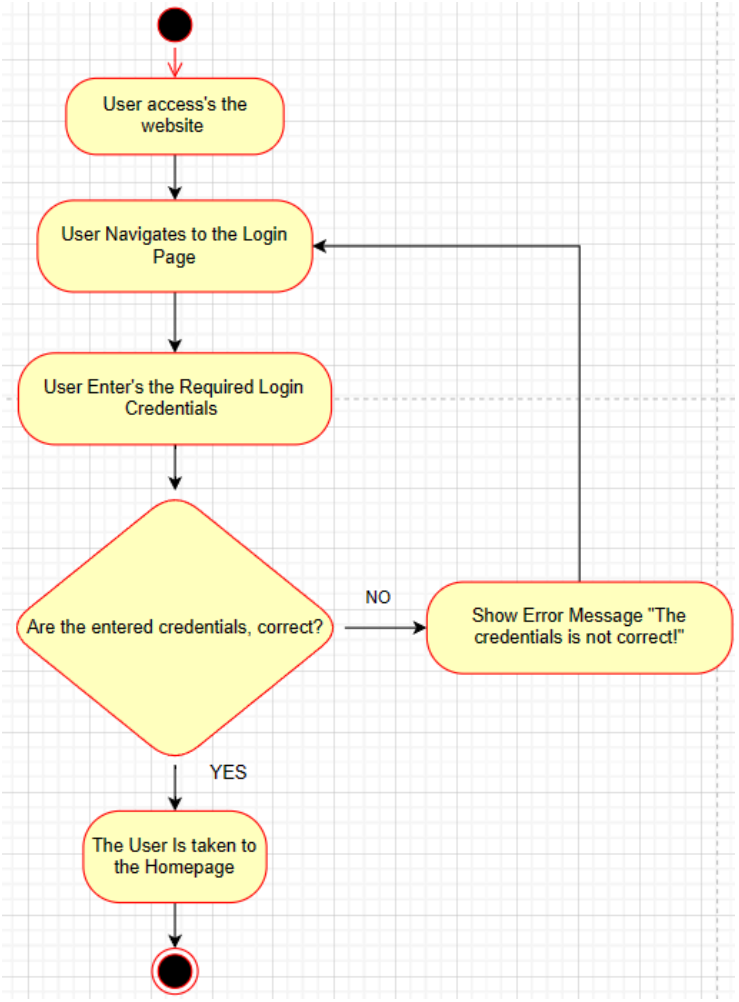
Subsystem	Description
Authentication	The Authentication subsystem is used for authenticating users during login, logout, and signup processes.
Profile Management	The Profile Management Subsystem is used to manage user profiles, preferences, personal information and includes operations related to changing user's passwords for enhanced security.
Property Management	The Property Management Subsystem allows property owners to add new properties and remove property from the system. It also provides search functionality for users (tenants) to find properties based on specific criteria. Tenants will be able to view available properties within the system and the property owners to view and manage their own properties. The Subsystem also supports the upload of property information, including details like descriptions, images, and amenities, and facilitates the upload of property-related documents.
User Management	The User Management Subsystem allows the administrators to search for specific users and provides a view of all users within the system.It also supports the update of user information, including profile details and preferences. Furthermore, administrators also can remove user's accounts.
Communication	The communication subsystem allows users to contact support for assistance. The tenant can send an agreement request and view the agreement. The Owner can view and choose to accept and send the agreement or reject the request. The subsystem also supports the scheduling of appointments for property viewings and facilitates the confirmation of appointments, typically by property owners.

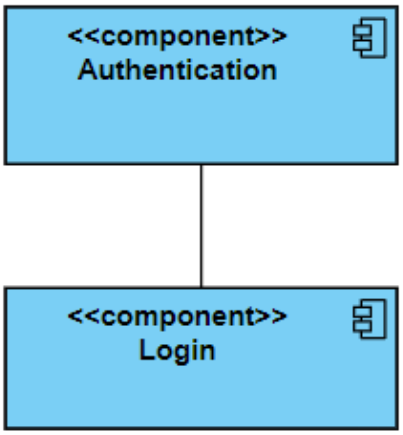
Financial	The Financial Subsystem allows users to view detailed receipts of past payments and provides the ability to view invoices, payment history, and records of bills. The subsystem also supports updating financial records from a Payment API and facilitates secure payment of bills through an integrated Payment Gateway. Lastly, users can view details of outstanding balances related to their tenancy.
Information Display	The Information Display Subsystem provides a centralized view for users to access a summarized overview of their account, agreements, payments, and other relevant information.

Subsystem	**Components**
Authentication	1. Login 2. Logout 3. Signup
Profile Management	4. Edit Profile 5. Change Password
Property Management	6. Add Property 7. Remove Property 8. View Property (Owner/Agent) 9. Search Property 10. Update Property Information 11. Upload Property Document 12. View Properties (Tenant & Searcher)
User Management	13. Search User 14. View all users 15. Update User Information 16. Remove User
Communication	17. Contact Support 18. Agreement Request 19. View Agreement 20. Sent Agreement Request 21. Make an Appointment (Tenant) 22. Appointment (Owner/Agent)
Financial	23. Pay Bill 24. View Invoices 25. View Receipt 26. Updated from Payment API 27. View Outstanding Balance
Information Display	28. Dashboard

4.5.1 Login(Owner/Agent,Tenant,Admin)

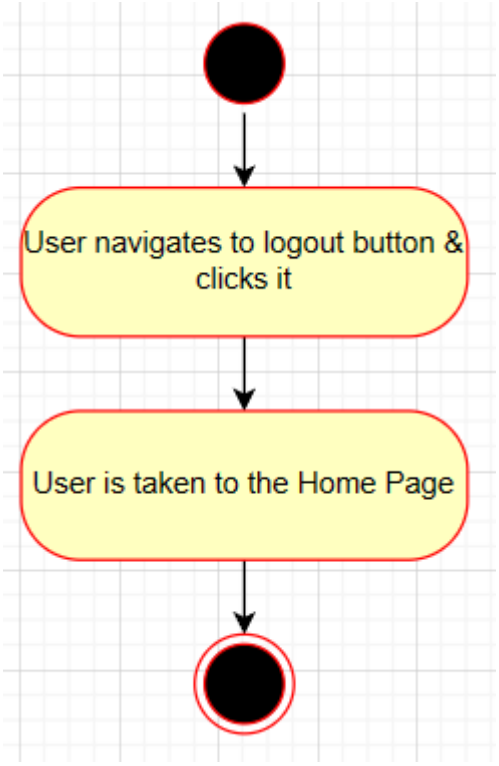
The user will use the authentication component to login into the website.

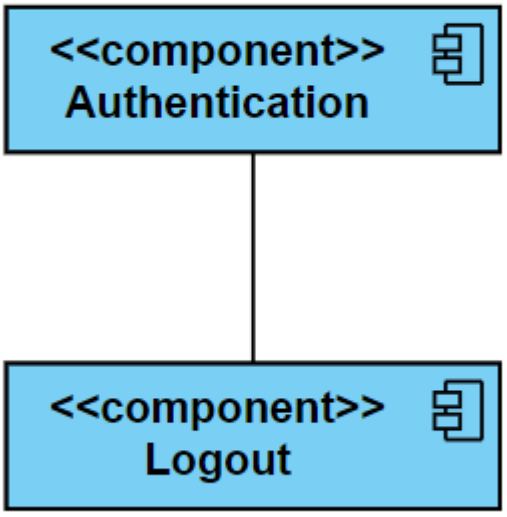




4.5.2 Logout(Owner/Agent, Tenant,Admin)

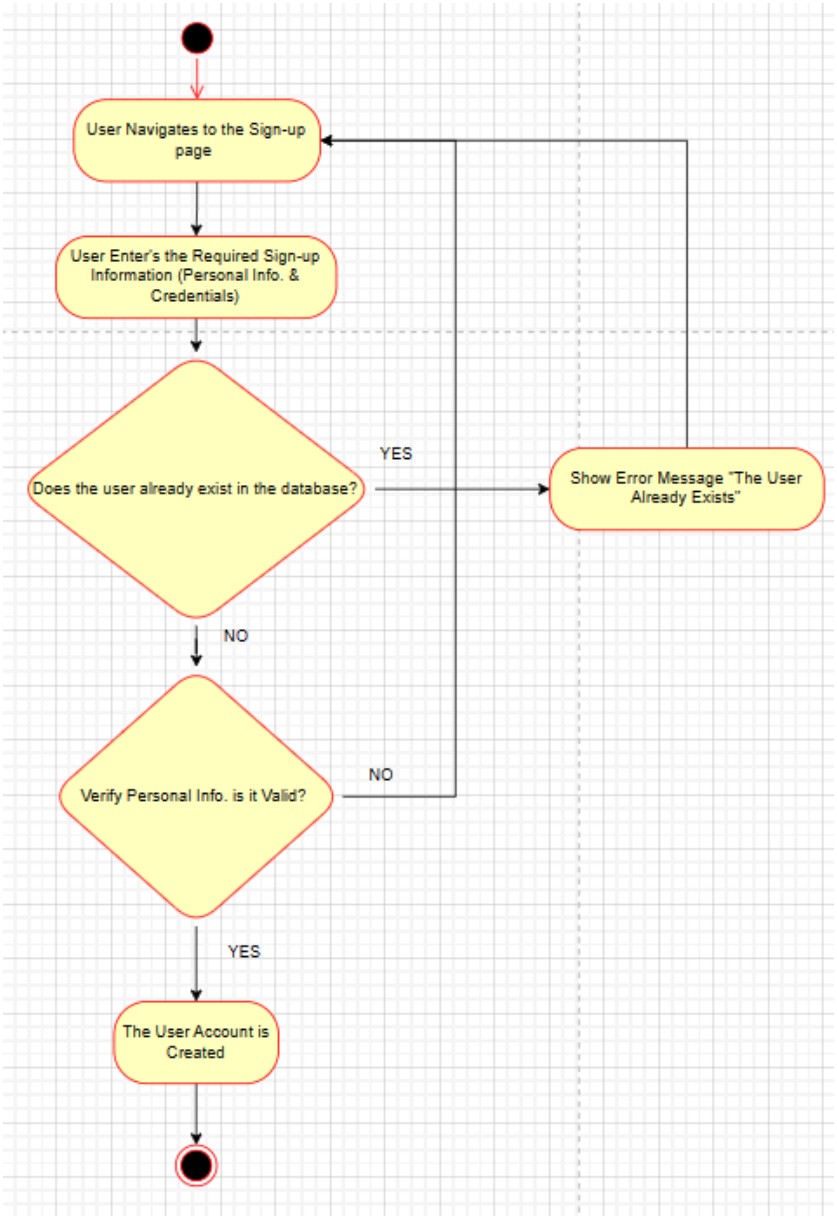
The user will use the authentication component to logout the website and redirected to the home page.

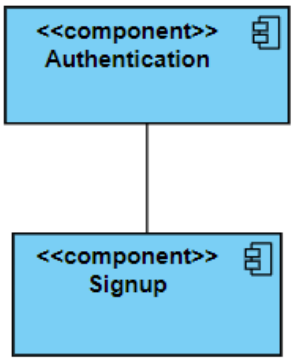




4.5.3 Signup

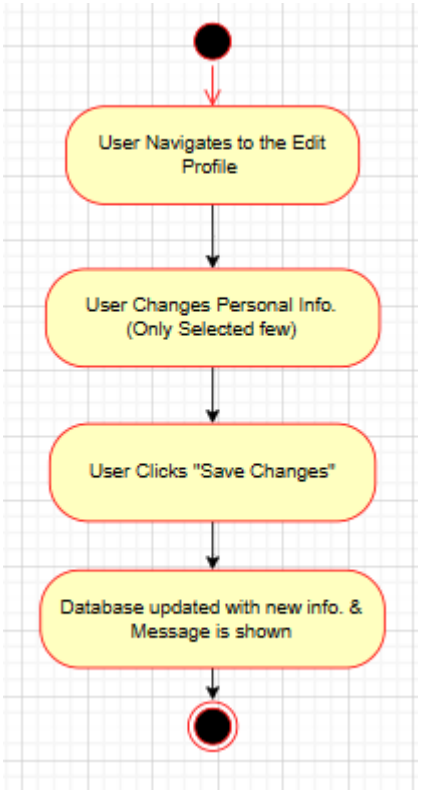
The user uses the authentication component to register an account.

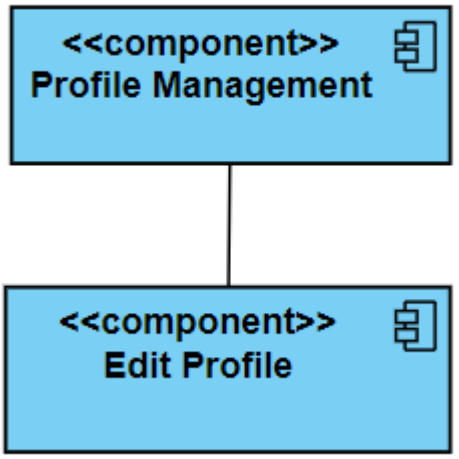




4.5.4 Edit Profile (Owner,Tenant)

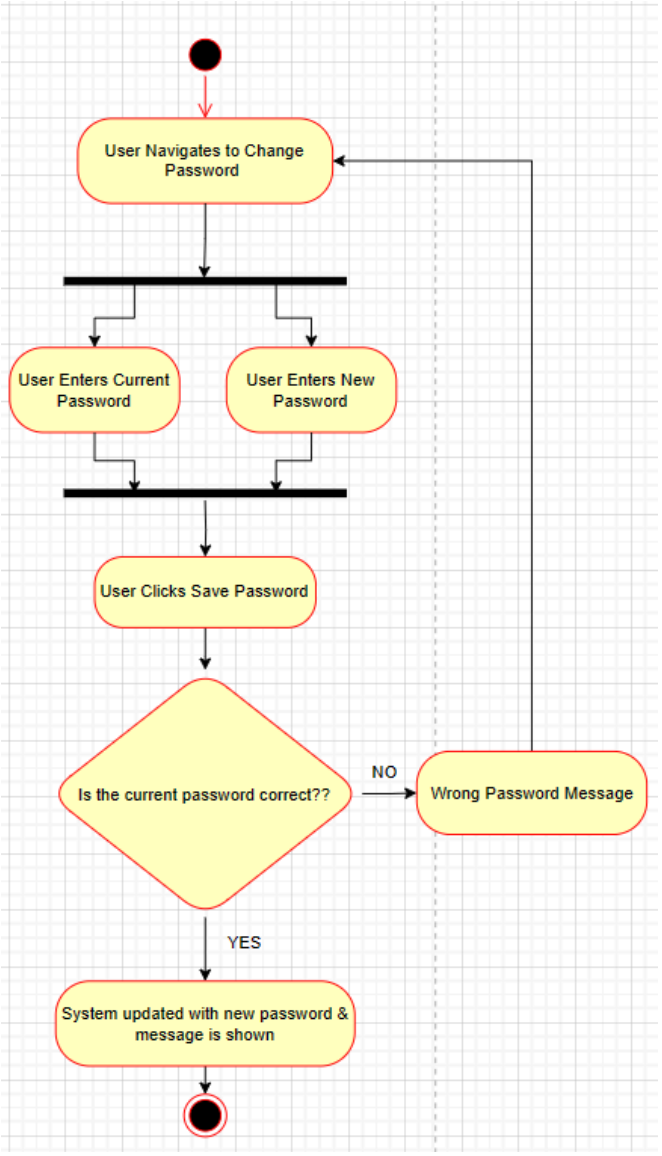
The user will use the Profile Management component to edit personal details, preferences, and other relevant information to keep their profiles up to date.

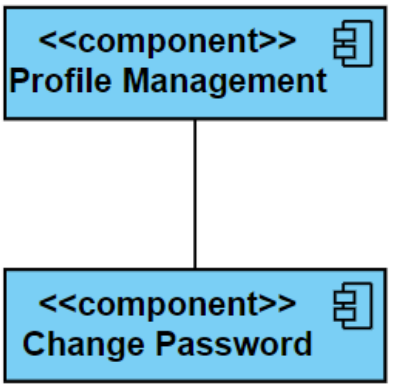




4.5.5 Change Password(Owner/Agent,Tenant)

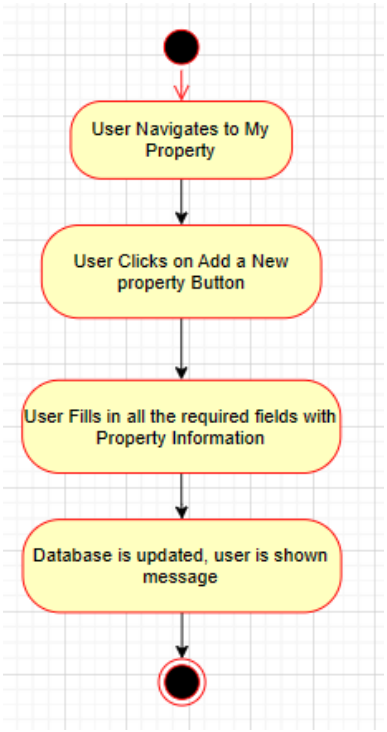
The user will use the Profile Management component to modify their existing password for security reasons, enhancing the security of their account.

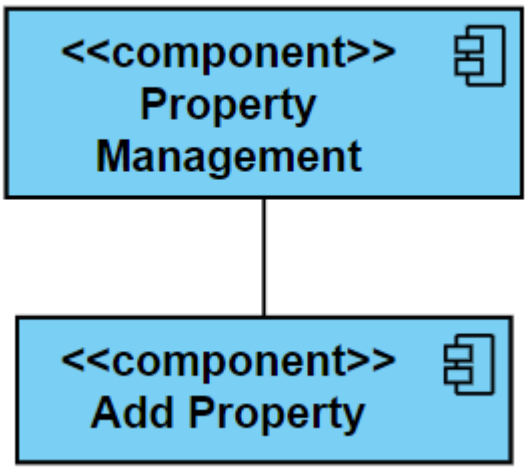




4.5.6 Add Property (Owner/Agent)

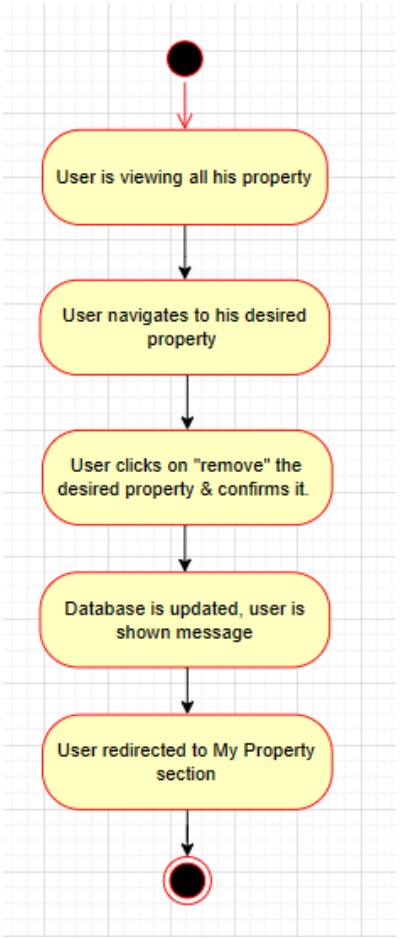
The user will use the Property Management component to add property with all the required information in the system which is only visible to this user.

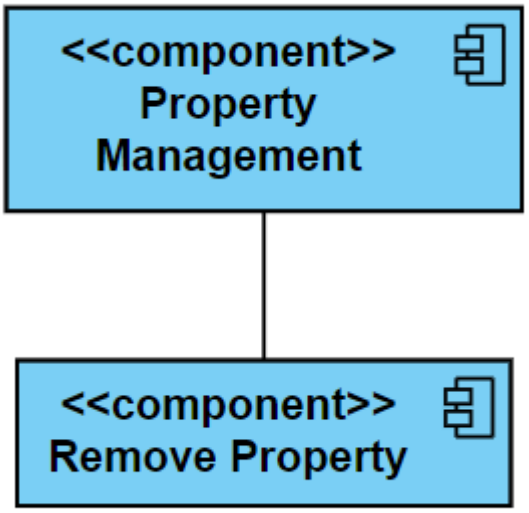




4.5.7 Remove Property(Owner/Agent)

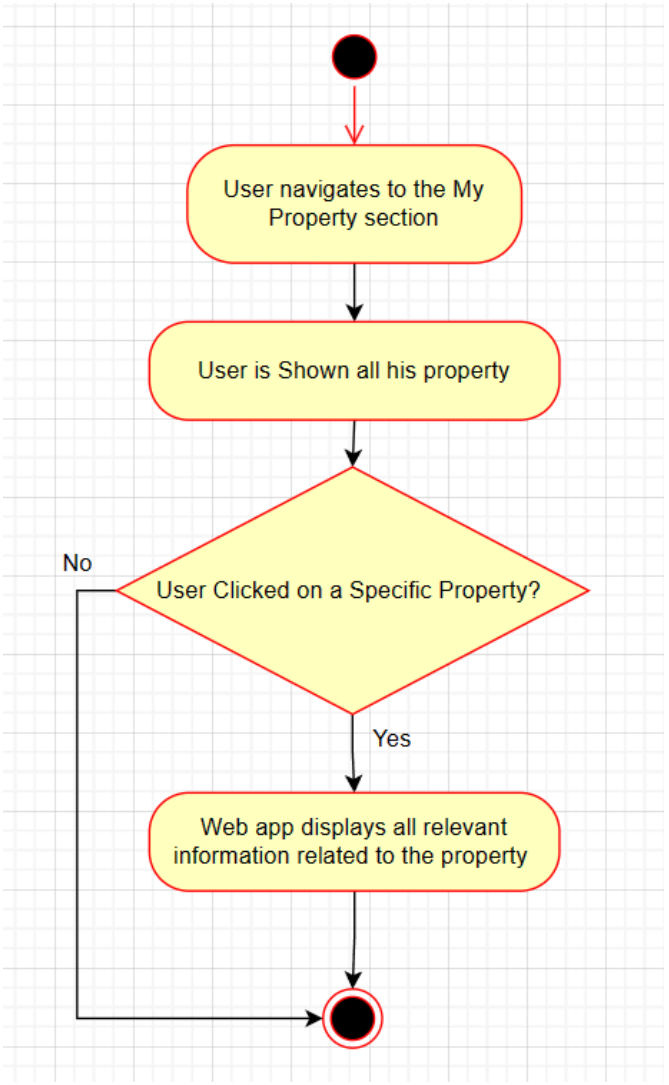
The user will use the Property Management component to removes the desired property from the system.

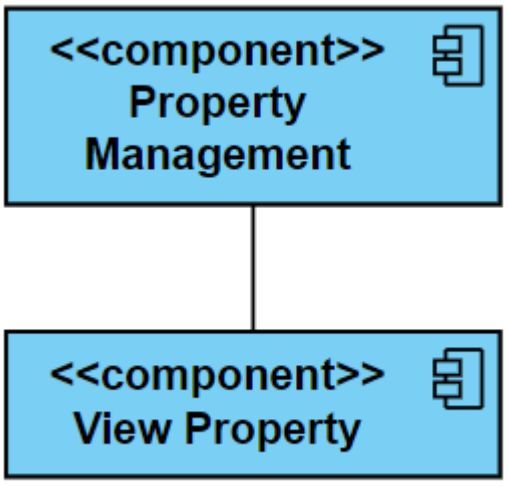




4.5.8 View Property (Owner/Agent)

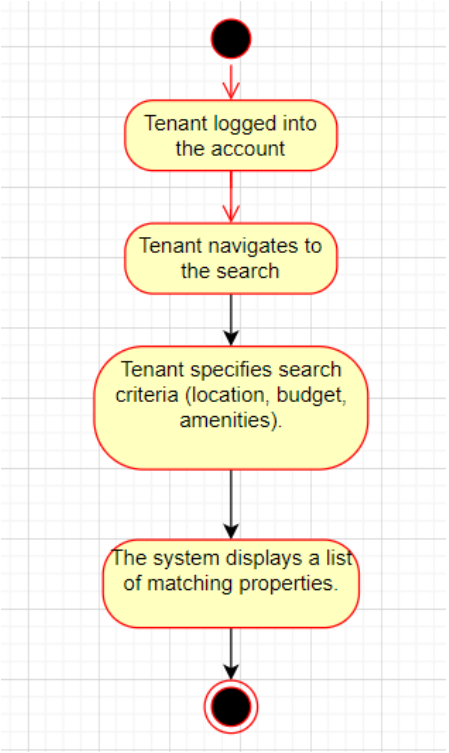
The user will use the Property Management component to view all his properties in the system with details & status.

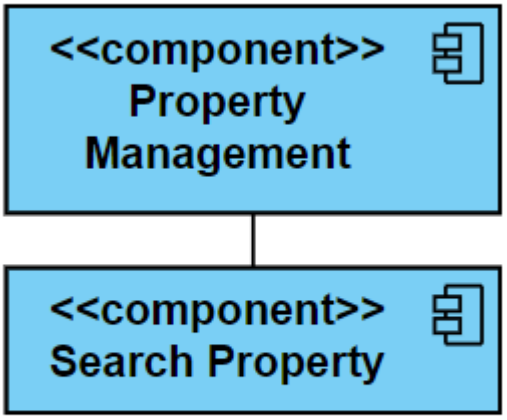




4.5.9 Search Property(Tenant)

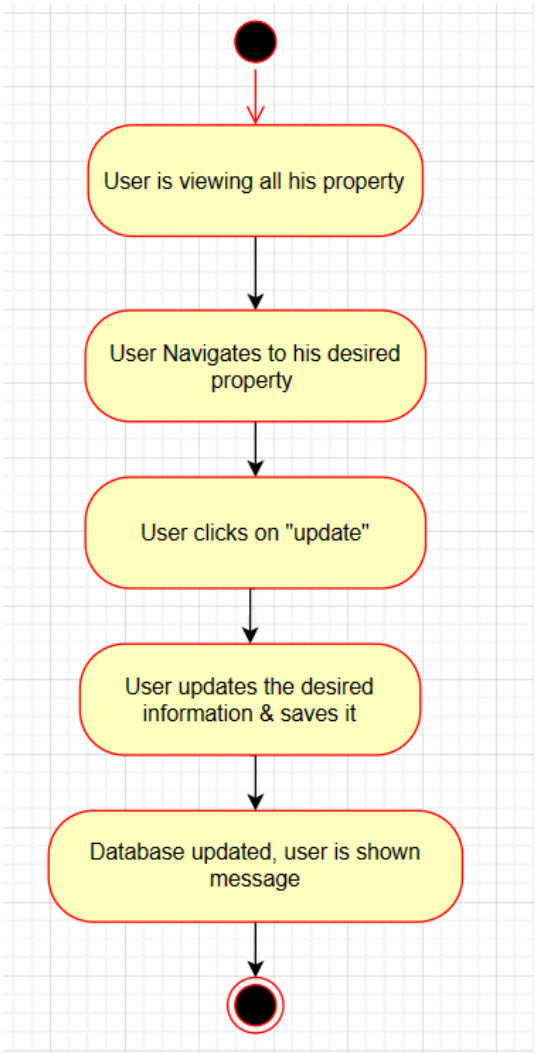
The user will use Property Management component to search properties based on specific criteria.

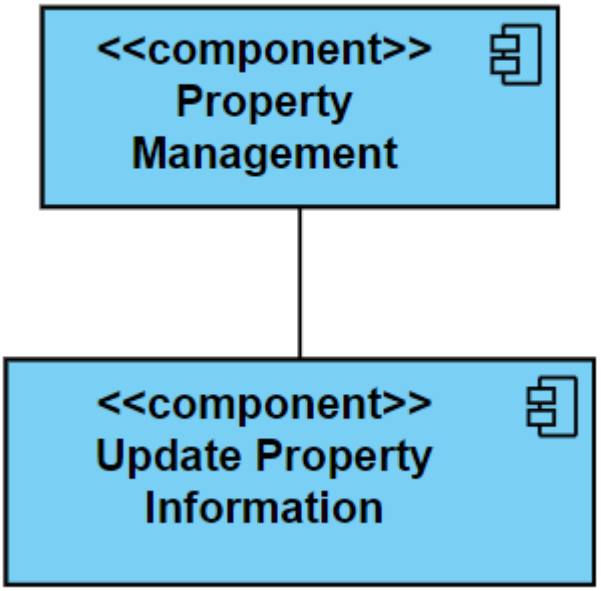




4.5.10 Update Property Information(Owner/Agent)

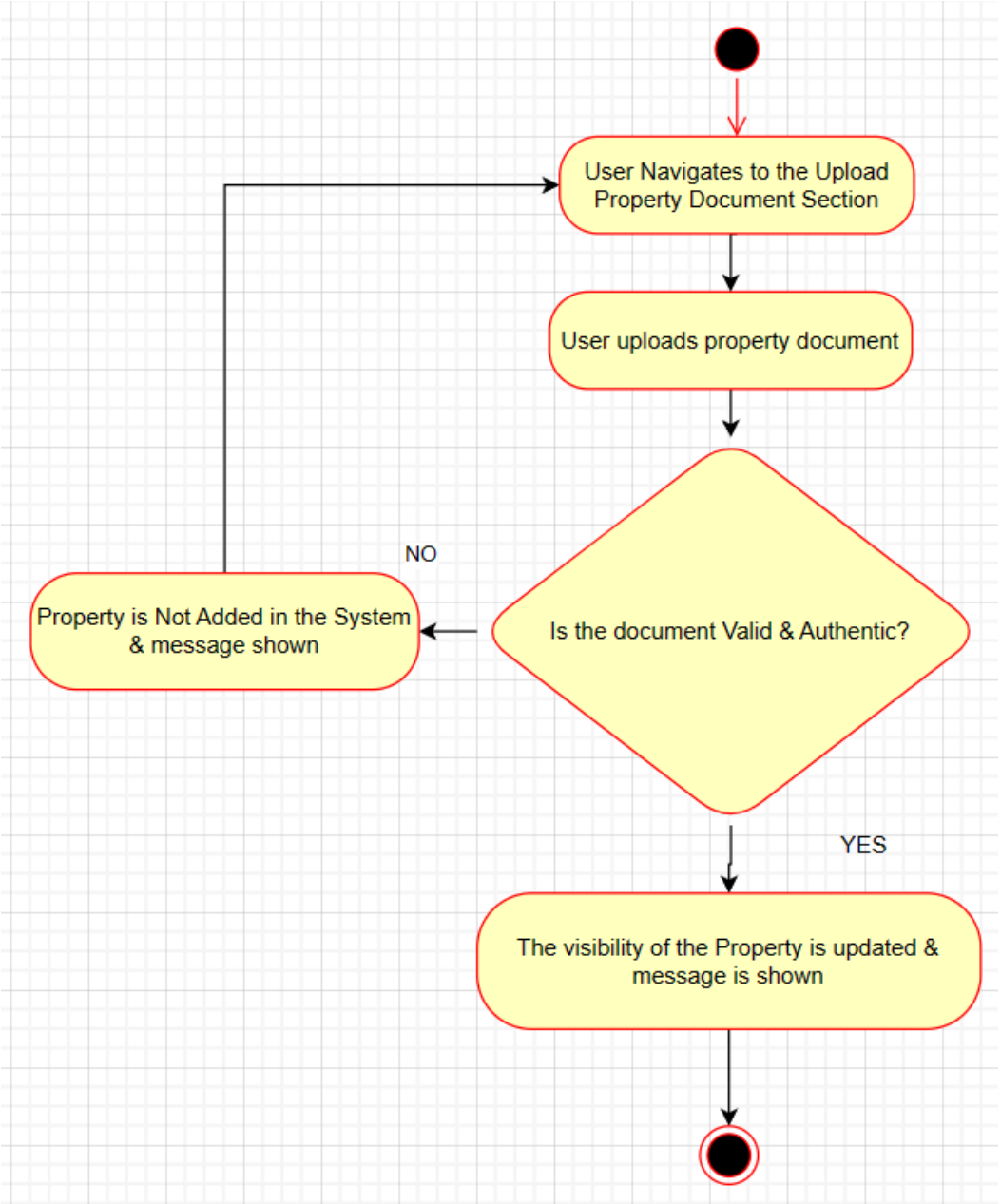
The user will use Property Management component to update any existing property information in the system.

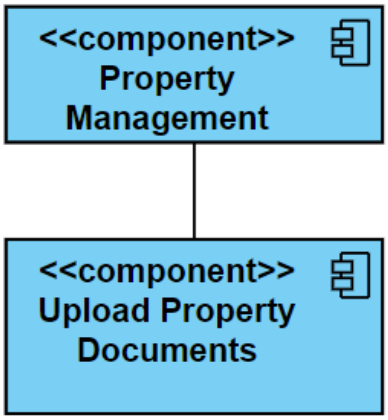




4.5.11 Upload Property Document(Owner/Agent)

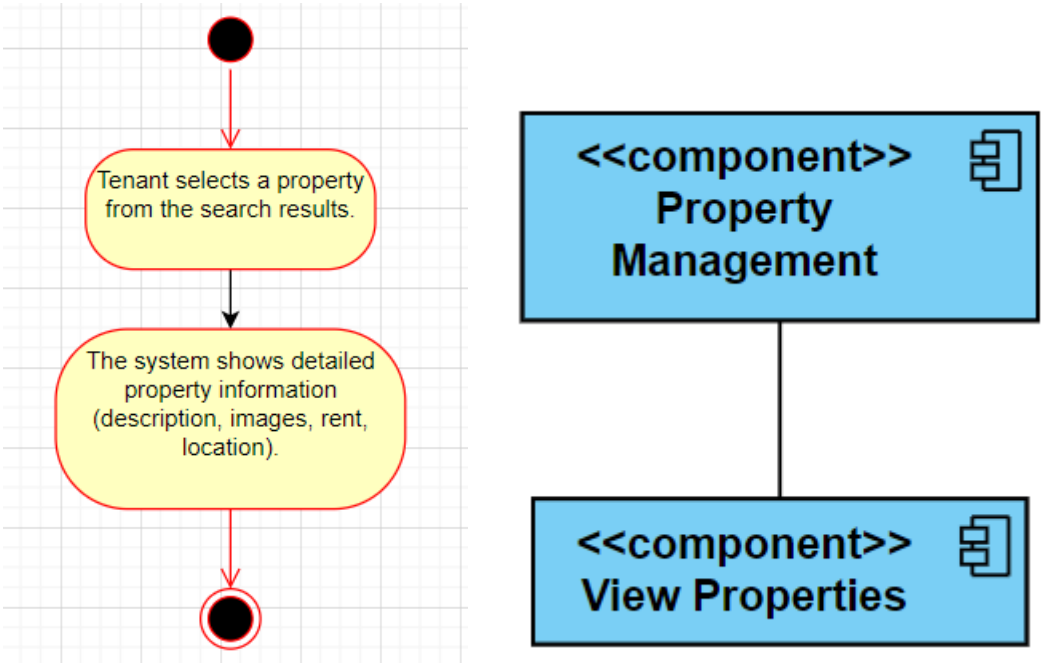
The user will use the Property Management component to upload property documents for verification.


User Navigates to the Upload Property Document Section
User uploads property document
Is the document Valid & Authentic?
NO
Property is Not Added in the System & message shown
YES
The visibility of the Property is updated & message is shown


<<component>>
Property Management
<<component>>
Upload Property Documents

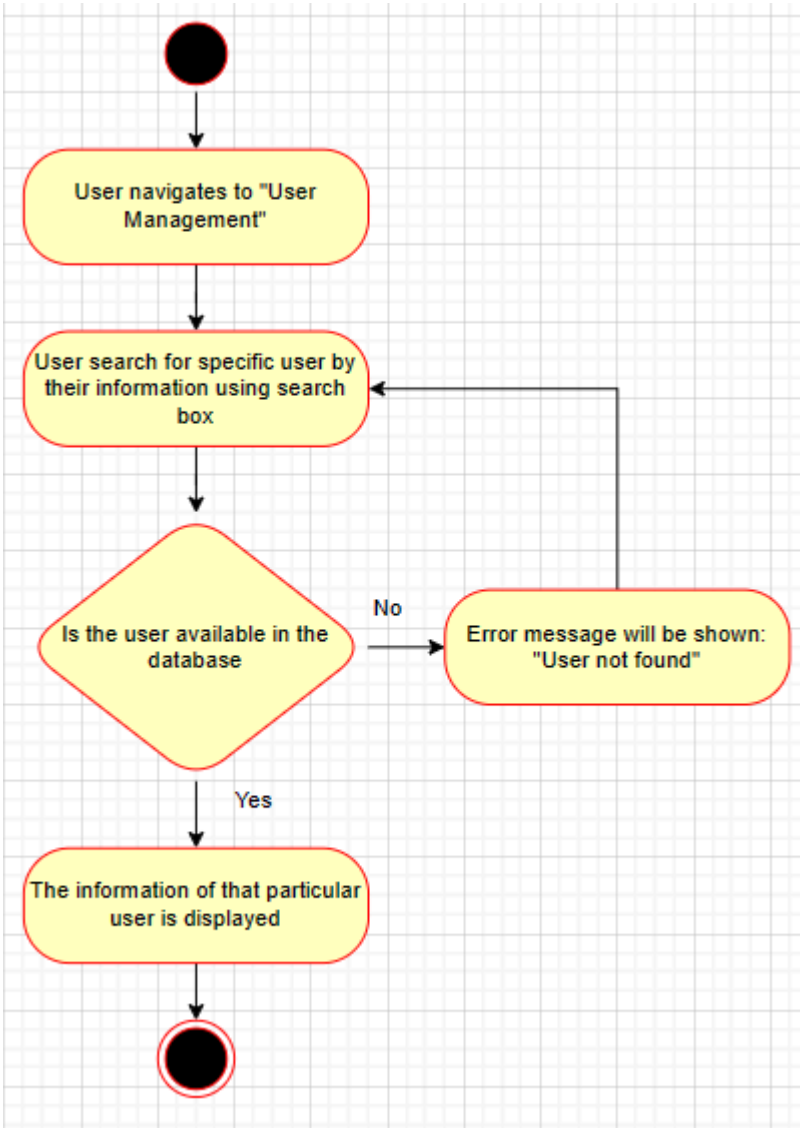
4.5.12 View Properties (Tenant & Searcher)

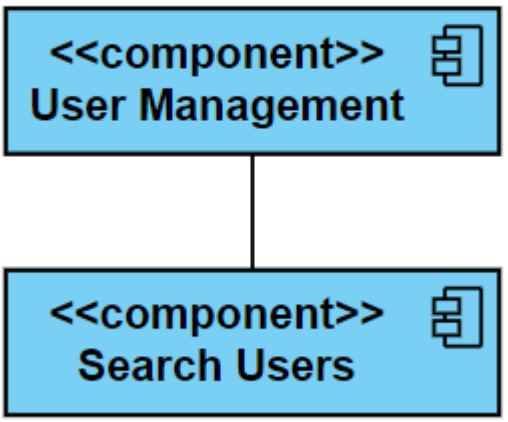
The user will use the Property Management component to view detailed information about a specific property of interest, including descriptions, images, rent, location, and other relevant details.



4.5.13 Search User(Admin)

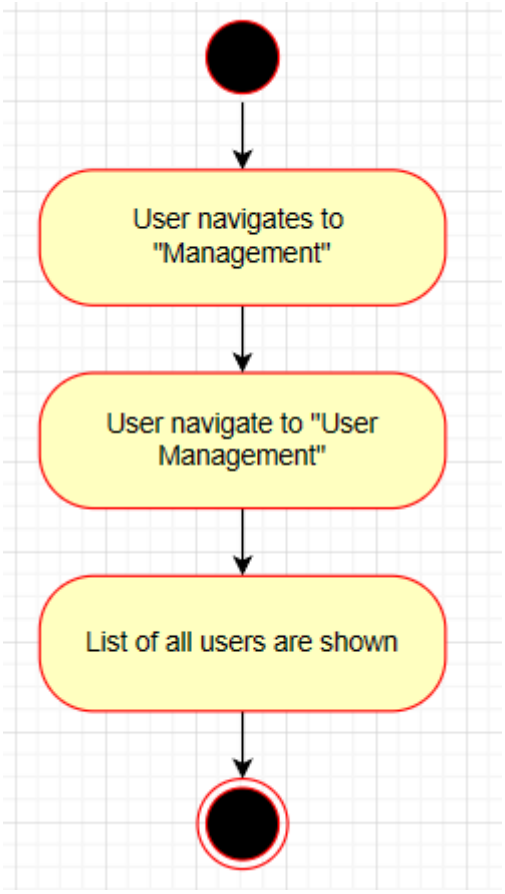
The user will use the User Management component to search for specific users.

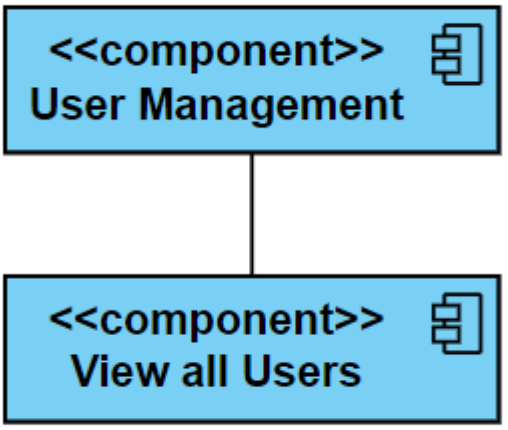

User navigates to "User Management"
User search for specific user by their information using search box
Is the user available in the database
No
Error message will be shown: "User not found"
Yes
The information of that particular user is displayed


<<component>>
User Management
<<component>>
Search Users

4.5.14 View all Users(Admin)

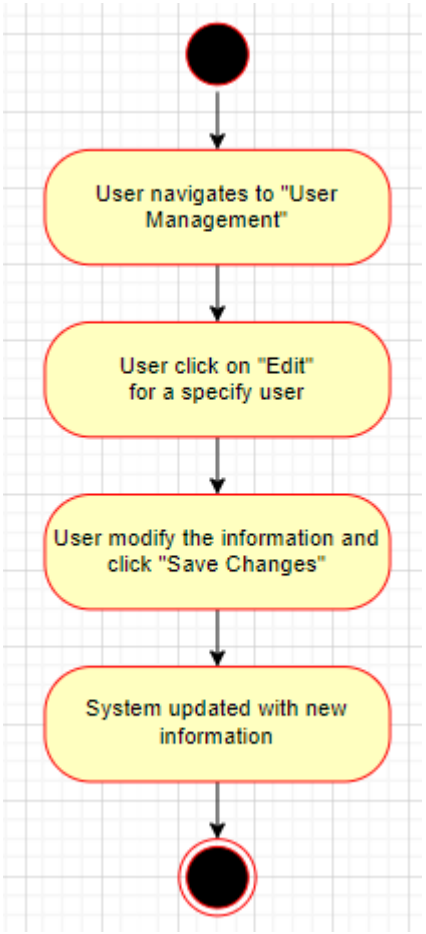
This user will use the User Management component to view the status and information of registered users.

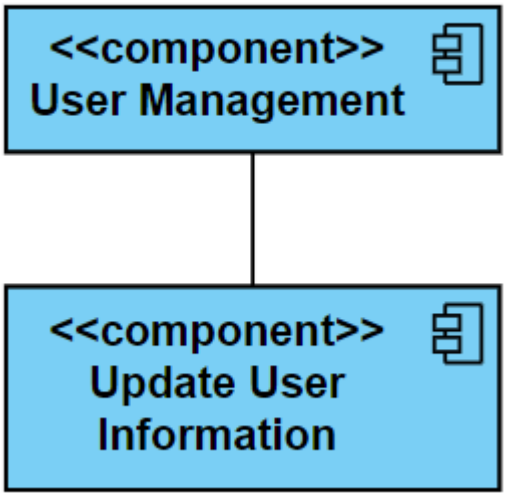




4.5.15 Update User Information(Admin)

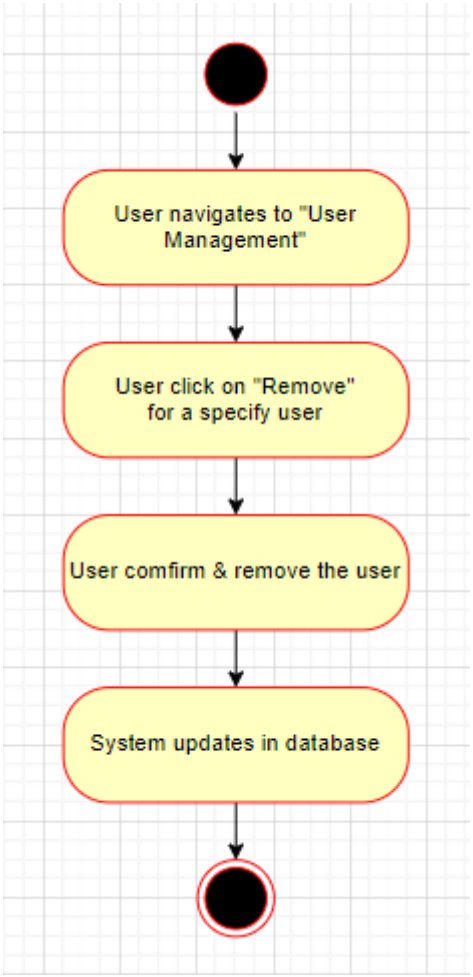
The user will use the Update User Information component to update the information of registered users.

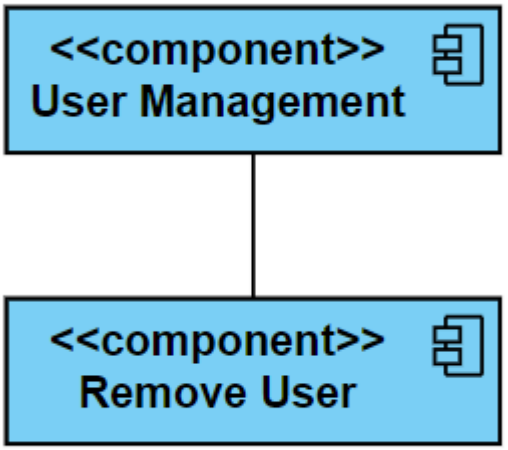




4.5.16 Remove User(Admin)

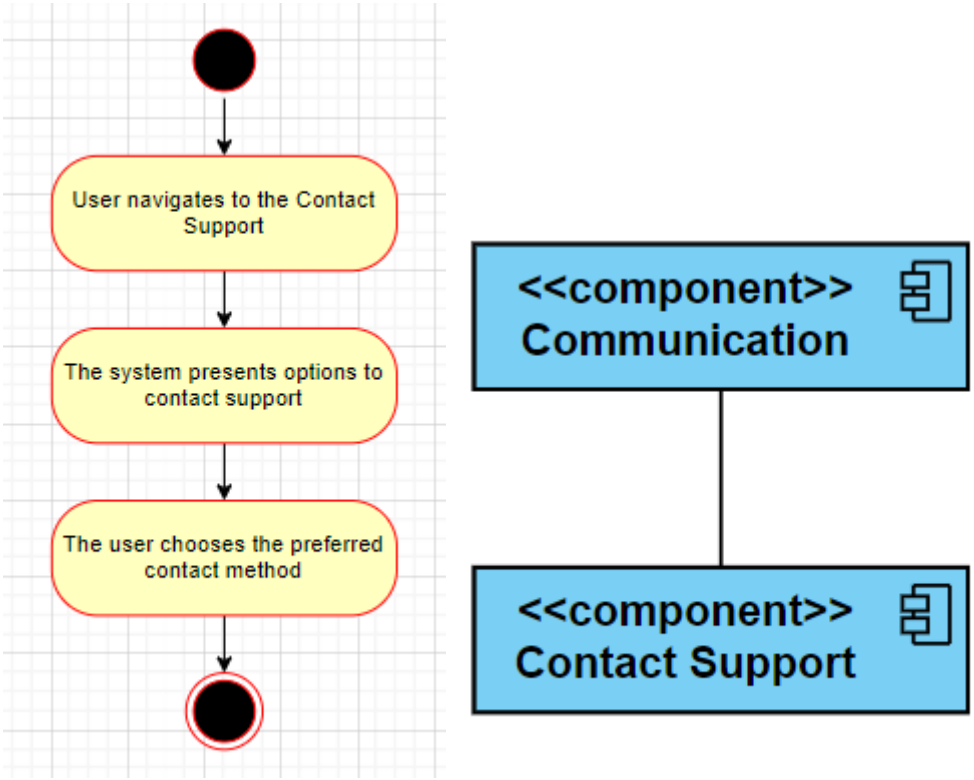
The user will use the Remove User Component to specific user that registered in the web app.





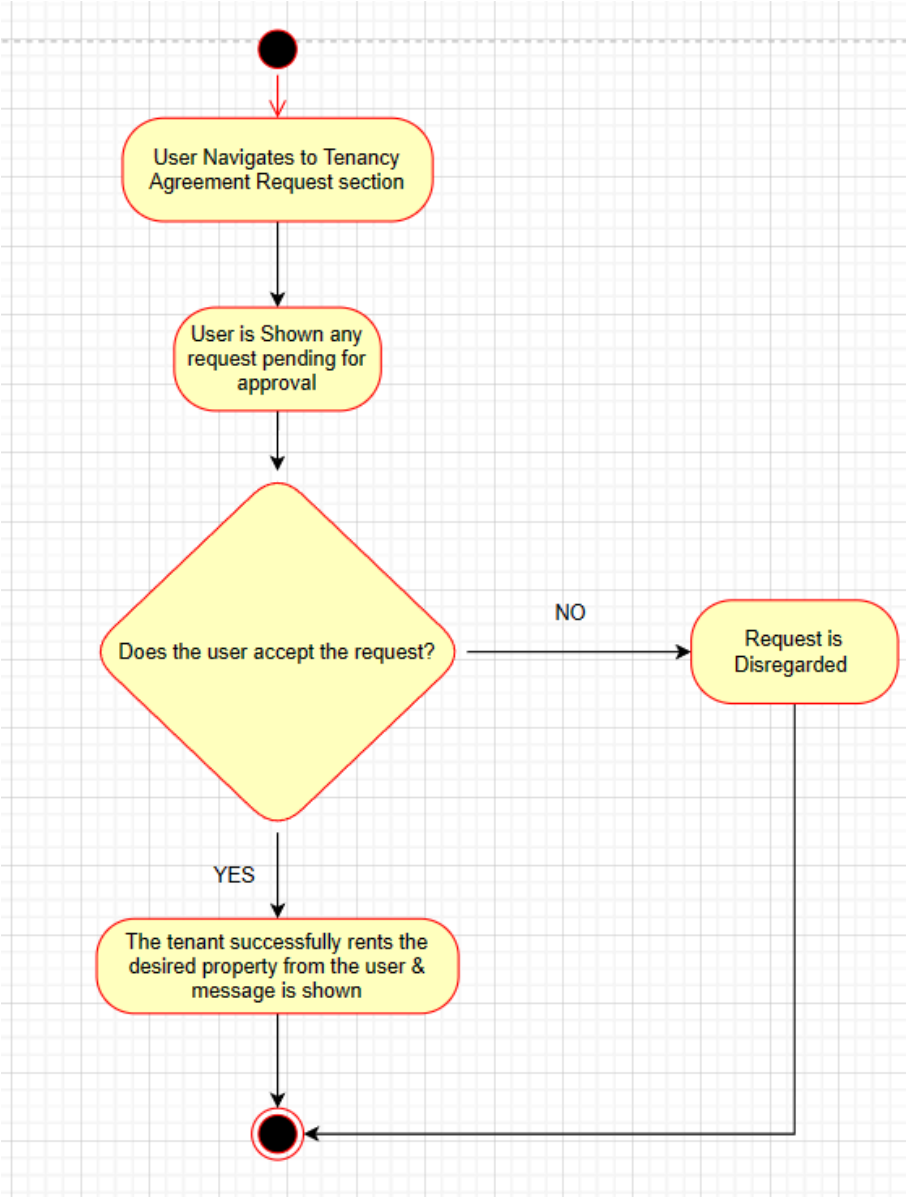
4.5.17 Contact Support(All actor)

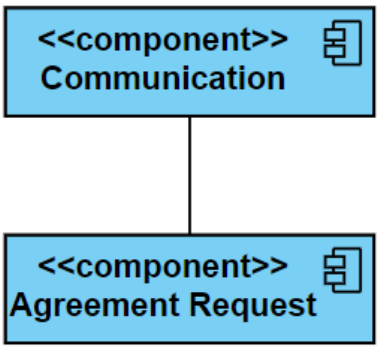
The user will use the Communication component to contact support for assistance.



4.5.18 Agreement Request (Owner/Agent)

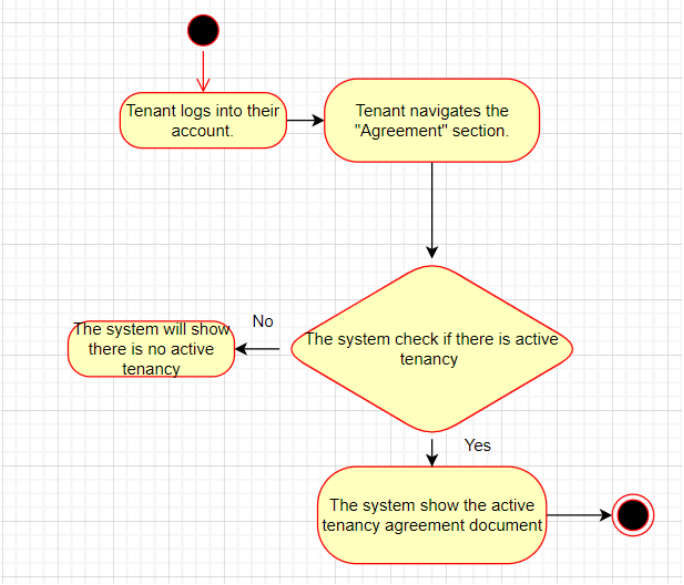
The user will use the Communication component to receive a tenancy agreement request sent by the tenant..

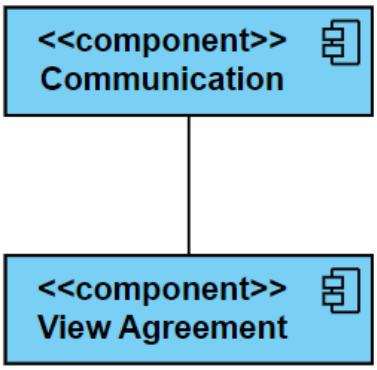

User Navigates to Tenancy Agreement Request section
User is Shown any request pending for approval
Does the user accept the request?
NO
Request is Disregarded
YES
The tenant successfully rents the desired property from the user & message is shown


<<component>>
Communication
<<component>>
Agreement Request

4.5.19 View Agreement(Tenant)

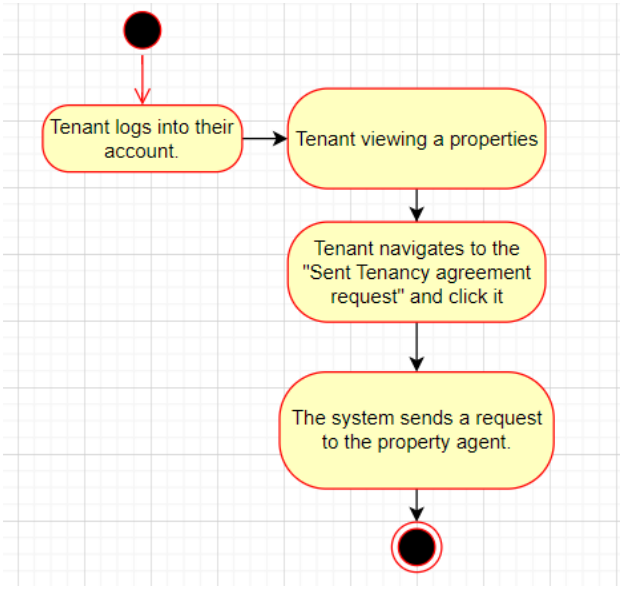
The user will use the Communication component to access and review the approved tenancy agreement for a specific property after the owner/agent accepts the agreement between them.

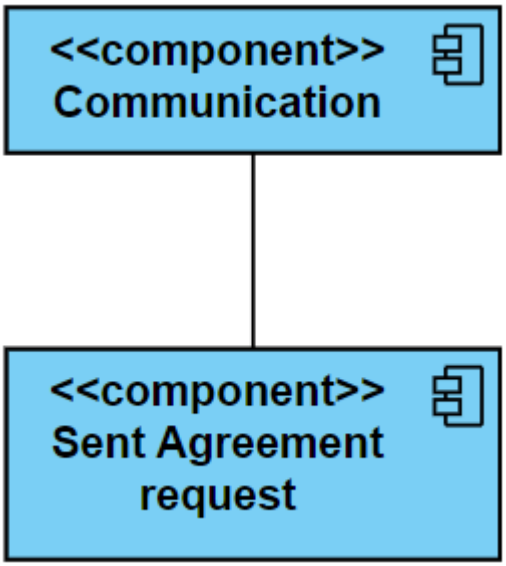




4.5.20 Sent Agreement Request (Tenant)

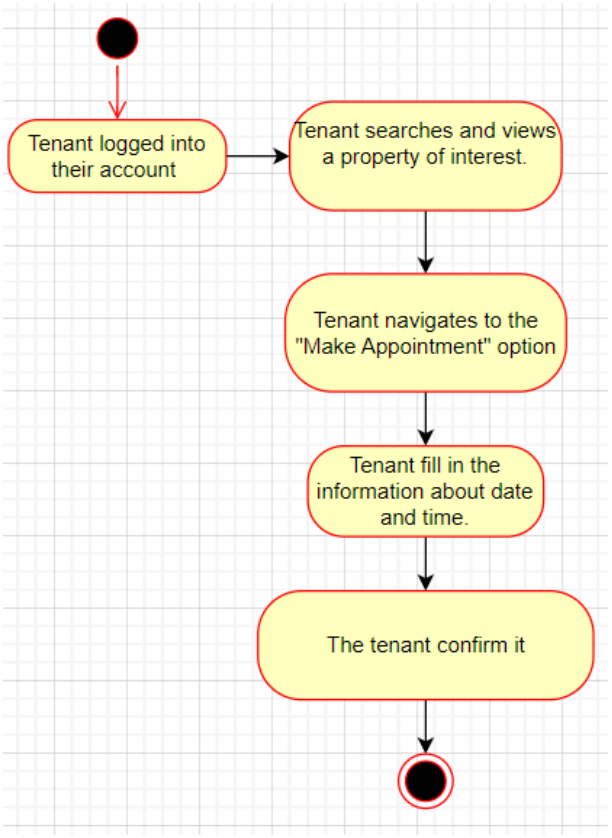
The user will use the Sent Agreement Request component to request access for tenancy agreement.

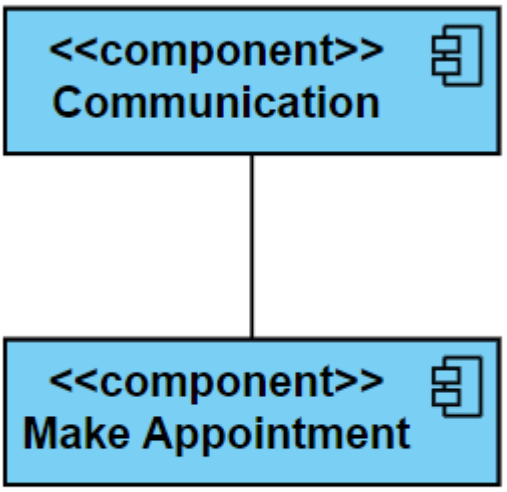




4.5.21 Make an Appointment (Tenant)

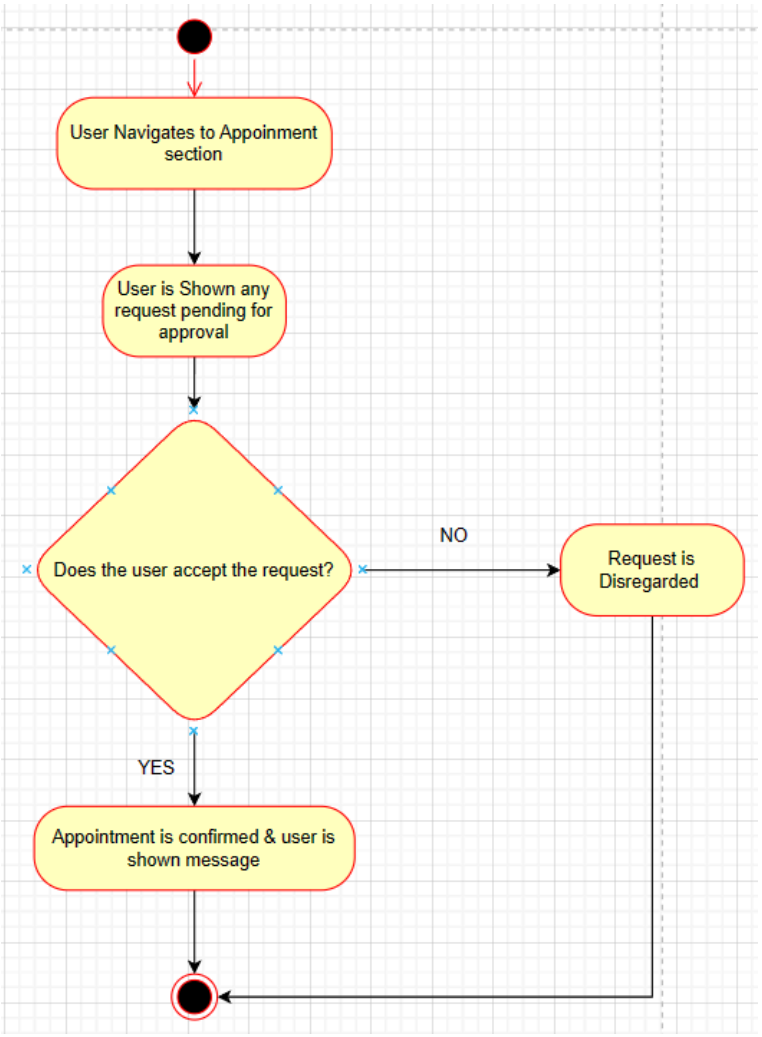
The user will use the Make an Appointment component to schedule appointments with the property owner or agent to view a specific property of interest.

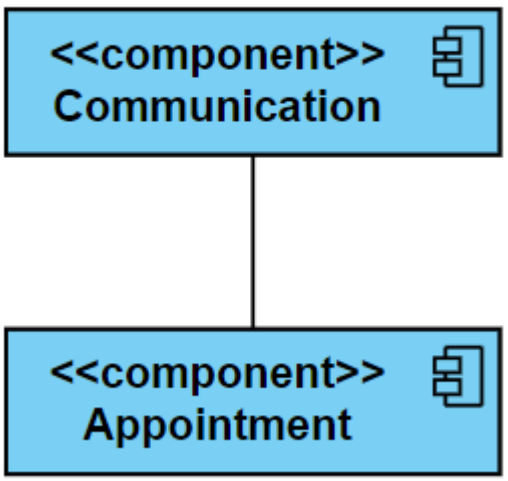




4.5.22 Appointment (Owner/Agent)

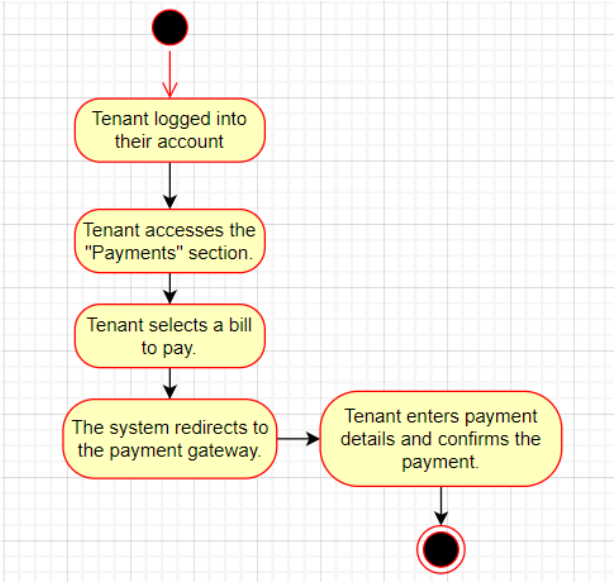
The user will use the Communication component to make a decision for an appointment request sent by the tenant, to view a specific property of interest.

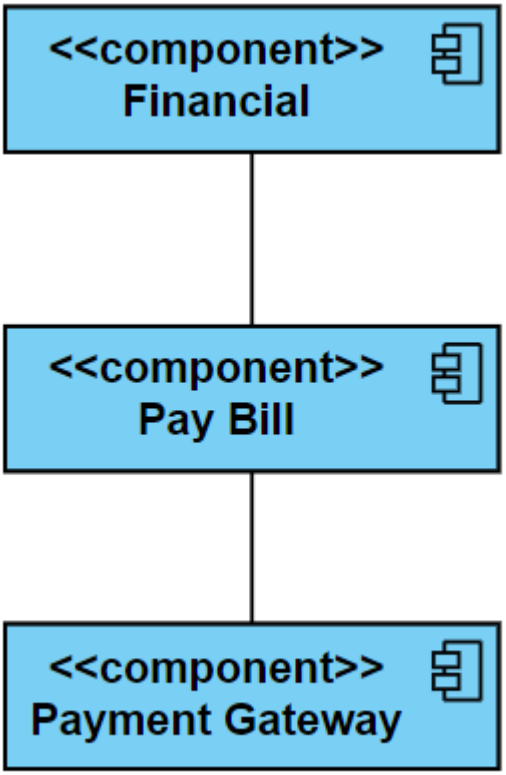




4.5.23 Pay Bill

The user will use the Financial Component to securely make payments for bills using the integrated payment gateway.

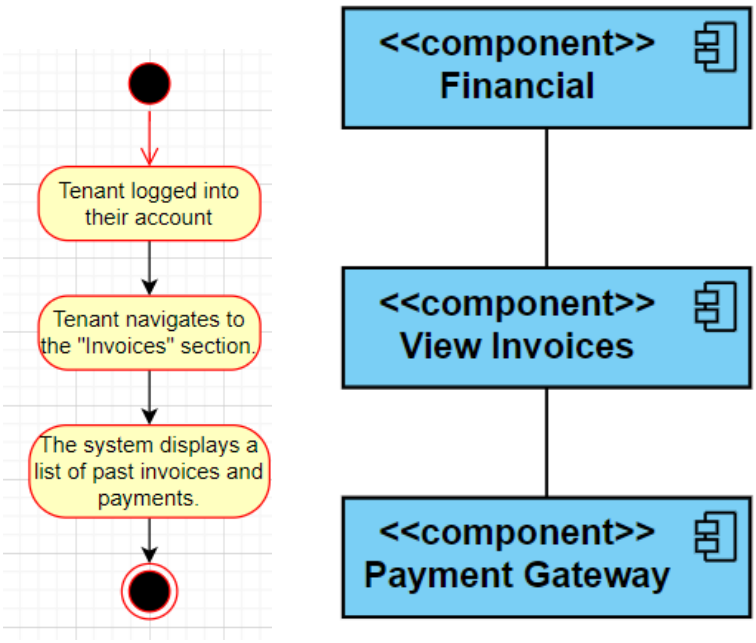




4.5.24 View Invoices

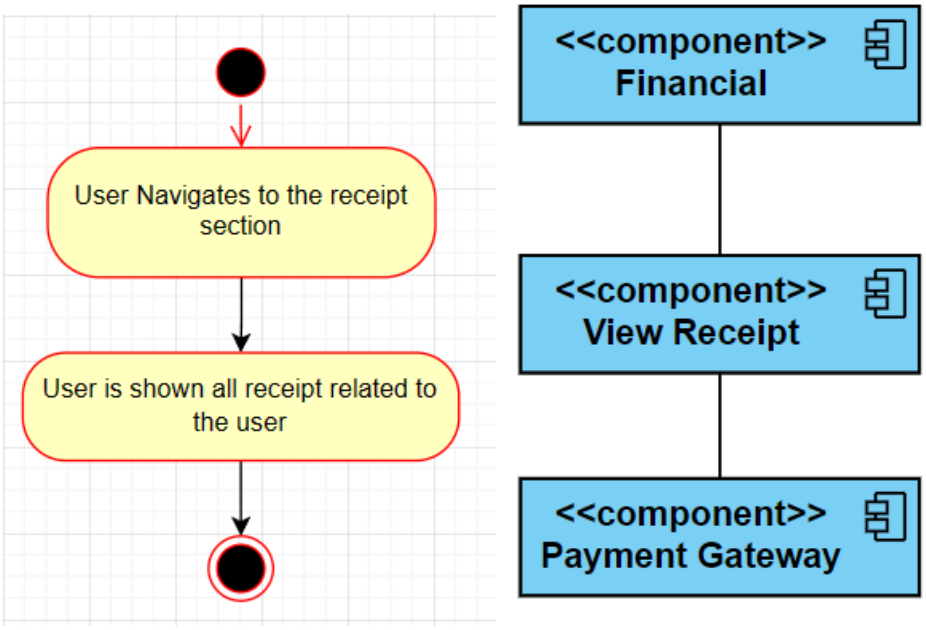
The user will use the Financial component to access and review past invoices associated

with their tenancy.



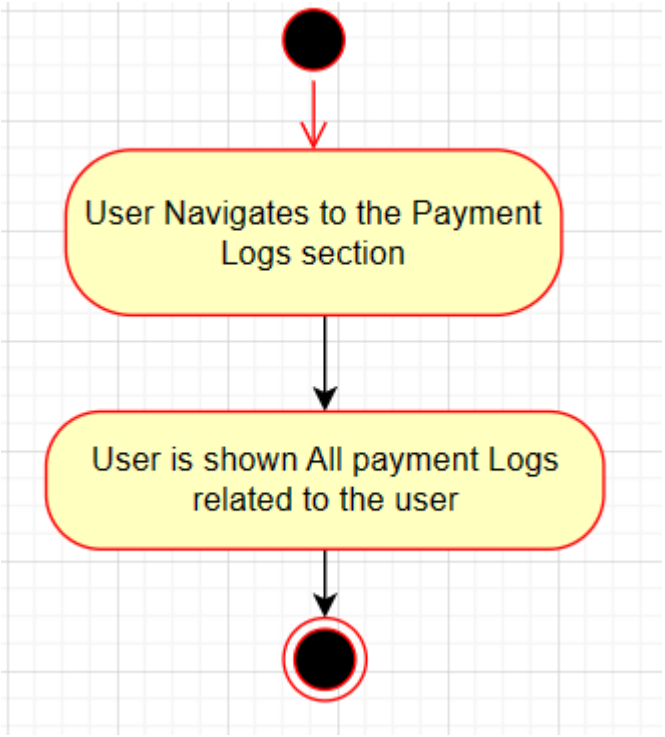
4.5.25 View Receipt

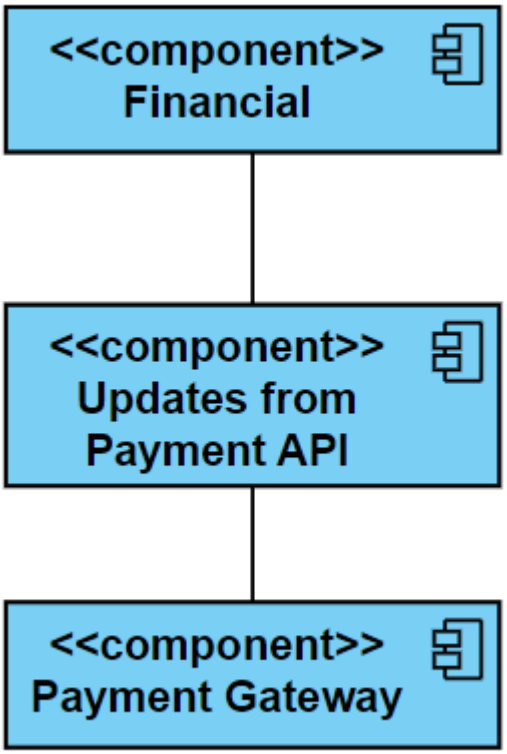
The user will use the Financial component to view review receipts of their past payments.



4.5.26 Updates from Payment API

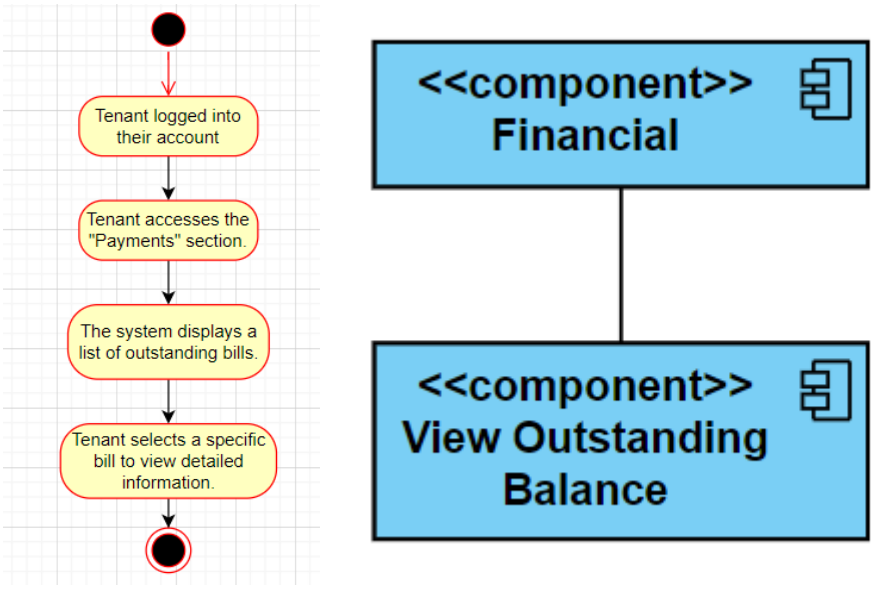
The user will get Updates from the Payment API component to view payment logs from the Payment Gateway system.





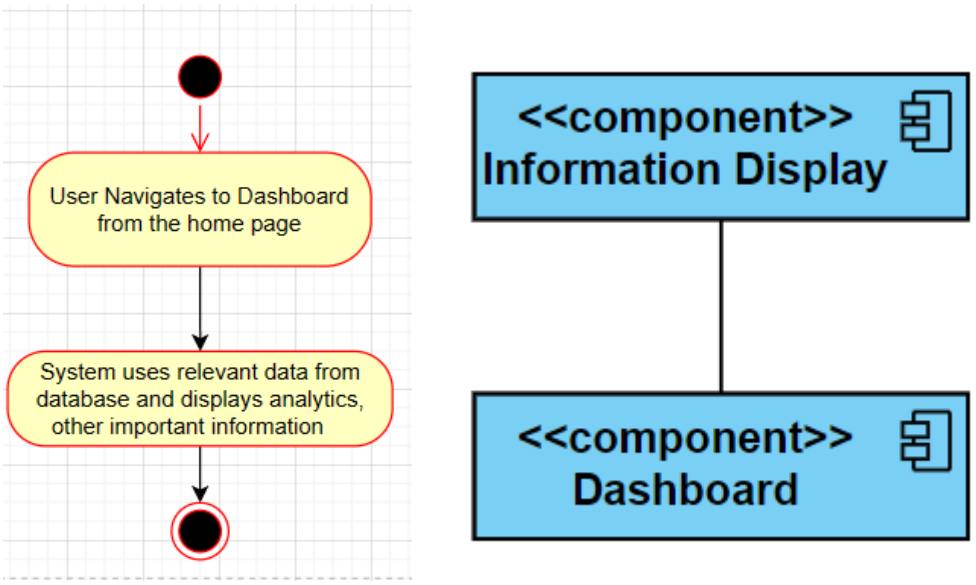
4.5.27 View Outstanding Balance

The user will use the Financial component to access and review details regarding any outstanding balance with their tenancy.

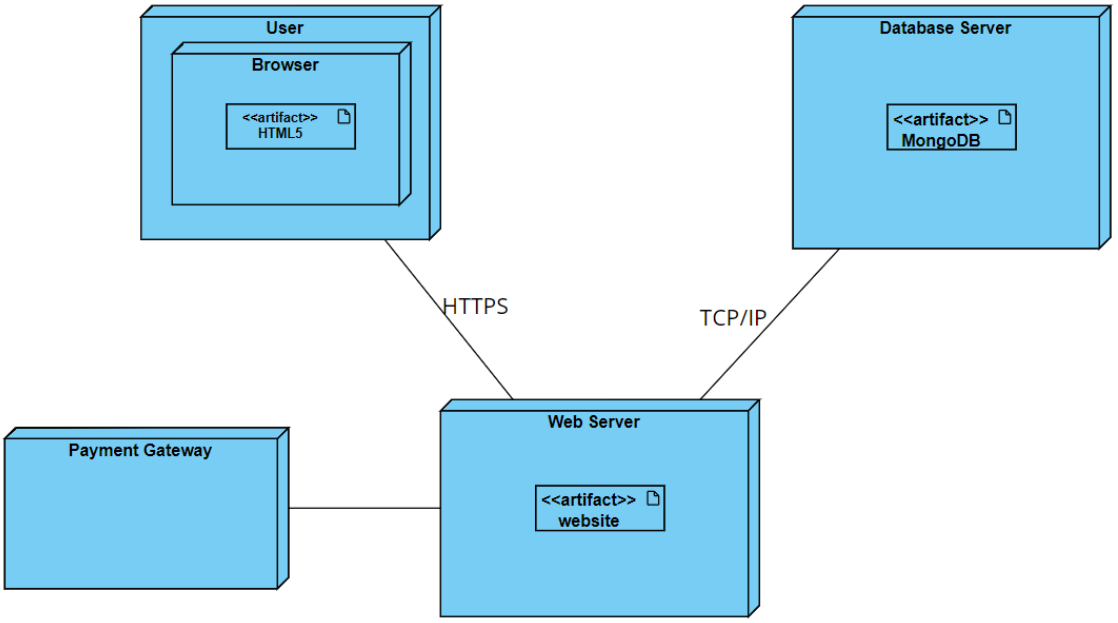


4.5.28 Dashboard

The user will use the Information Display component to view relevant information associated with their account.



4.6 Deployment Diagram



There is a web server, a database server, and the machine where the user views the website.

The payment gateway is a separate service that is used to process online payments. When a user makes a payment on the website, their payment information is sent to the payment gateway, which then processes the payment and sends the results back to the website.

User Node:
Components: This node represents the end-user perspective, equipped with a web browser (like Chrome, Firefox, Safari, etc.).
Artifact: It holds an HTML5-based artifact, which could be a web application, a set of web pages, or any content developed using HTML5 technology. HTML5 provides a versatile framework for creating interactive and multimedia-rich content that can be accessed through web browsers.

Web Server Node:
Components: This node hosts and serves the web content to users accessing the application or website.
Artifact: The web server node holds the website or web application, composed of HTML, CSS, NodeJS(Javascript).

Database Server Node:
Components: This node primarily manages and stores the data required by the web application.
Artifact: The database server node hosts MongoDB, a NoSQL database solution. MongoDB is designed for flexible and scalable data storage. It stores structured data in a

document-oriented format and is commonly used for various types of applications requiring scalable and efficient data handling.

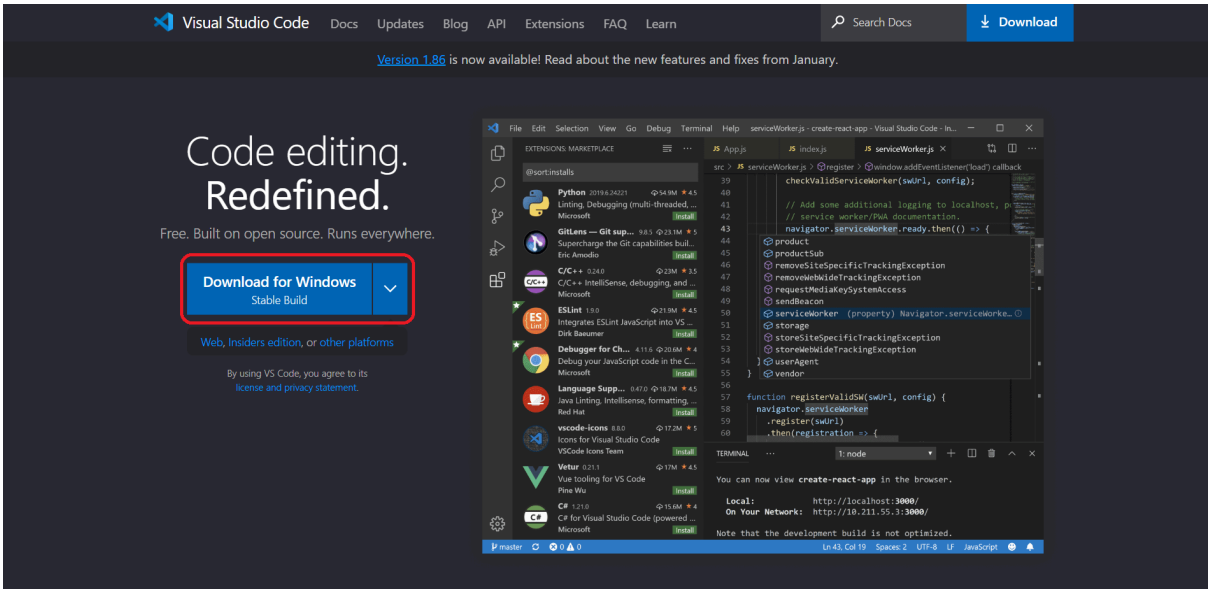
5 Implementation

5.1 Development Environment

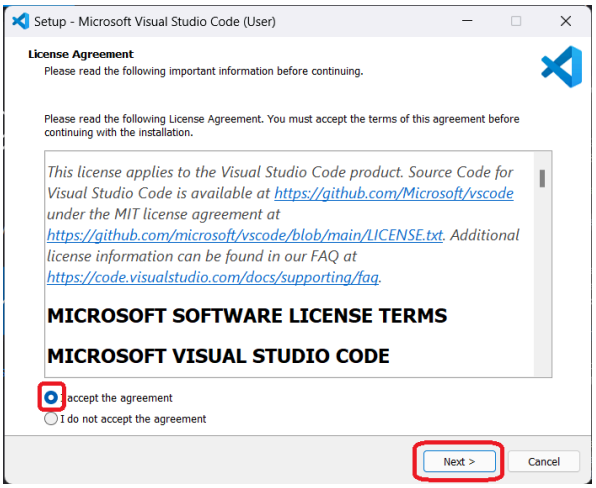
<TO DO: Describe the tools and programming elements here. Place screenshots of the IDE and solution explorer or project folder contents to illustrate the completed software.>

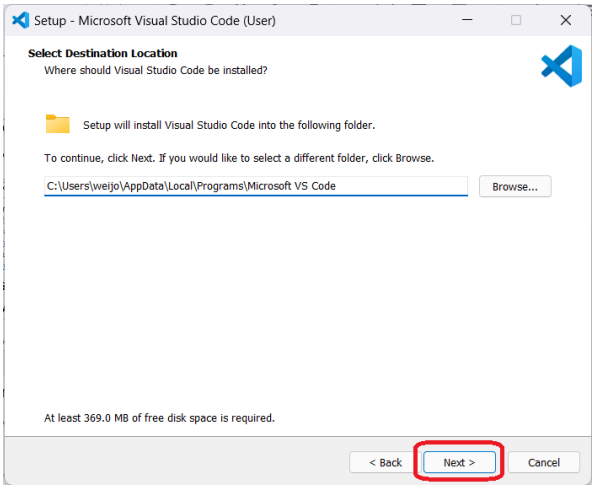
Tools: Visual Studio Code, MongoDB, Node.js, Google Chrome(browser view)
Programming elements: HTML, CSS, Javascript

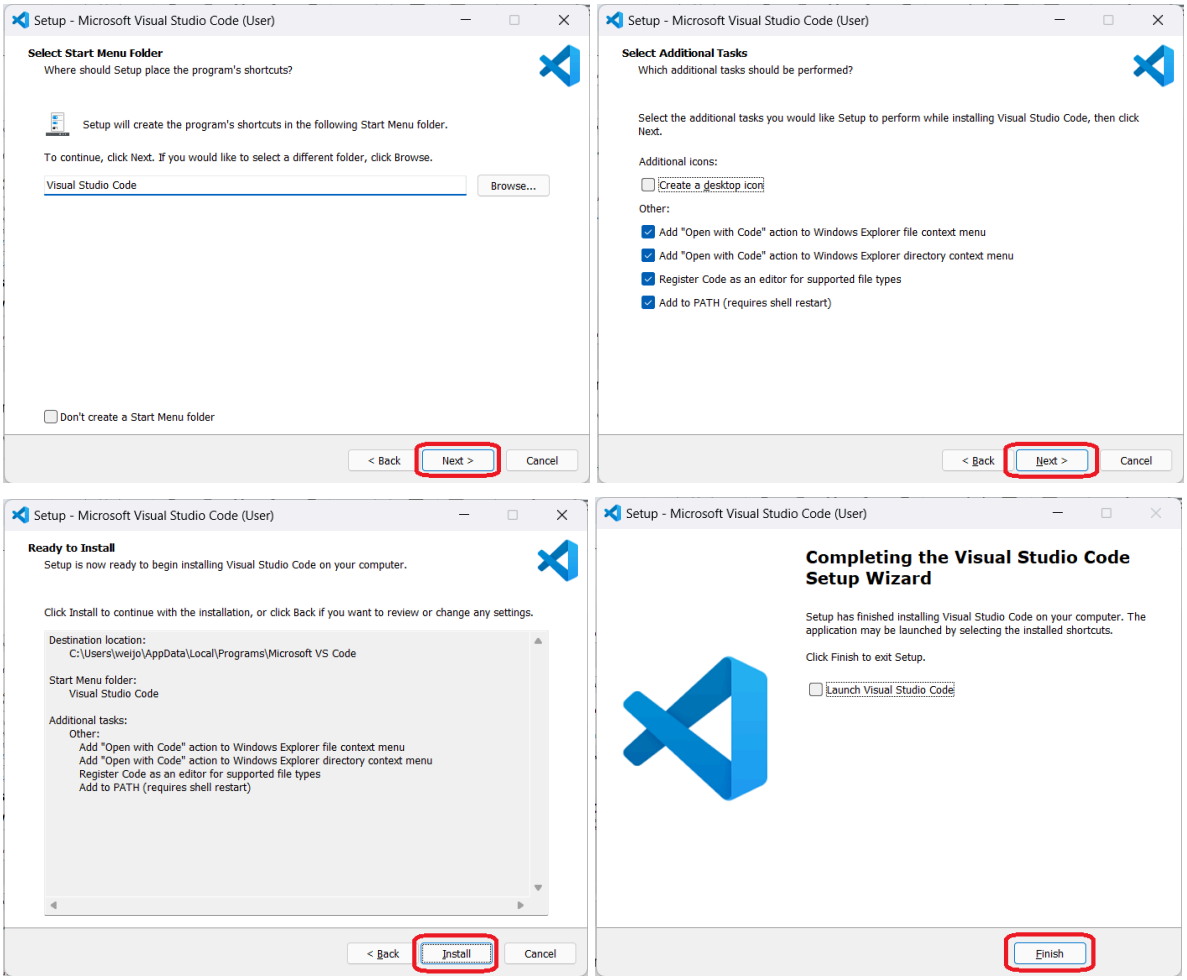
Download Visual Studio Code at https://code.visualstudio.com/



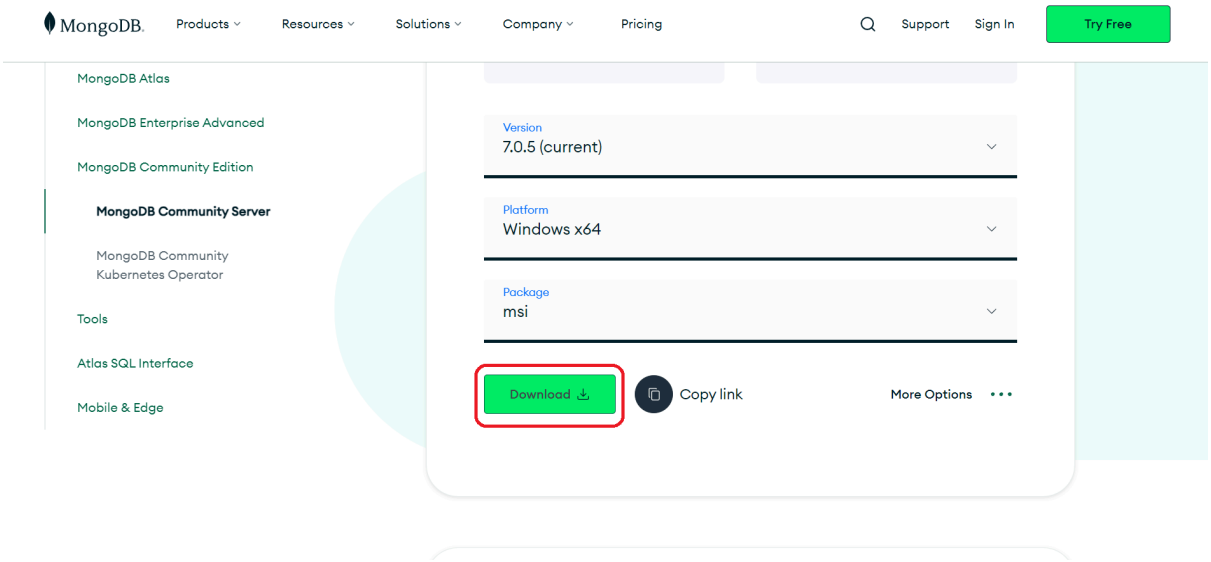
Follow the steps below to complete the installation:



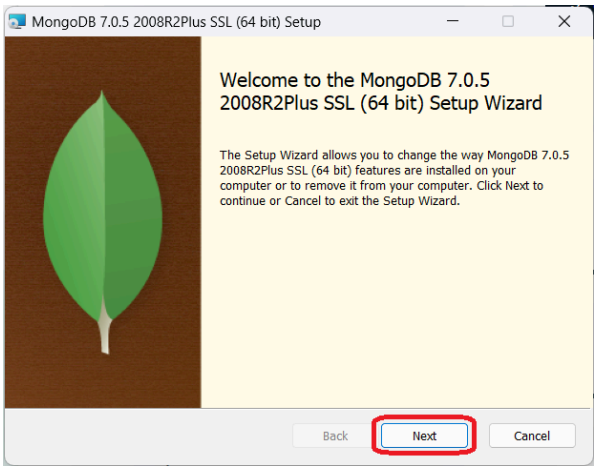


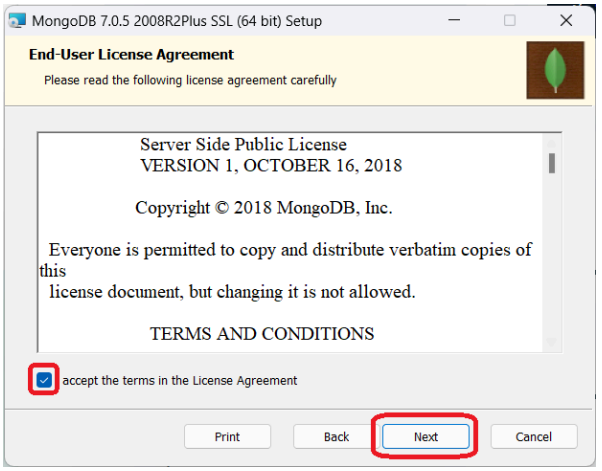


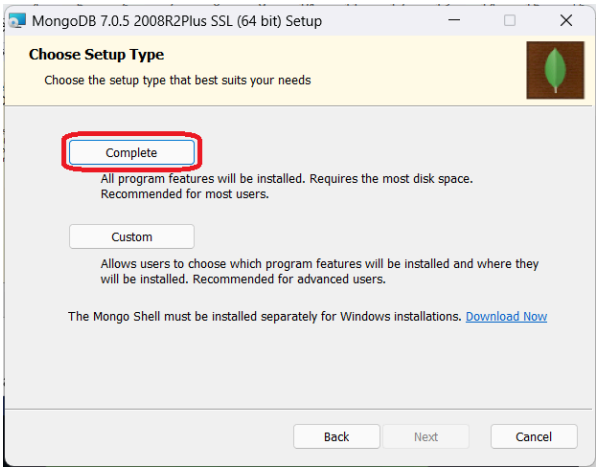
Download MongoDB at https://www.mongodb.com/try/download/community

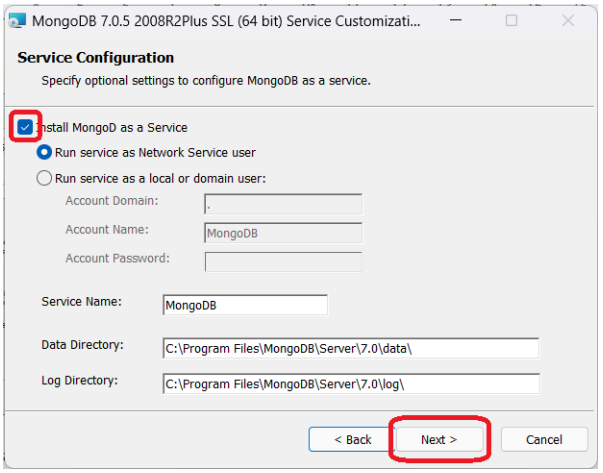


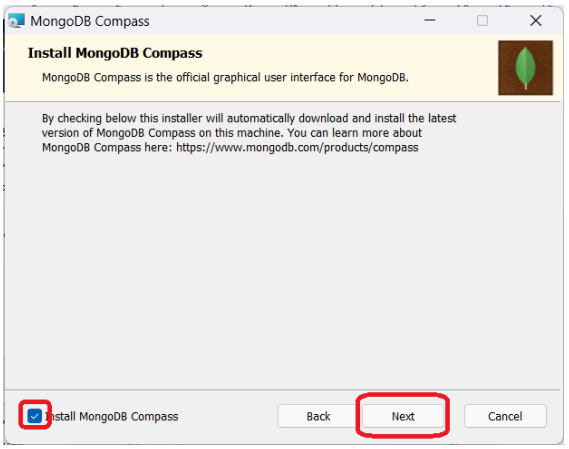
Follow the steps below to complete the installation:

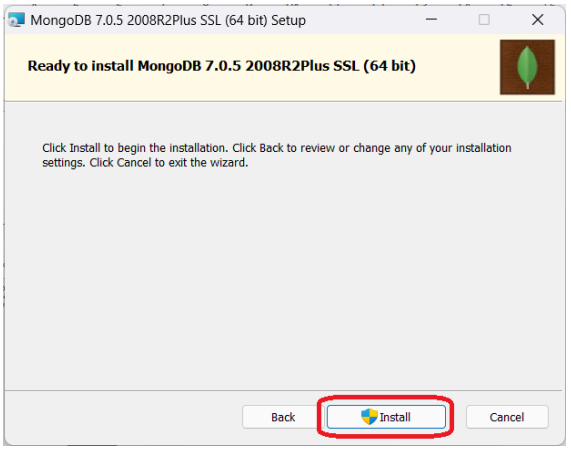

MongoDB 7.0.5 2008R2Plus SSL (64 bit) Setup
Welcome to the MongoDB 7.0.5 2008R2Plus SSL (64 bit) Setup Wizard
The Setup Wizard allows you to change the way MongoDB 7.0.5 2008R2Plus SSL (64 bit) features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel

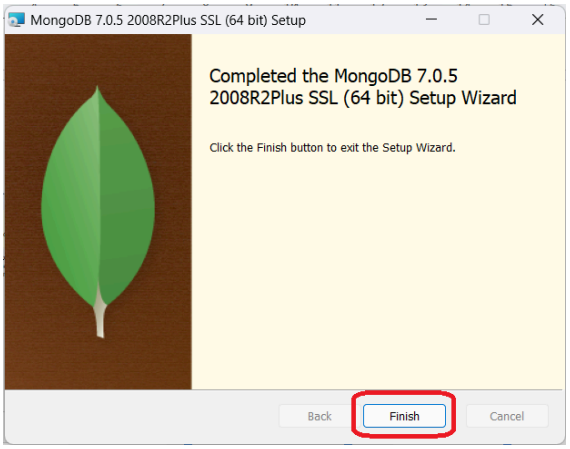

MongoDB 7.0.5 2008R2Plus SSL (64 bit) Setup
End-User License Agreement
Please read the following license agreement carefully
Server Side Public License
VERSION 1, OCTOBER 16, 2018
Copyright © 2018 MongoDB, Inc.
Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.
TERMS AND CONDITIONS
I accept the terms in the License Agreement
Print
Back
Next
Cancel


MongoDB 7.0.5 2008R2Plus SSL (64 bit) Setup
Choose Setup Type
Choose the setup type that best suits your needs
Complete
All program features will be installed. Requires the most disk space.
Recommended for most users.
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.
The Mongo Shell must be installed separately for Windows installations. Download Now
Back
Next
Cancel

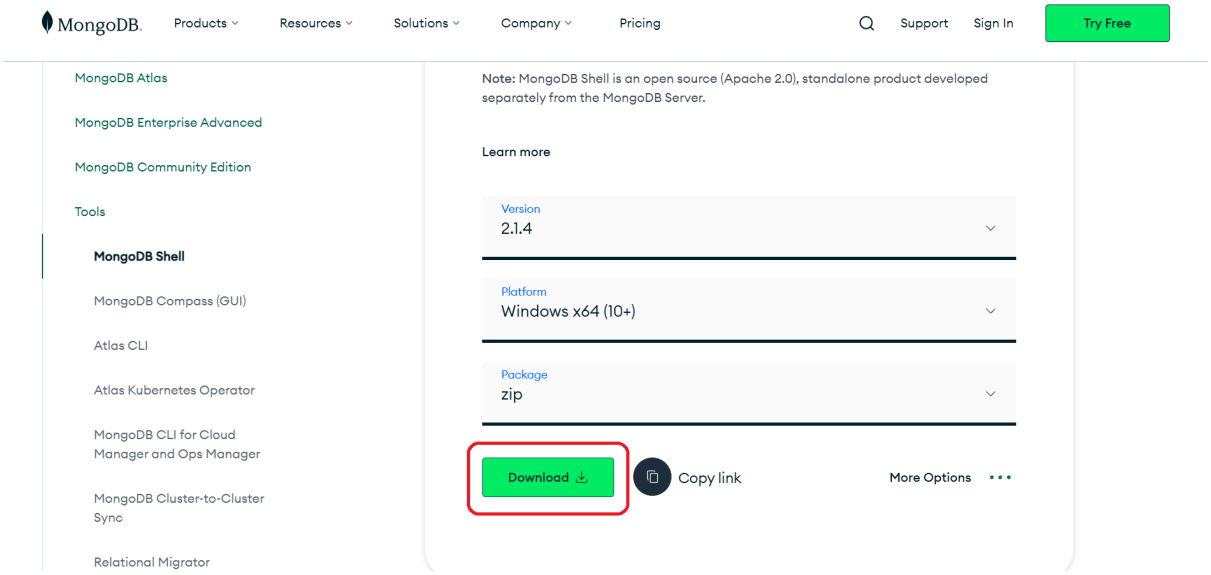

MongoDB 7.0.5 2008R2Plus SSL (64 bit) Service Customizati...
Service Configuration
Specify optional settings to configure MongoDB as a service.
Install MongoD as a Service
Run service as Network Service user
Run service as a local or domain user:
Account Domain:
Account Name:
MongoDB
Account Password:
Service Name:
MongoDB
Data Directory:
C:\Program Files\MongoDB\Server\7.0\data\
Log Directory:
C:\Program Files\MongoDB\Server\7.0\log\
< Back
Next >
Cancel


MongoDB Compass
Install MongoDB Compass
MongoDB Compass is the official graphical user interface for MongoDB.
By checking below this installer will automatically download and install the latest version of MongoDB Compass on this machine. You can learn more about MongoDB Compass here: https://www.mongodb.com/products/compass
Install MongoDB Compass
Back
Next
Cancel

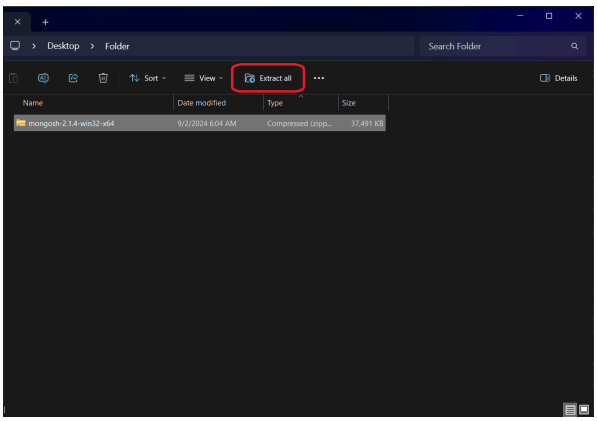

MongoDB 7.0.5 2008R2Plus SSL (64 bit) Setup
Ready to install MongoDB 7.0.5 2008R2Plus SSL (64 bit)
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel

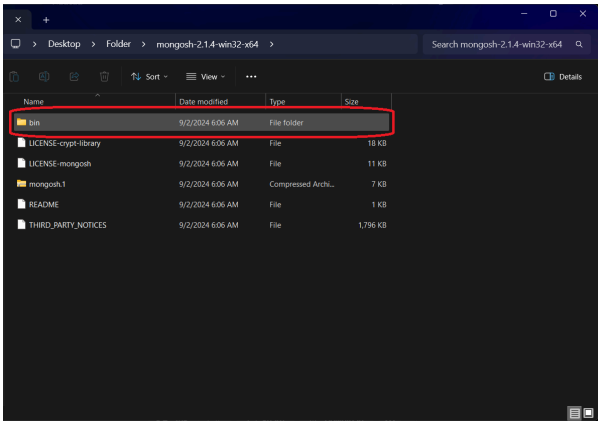


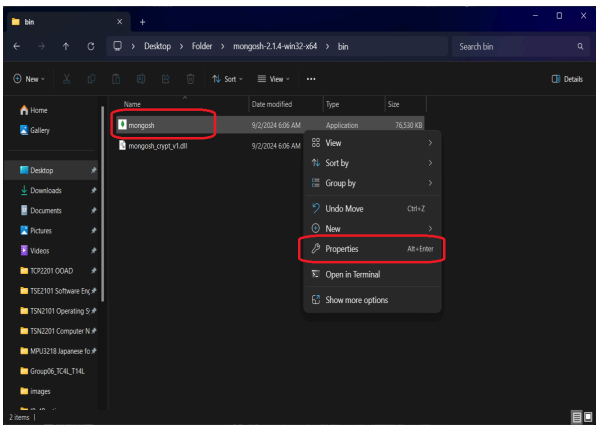
Download MongoDB Shell at https://www.mongodb.com/try/download/shell

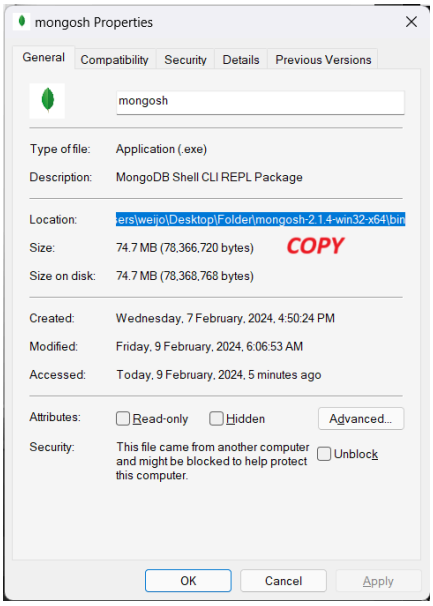


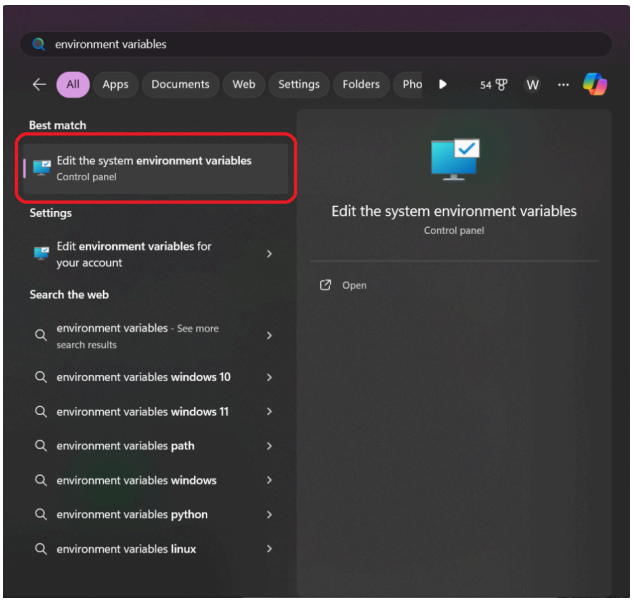
Follow the steps below to after the installation is done:

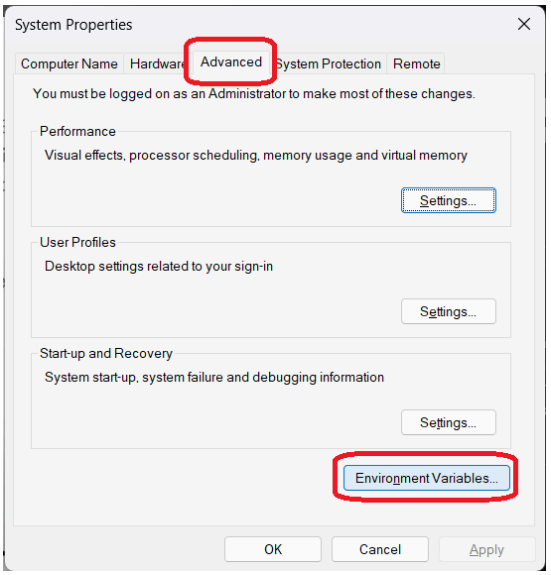


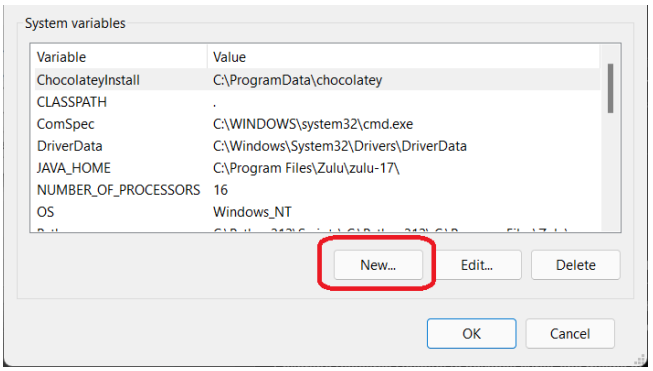


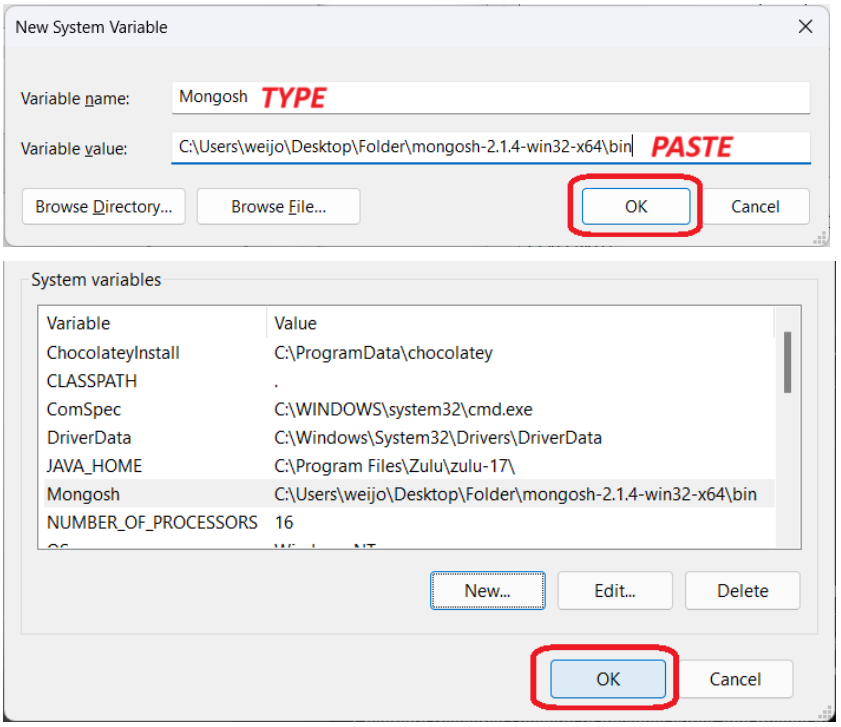

bin
Desktop › Folder › mongosh-2.1.4-win32-x64 › bin
Search bin
mongosh
mongosh_crypt_v1.dll
Properties
Open in Terminal
Show more options


mongosh Properties
General
Compatibility
Security
Details
Previous Versions
mongosh
Type of file: Application (.exe)
Description: MongoDB Shell CLI REPL Package
Location: ers\weijo\Desktop\Folder\mongosh-2.1.4-win32-x64\bin
Size: 74.7 MB (78,366,720 bytes)
COPY
Size on disk: 74.7 MB (78,368,768 bytes)
Created: Wednesday, 7 February, 2024, 4:50:24 PM
Modified: Friday, 9 February, 2024, 6:06:53 AM
Accessed: Today, 9 February, 2024, 5 minutes ago
Attributes: Read-only Hidden Advanced...
Security: This file came from another computer and might be blocked to help protect this computer. Unblock
OK
Cancel
Apply

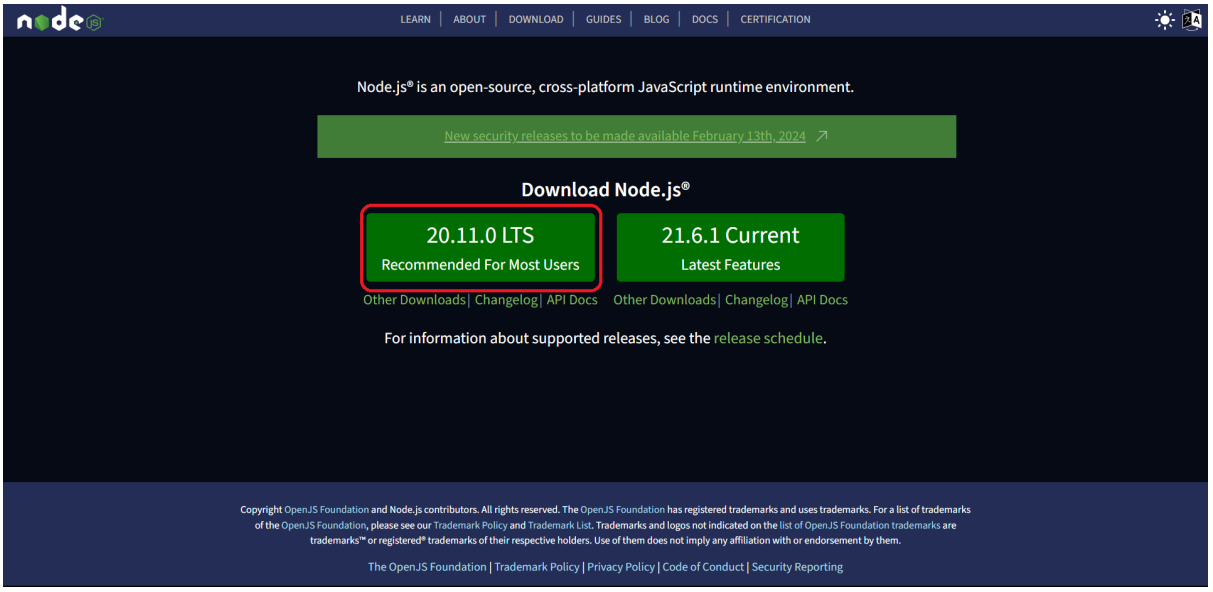

environment variables
All
Apps
Documents
Web
Settings
Folders
Best match
Edit the system environment variables
Control panel
Settings
Edit environment variables for your account
Search the web
environment variables - See more search results
environment variables windows 10
environment variables windows 11
environment variables path
environment variables windows
environment variables python
environment variables linux
Edit the system environment variables
Control panel
Open


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Start-up and Recovery
System start-up, system failure and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

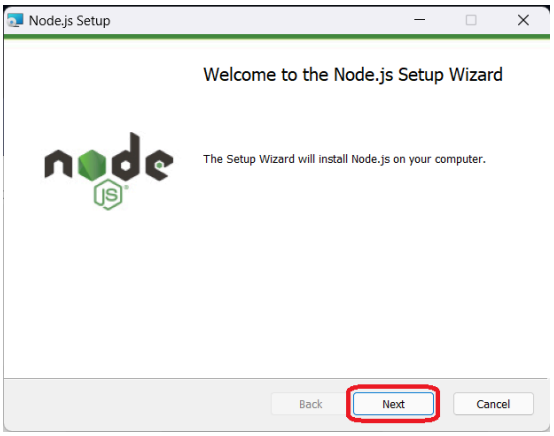

System variables
Variable
Value
ChocolateyInstall
C:\ProgramData\chocolatey
CLASSPATH
.
ComSpec
C:\WINDOWS\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
JAVA_HOME
C:\Program Files\Zulu\zulu-17\
NUMBER_OF_PROCESSORS
16
OS
Windows_NT
New...
Edit...
Delete
OK
Cancel

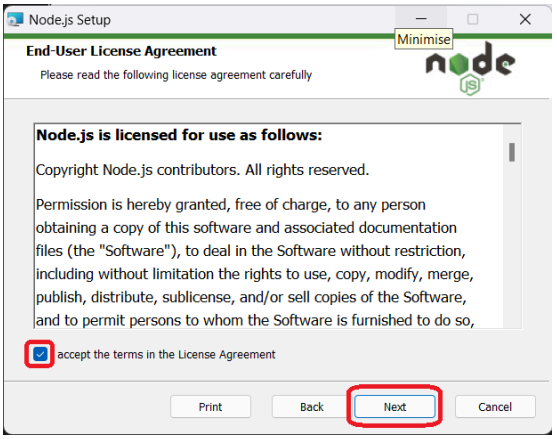


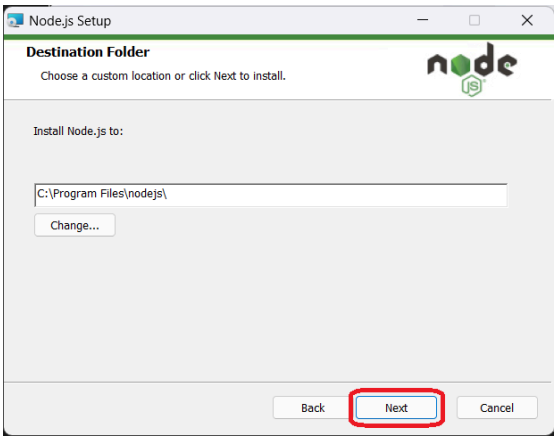
Download Node.js at https://nodejs.org/en

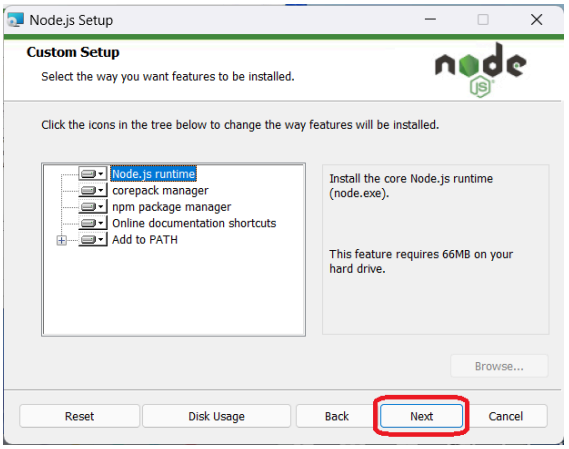


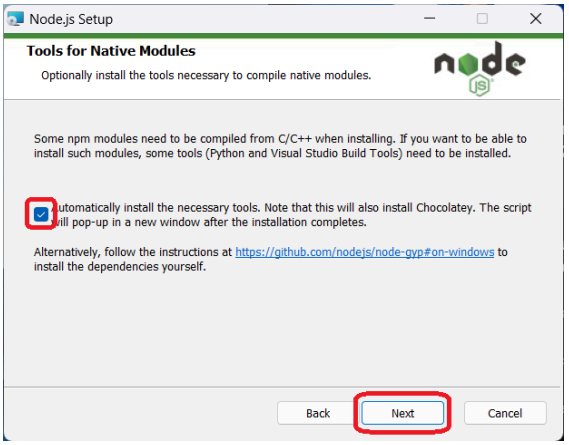
Follow the steps below to complete the installation:

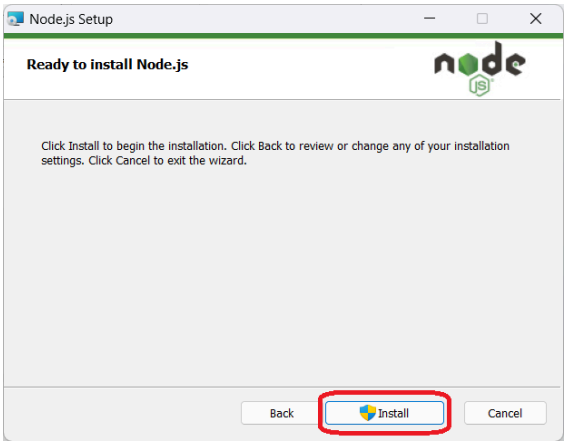

Node.js Setup
Welcome to the Node.js Setup Wizard
The Setup Wizard will install Node.js on your computer.
Back
Next
Cancel

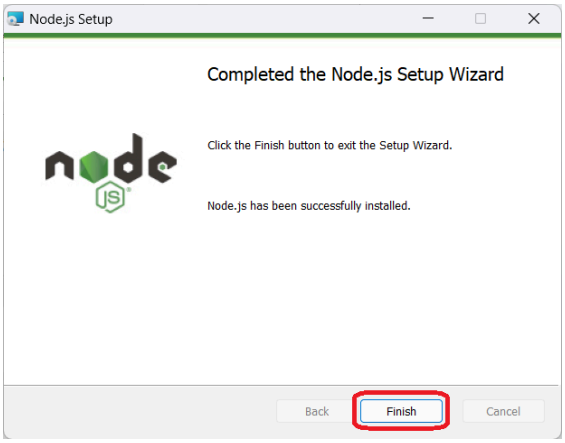

Node.js Setup
Minimise
End-User License Agreement
Please read the following license agreement carefully
Node.js is licensed for use as follows:
Copyright Node.js contributors. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so,
accept the terms in the License Agreement
Print
Back
Next
Cancel


Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files\nodejs\
Change...
Back
Next
Cancel

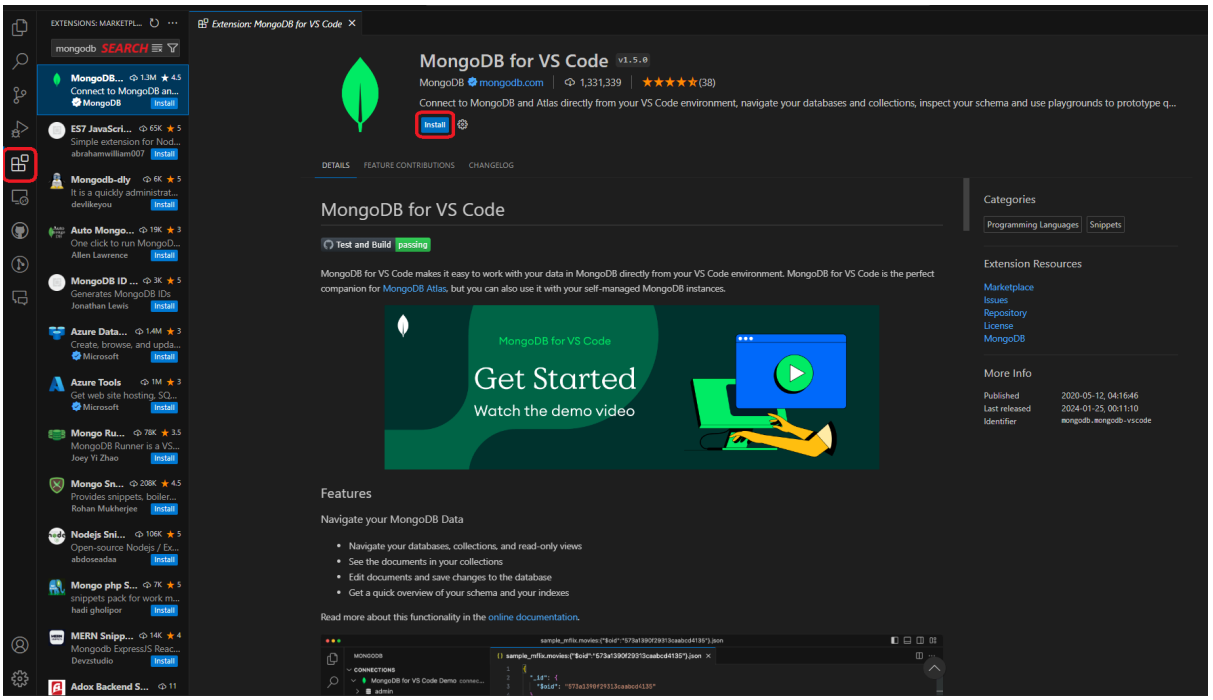

Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
corepack manager
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature requires 66MB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel


Node.js Setup
Tools for Native Modules
Optionally install the tools necessary to compile native modules.
Some npm modules need to be compiled from C/C++ when installing. If you want to be able to install such modules, some tools (Python and Visual Studio Build Tools) need to be installed.
utomatically install the necessary tools. Note that this will also install Chocolatey. The script ill pop-up in a new window after the installation completes.
Alternatively, follow the instructions at https://github.com/nodejs/node-gyp#on-windows to install the dependencies yourself.
Back
Next
Cancel

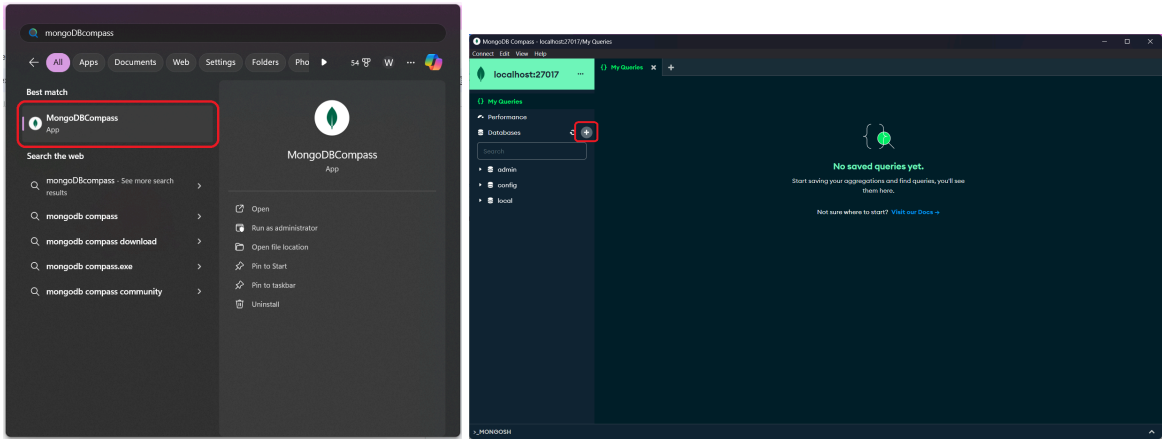

Node.js Setup
Ready to install Node.js
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel

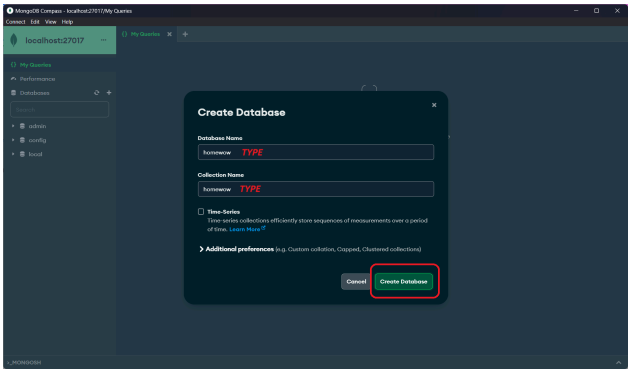


Once everything is installed, launch Visual Studio Code

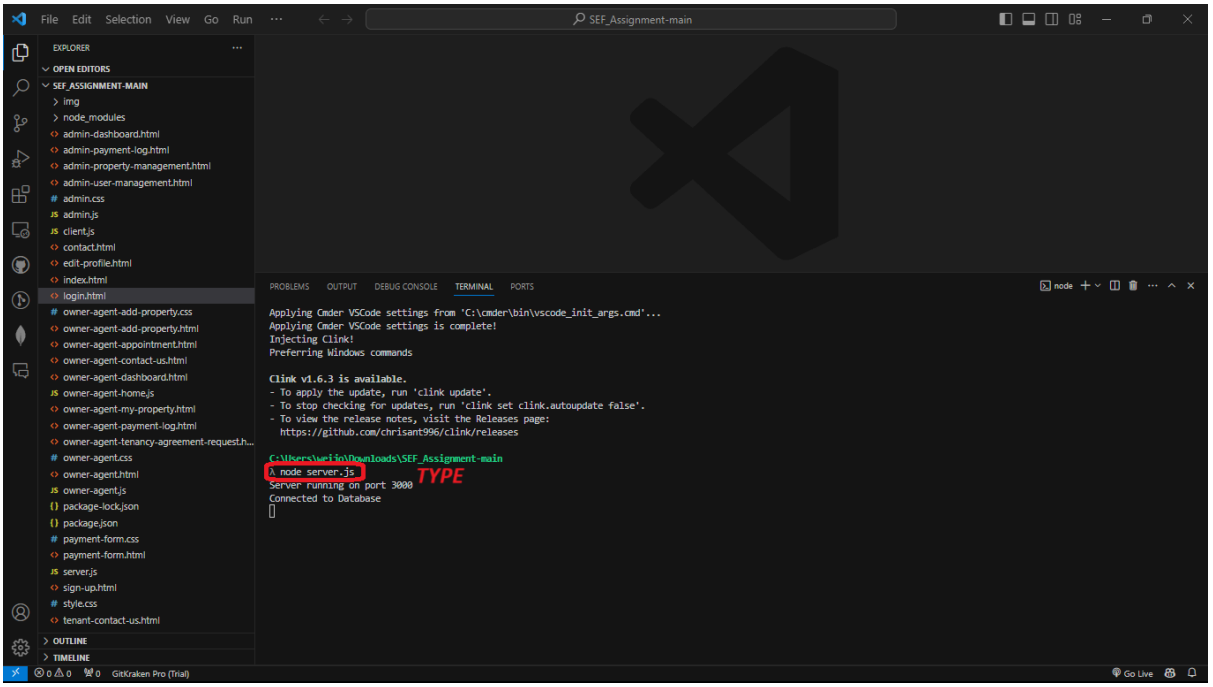


launch MongoDB Compass

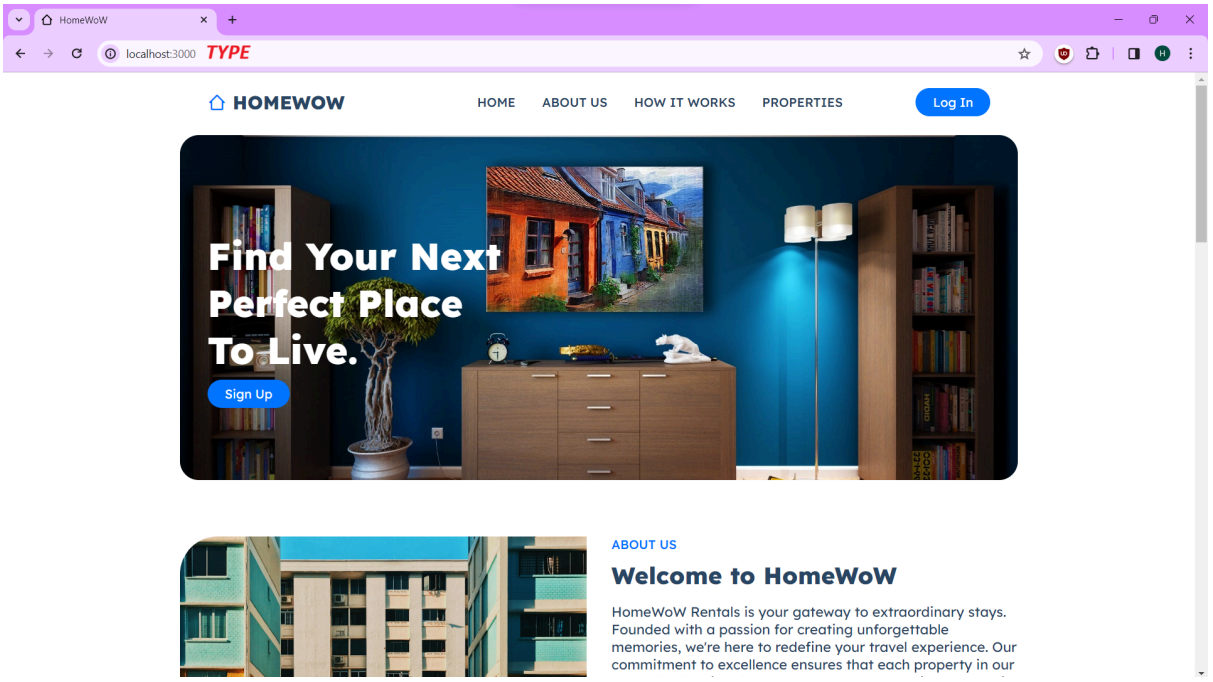




After that, launch Visual Studio code



Once connected, launch Google Chrome (browser)



Done, can start using the webapp.

5.2 Software Integration

5.2.1 Login

Main Files:	Description
1. login.html 2. server.js	The integration of the login functionality involves combining the frontend HTML files with the backend server logic to enable user authentication. This process ensures that users can securely log in to the web application using their credentials.

src:

```
app.post('/login', (req, res) => {
  const { email, password } = req.body;

  UserModel.findOne({ email: email })
    .then(user => {
      if (!user || password !== user.password) { // Note: In a real
application, never store passwords in plain text
        res.redirect('/login.html?loginError=Invalid email or password');
      } else {
        const loggedInUser = new User(user._id, user.name, user.email,
user.userType);
```

```
        req.session.user = loggedInUser;
        console.log(loggedInUser);

        // Check the userType and redirect accordingly
        if (user.userType === 'tenant') {

res.redirect(`/tenant.html?username=${user.name}&userType=${user.userType}&id
=${user._id}`);
        }
        else if (user.userType === 'Owner/Agent') {

res.redirect(`/owner-agent.html?username=${user.name}&userType=${user.userTyp
e}&id=${user._id}`);
        }
        else if (user.userType === 'Admin'){

res.redirect(`/admin-dashboard.html?username=${user.name}&userType${user.user
Type}&userId=${user.id}`);
        }
      }
    })
    .catch(error => console.error(error));
});
```

5.2.2 Signup

Main Files:	Description
• sign-up.html.html • server.js	The signup integration seamlessly combines frontend and backend processes. The signup form in sign-up.html collects user details, and upon submission, a POST request is sent to "/signup" on the server. In server.js, using Node.js, the backend validates and stores user data, allowing successful user registration.

Src:

```
app.post('/signup', (req, res) => {
  const { name, password, email, icNum, phone, gender, userType } =
req.body;

  const newUser = new UserModel({ name, password, email, icNum, phone,
gender, userType });
```

```
  newUser.save()
    .then(() => res.redirect('/login.html'))
    .catch(error => console.error(error));
});
```

5.2.3 Log out

Main Files:	Description
• index.html • server.js	The logout integration enhances user experience on the system. In server.js, a GET request at "/logout" is implemented to clear the user's session or token, facilitating a secure logout. Within index.html, a logout button prompts a request to "/logout" when clicked. Upon logout, users seamlessly return to the main page, ensuring a smooth transition and enhancing overall usability in the web application.

Src:

```
app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if(err) {
      return console.log(err);
    }
    console.log("Logout successful");
    res.redirect('/index.html');
  });
});
```

5.2.4 Edit Profile

Main Files:	Description
• edit-profile.html • server.js • client.js	The edit profile integration in web app seamlessly connects the frontend and backend to empower users with the ability to update their profile information. Utilizing

	edit-profile.html for the frontend and server.js for the backend, the integration begins by fetching the current user's details via a GET request to "/current-user." On the backend, a POST request to "/update-profile" is handled by server.js, updating the user's profile information in the MongoDB database using Mongoose. The backend code demonstrates robustness by validating and updating the user's name, email, and phone number.

Src:

```
document.getElementById('general-save-changes-button').addEventListener
('click', saveChangesGeneral);

function saveChangesGeneral() {
    const name = document.getElementById('name').value;
    const email = document.getElementById('email').value;
    const phone = document.getElementById('phone').value;

    const urlParams = new URLSearchParams(window.location.search);
    const userId = urlParams.get('userId');

    console.log("from client.js");
    console.log(userId);
    const data = { id: userId, name, email, phone };
    console.log(data);
    console.log("saveChangesGeneral");

    fetch('/update-profile', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(data),
    })
    .then(response => response.json())
    .then(data => {
        console.log('Success:', data);
        window.alert('You will be logged out to save the new
changes.');
        window.location.href = '/logout';
```

```
    })
    .catch((error) => {
        console.error('Error:', error);
    });

}
```

```
app.get('/current-user', (req, res) => {
  if (req.session.user) {
    res.json(req.session.user);
  } else {
    res.status(401).json({ message: 'Not logged in' });
  }
});

app.use(express.json());

app.post('/update-profile', (req, res) => {
  const { id, name, email, phone } = req.body;

  console.log(`Updating profile for user with id: ${id}`);

  UserModel.findOneAndUpdate({ _id: id }, { name, email, phone }, {
new: true })
    .then(user => {
        if (!user) {
            console.log(`No user found with id: ${id}`);
            res.status(404).json({ message: 'User not found' });
        } else {
            console.log(`Updated profile for user with id: ${id}`);
            res.json({ message: 'Profile updated successfully' });
        }
    })
    .catch(error => console.error(`Error updating user: ${error}`));
});
```

5.2.5 Change Password

Main Files:	Description
• edit-profile.html • server.js • client.js	The change password integration in web app, facilitated by the backend logic in server.js, empowers users to enhance their

	account security by updating their existing passwords. This functionality is implemented through a secure and straightforward process. When a user initiates a password change through the frontend, a POST request is sent to "/change-password," containing the user's ID, current password, and the new desired password. The server.js code effectively handles this request by first checking if the user exists in the database using the user's ID. If the user is not found, a 404 status is returned, and the process is halted. Similarly, if the provided current password does not match the user's stored password, a 401 status is returned, and the operation is terminated. If the user is found and the current password is validated, the user's password is updated to the new password in the MongoDB database using Mongoose. The response to the client confirms the successful password change.

Src:

```
document.getElementById('change-password-save-changes-button').addEventListener('click', () => {
    const currentPassword = document.getElementById('current-password').value;
    const newPassword = document.getElementById('new-password').value;
    const repeatNewPassword = document.getElementById('repeat-new-password').value;

    if (newPassword !== repeatNewPassword) {
      alert('New passwords do not match.');
      return;
    }

    // Get the userID from the URL
    const urlParams = new URLSearchParams(window.location.search);
    const id = urlParams.get('userId');

    fetch('/change-password', {
      method: 'POST',
      body: JSON.stringify({ id, currentPassword, newPassword }),
      headers: { 'Content-Type': 'application/json' },
    })
    .then(response => response.json())
    .then(data => {
```

```
      console.log('Success:', data);
      window.alert('Password changed successfully. You will be logged
out.');
      window.location.href = '/logout';
    })
    .catch(error => console.error('Error:', error));
  });
```

```
app.post('/change-password', (req, res) => {
  const { id, currentPassword, newPassword } = req.body;

  UserModel.findById(id)
    .then(user => {
      if (!user) {
        res.status(404).json({ message: 'User not found' });
        throw new Error('User not found'); // Stop execution if user is
not found
      } else if (user.password !== currentPassword) {
        res.status(401).json({ message: 'Current password is incorrect'
});
        throw new Error('Current password is incorrect'); // Stop
execution if password is incorrect
      } else {
        return UserModel.findByIdAndUpdate(id, { password: newPassword
}, { new: true });
      }
    })
    .then(() => res.json({ message: 'Password changed successfully' }))
    .catch(error => console.error(`Error changing password:
${error}`));
});
```

5.2.6 Add Property

Main Files:	Description
• owner-agent-add-property.html • server.js	The front-end interaction for adding a property in HomeWow is facilitated through the owner-agent-add-property.html file. When an Owner/Agent fills out the property details and clicks the "Add Property" button, the associated JavaScript code extracts the

	entered information and constructs a property object. This object is then sent to the server through a fetch POST request to the '/add-property' route. The server-side implementation, present in the server.js file, processes this request by creating a new PropertyModel instance and saving it to the Property Database. If the property is added successfully, the server responds with a success message. In case of any errors during this process, an appropriate error response is sent back to the client.

Src:

```
// Add a new route to handle the POST request
app.post('/add-property', (req, res) => {
  const property = new PropertyModel(req.body);
  property.save()
    .then(() => {
      res.json({ message: 'Property added successfully' });
    })
    .catch(err => {
      res.status(500).json({ error: err });
    });
});
```

5.2.7 Remove Property

Main Files:	**Description**
• admin-property-management.html • owner-agent-my-property.html • server.js	In the web app system, the removal of a property is seamlessly integrated into both the admin and Owner/Agent interfaces. The admin-property-management.html file handles the presentation and interaction for administrators, while owner-agent-my-property.html caters to Owner/Agents.A fetch DELETE request is sent to the server.js file, specifically targeting the '/properties/:id' route. The server-side implementation leverages the PropertyModel to delete the specified property from the Property Database. Upon successful deletion, the client UI updates in real-time, reflecting the removal of the property. This integration ensures a user-friendly and efficient process for

	administrators and Owner/Agents to manage their property portfolios within the web app system.

Src:

```
app.get('/properties', async (req, res) => {
  try {
      const userId = req.query.userId;
      let properties;
      if (userId) {
          properties = await PropertyModel.find({ userId: userId });
      } else {
          properties = await PropertyModel.find();
      }
      res.json(properties);
  } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Server error' });
  }
});

app.delete('/properties/:id', async function(req, res) {
  const id = req.params.id;

  try {
      await PropertyModel.deleteOne({ _id: id });
      res.status(200).send({ message: 'Property was deleted
successfully' });
  } catch (err) {
      res.status(500).send({ error: 'There was an error deleting the
property' });
  }
});
```

5.2.8 View Property(Owner/Agent)

Main Files:	Description
• owner-agent-my-property.html • server.js	The web app system seamlessly integrates property viewing functionality for Owner/Agents through the

	owner-agent-my-property.html page. The user interface is dynamically updated based on the current user's information, fetched from the server using a '/current-user' endpoint. Upon loading the page, the system retrieves the properties associated with the specific Owner/Agent by making a fetch request to the '/properties' endpoint, passing the user ID as a parameter. The server-side implementation utilizes the PropertyModel to fetch property details from the Property Database. The retrieved property data is then dynamically presented in the user interface, providing essential details such as property name, location, price, and specifications. Each property listing is accompanied by "Edit" and "Remove" buttons, facilitating further actions.

Src:

```
app.get('/properties', async (req, res) => {
  try {
      const userId = req.query.userId;
      let properties;
      if (userId) {
          properties = await PropertyModel.find({ userId: userId });
      } else {
          properties = await PropertyModel.find();
      }
      res.json(properties);
  } catch (error) {
      console.error(error);
      res.status(500).json({ message: 'Server error' });
  }
});
```

5.2.9 Upload property document

Main Files:	Description
• admin-edit-property.html • owner-agent-edit-property.html • server.js	Web app system presents a unified property information update feature catering to both administrators and Owner/Agents. Administrators, through the admin-edit-property.html page, and

	Owner/Agents, via the owner-agent-edit-property.html page, experience a streamlined process. Upon logging in, the system fetches the current user's details, users can effortlessly modify property details and trigger a "Save Changes" action. Internally, the system communicates with the server through distinct endpoints ('/propertiesAdmin/:id' for admins and '/properties/:id' for Owner/Agents). The server-side logic, implemented in server.js, leverages the PropertyModel to efficiently locate and update property information based on the provided details. Once the update is successfully executed, the system promptly communicates the outcome back to the user, affirming the success and presenting the updated property details.

Src:

```
app.put('/properties/:id', async (req, res) => {
  const { name, location, price, description, bedroom, bathroom,
ownerName } = req.body;

  try {
      const property = await PropertyModel.findByIdAndUpdate(
          req.params.id,
          { name, location, price, description, bedroom, bathroom,
ownerName },
          { new: true, useFindAndModify: false }
      );

      if (!property) {
          return res.status(404).json({ success: false, message:
'Property not found' });
      }

      res.json({ success: true, message: 'Property updated
successfully', property });
  } catch (err) {
      console.error('There was an error updating the property:', err);
      res.status(500).json({ success: false, message: 'There was an
error updating the property' });
  }
});
```

5.2.10 Update property information

Main Files:	Description
• admin-edit-property.html • owner-agent-edit-property.html • server.js	Web app system presents a unified property information update feature catering to both administrators and Owner/Agents. Administrators, through the admin-edit-property.html page, and Owner/Agents, via the owner-agent-edit-property.html page, experience a streamlined process. Upon logging in, the system fetches the current user's details, users can effortlessly modify property details and trigger a "Save Changes" action. Internally, the system communicates with the server through distinct endpoints ('/propertiesAdmin/:id' for admins and '/properties/:id' for Owner/Agents). The server-side logic, implemented in server.js, leverages the PropertyModel to efficiently locate and update property information based on the provided details. Once the update is successfully executed, the system promptly communicates the outcome back to the user, affirming the success and presenting the updated property details.

Src:

```
app.put('/properties/:id', async (req, res) => {
  const { name, location, price, description, bedroom, bathroom,
ownerName } = req.body;

  try {
      const property = await PropertyModel.findByIdAndUpdate(
          req.params.id,
          { name, location, price, description, bedroom, bathroom,
ownerName },
          { new: true, useFindAndModify: false }
      );

      if (!property) {
          return res.status(404).json({ success: false, message:
'Property not found' });
      }
```

```
      res.json({ success: true, message: 'Property updated
successfully', property });
  } catch (err) {
      console.error('There was an error updating the property:', err);
      res.status(500).json({ success: false, message: 'There was an
error updating the property' });
  }
});
```

5.2.11 View Properties(Tenant&Searcher)

Main Files:	Description
• index.html • server.js	Upon accessing the web app, users are greeted with a visually appealing display of properties fetched from the server. Each property is showcased with an image, pricing details, and essential information such as location, bedroom count, and bathroom count. The layout ensures a quick and comprehensive overview, facilitating an efficient property search experience. The server-side logic, implemented in server.js, distinguishes between tenants and property searchers. By utilizing distinct endpoints ('/properties' for property searchers), the system fetches properties tailored to the user's specific needs. The userId parameter enables personalization, ensuring that tenants view only the properties relevant to their profile. successful user registration.

Src:

```
app.get('/properties', async (req, res) => {
  try {
      const userId = req.query.userId;
      let properties;
      if (userId) {
          properties = await PropertyModel.find({ userId: userId });
      } else {
          properties = await PropertyModel.find();
      }
      res.json(properties);
  } catch (error) {
      console.error(error);
```

```
      res.status(500).json({ message: 'Server error' });
  }
});
```

5.2.12 View all users

Main Files:	Description
• admin-user-managements • server.js	Web app admin-user-management.html page provides administrators with a comprehensive overview of all users, excluding admin accounts. The dynamic rendering of user details includes names, contact numbers, email addresses, and corresponding user types. User types are visually represented with color-coded labels for easy identification. Upon accessing the admin user management section, the system fetches user data from the server using the '/users' endpoint. The server-side logic in server.js retrieves user information from the UserModel, excluding admin accounts. The retrieved user data is then sent to the client for rendering.

Src:

```
app.get('/users', (req, res) => {
  UserModel.find()
  .then(results => {
    res.json(results);
  })
  .catch(error => console.error(error));
});
```

5.2.13 Remove user

Main Files:	Description
• admin-user-management.html • server.js	The admin-user-management.html page provides administrators with a comprehensive overview of all users, excluding admin accounts. Upon accessing the admin user management section, the system dynamically populates a table with user details. The information includes user

	names, contact numbers, email addresses, and user types. The user types are visually represented with corresponding color-coded labels for easy identification. The server-side logic, implemented in server.js, handles user deletion requests initiated by administrators. The 'DELETE' endpoint ('/users/:id') allows administrators to remove non-admin users seamlessly. A confirmation prompt ensures careful consideration before any user is deleted. Upon deletion, the system updates in real-time, and a success message is logged to the console.

Src:

```
app.delete('/users/:id', async (req, res) => {
  const { id } = req.params;

  try {
      const result = await UserModel.deleteOne({ _id: id });

      if (result.deletedCount === 0) {
          return res.status(404).json({ message: 'User not found' });
      }

      res.json({ message: 'User deleted successfully' });
  } catch (err) {
      console.error(err);
      res.status(500).json({ message: 'Server error' });
  }
});
```

5.2.14 Agreement request

Main Files:	Description
• owner-agent-tenancy-agreement-request.html • server.js	The owner-agent-tenancy-agreement-request.html page facilitates the management of tenancy agreement requests from tenants. It dynamically populates the page with agreement details, including the tenant's name, property name, request date, and status. The system allows property owners or agents to accept or reject agreement requests. For agreements that are not yet accepted, the system provides accept and reject buttons. Clicking the accept button sends a PUT request to the server, updating the agreement

	status to 'Accepted'. Conversely, clicking the reject button sends a DELETE request to remove the agreement from the system. The server-side logic for handling these requests (app.put and app.delete endpoints) ensures that the agreement status is updated or the agreement is deleted accordingly. Successful operations result in a 200 status response, while any errors trigger a 500 status response.

Src:

```
// Get all agreements
app.get('/agreements', async (req, res) => {
  try {
      const ownerId = req.query.ownerId;
      const agreements = await Agreement.find({ ownerId: ownerId });
      res.json(agreements);
  } catch (err) {
      res.status(500).json({ message: err.message });
  }
});

//Accept agreement
app.put('/agreements/:id', (req, res) => {
  const { id } = req.params;
  const { status } = req.body;

  // Update the status of the agreement in the database
  Agreement.findByIdAndUpdate(id, { status: status }, { new: true })
      .then(() => {
          // If successful, return a 200 status
          res.sendStatus(200);
      })
      .catch(error => {
          // If an error occurred, return a 500 status
          console.error(error);
          res.status(500).send('An error occurred');
      });
});

// Delete an agreement
app.delete('/agreements/:id', (req, res) => {
  const agreementId = req.params.id;

  Agreement.deleteOne({ _id: agreementId })
```

```
        .then(() => {
            res.status(200).send({ message: 'Agreement deleted
successfully.' });
        })
        .catch((err) => {
            res.status(500).send({ message: 'An error occurred while
deleting the agreement.' });
        });
});
```

5.2.15 Sent Agreement request

Main Files:	Description
• tenant-tenancy.html • Server.js • index.html	The index.html page displays a list of available properties. Tenants can view property details and, upon deciding to proceed, send an agreement request. The server handles the agreement request by first verifying the existence of the specified property using its ID (req.params.id). If the property is found, a new Agreement instance is created with details such as the tenant's name, request date, property name, and IDs for the property, owner, and tenant. The agreement status is set to 'Pending'. The created agreement is then saved to the database. If successful, the server sends a response indicating that the agreement request has been sent. The tenant-tenancy.html page is designed for tenants to check the status of their agreement requests. Upon loading the page, the script fetches the current user's information and updates navigation links accordingly. Additionally, the script queries the server for agreement data associated with the tenant's ID using the /agreements endpoint. The retrieved data includes details such as the tenant's name, property name, request date, and the status of each agreement. The page dynamically generates a table to display this information, providing tenants with an overview of their tenancy

	agreements and their current statuses.

Src:

```
app.post('/properties/:id/agreement-request', (req, res) => {
  const { username, date } = req.body;

  PropertyModel.findById(req.params.id)
      .then(property => {
          if (!property) {
              console.log(`Property with ID ${req.params.id} not
found`);
              return res.status(404).send('Property not found');
          }

          const agreement = new Agreement({
              tenantName: username,
              date: new Date(date),
              propertyName: property.name,
              propertyId: property._id,
              ownerId: property.userId,
              status: 'Pending'
          });

          return agreement.save();
      })
      .then(() => res.send('Agreement request sent!'))
      .catch(err => {
          console.error('There was an error processing the agreement
request', err);
          res.status(500).send('There was an error processing the
agreement request');
      });
});
```

5.2.16 Pay bill

Main Files:	Description
• tenant-payment.html	Web app enable tenants to seamlessly pay their bills through our secure payment gateway. In the tenant-payment.html, tenants can click on the designated payment button to initiate the payment process. This action seamlessly redirects

	them to a secure payment gateway, where they can securely enter their payment details. Our integration leverages the Stripe payment platform to ensure a smooth and secure transaction experience for tenants.

Src:

```
<stripe-buy-button buy-button-id="buy_btn_1Og0qpCCsVzvQDGlc44NvpLd"

publishable-key="pk_test_51Og0nUCCsVzvQDGleHCU4ZmfFt3l77KuOVOaYu5cJL57c
2JmIgmlGHAoghr6Y2uNTjRtI9jGNse9nOK5gQ5GD3pu00JL8ChbNn">
                            </stripe-buy-button>
                        </div>
        </main>
    </div>
    <script src="tenant.js"></script>
    <script async src="https://js.stripe.com/v3/buy-button.js">
</script>
```

5.2.17 View invoices

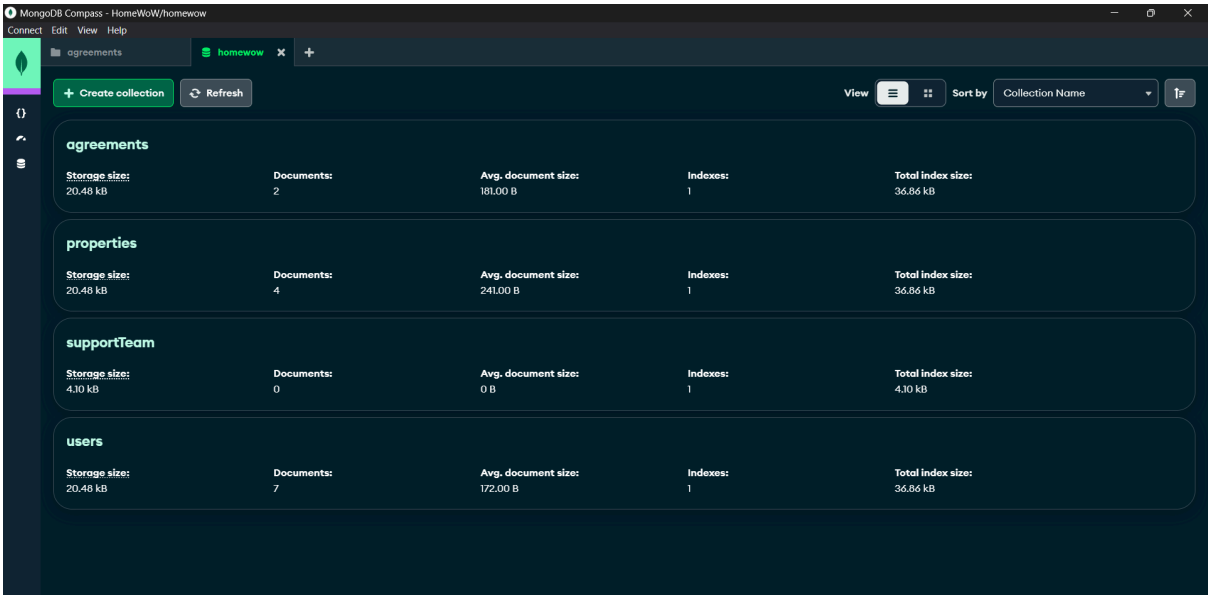
Main Files:	Description
• tenant-invoice.html	.Tenants can easily track and manage their invoices through the tenant-invoice.html page. The page displays a list of invoices with details such as the recipient's name, due date, and payment status. Each invoice is accompanied by a "View & Pay Now" button, providing tenants with a convenient way to access and settle their outstanding bills. Clicking on the button redirects tenants to a secure payment portal, where they can review the invoice details and proceed with the payment. This streamlined integration enhances transparency and efficiency in the billing process, offering tenants a user-friendly interface for invoice management and payment.

Src:

```
 <td>
                                    <a
href="https://invoice.stripe.com/i/acct_1Og0nUCCsVzvQDGl/test_YWNjdF8xT
2cwblVDQ3NWenZRREdsLF9QV3RPRkU5bHo2cGczT2ZENmh0Qm5nWmxReW5ONzAxLDk4MDA4
OTIz0200k626kESv?s=db">
                                        <button class="accept"
type="button">View & Pay Now</button>
                                    </a>
```

```
                </td>
            </tr>
```

5.3 Database



6 Testing

6.1 Testing Strategy

Black box testing will be employed to evaluate the functionality of the system without knowledge of its internal structure or implementation details. Additionally lots of responses from the server side code is added in to ensure that the data being sent from the frontend is same as in the backend

6.2 Test Data

6.2.1 Owner/Agent Test Data Set

- Includes Actor & Properties test data & other user data that used for testing certain use cases

Owner/Agent Data:

User 1

Field Name	Data
name:	Patric Nuggets
password:	Patric
email:	pankers1@businessweek.com
icNum:	881213011234
phone:	0153715732
gender:	Male
userType:	Owner/Agent

User 2

Field Name	Data
name:	Brenna Baiden
password:	Brenna
email:	bbaiden8@biglobe.ne.jp
icNum:	442113011234
phone:	0157615734
gender:	Female

userType:	Owner/Agent

Property Data:

Property 1

Field Name	Data
name:	Seri Maya Condominium
location:	Kuala Lumpur
price:	600
description:	Modern condominium with excellent facilities.
bedroom:	3
bathroom:	2
isRented:	false

Property 2

Field Name	Data
name:	Eco Majestic Terrace House
location:	Semenyih, Selangor
price:	550
description:	Spacious house in a gated community.
bedroom:	4
bathroom:	3
isRented:	false

Property 3

Field Name	Data
name:	Mutiara Ville

location:	Cyberjaya, Selangor
price:	650
description:	Waterfront condominium with stunning sea views
bedroom:	2
bathroom:	2
isRented:	false

Property 4

Field Name	Data
name:	Garden Plaza Serviced Apartment
location:	Cyberjaya, Selangor
price:	420
description:	Serviced apartment with lush green surroundings.
bedroom:	2
bathroom:	1
isRented:	false

Other User Data Used:

User 1

Field Name	Data
name:	Christal Edi
password:	Christal
email:	chawse0@odnoklassniki.ru
icNum:	950101145678
phone:	0123456789
gender:	Female
userType:	tenant

User 2

Field Name	Data

name:	Bank Prose
password:	Bank
email:	bprose2@bloomberg.com
icNum:	930303149876
phone:	0111222333
gender:	Male
userType:	tenant

6.2.2 Tenant Test Data Set

- Includes Actor test data

Tenant Data:

User 1

Field Name	Data
name:	Christal Edi
password:	Christal
email:	chawse0@odnoklassniki.ru
icNum:	950101145678
phone:	0123456789
gender:	Female
userType:	tenant

User 2

Field Name	Data
name:	Bank Prose
password:	Bank
email:	bprose2@bloomberg.com
icNum:	930303149876
phone:	0111222333
gender:	Male
userType:	tenant

Tenant Property Data:

Property 1

Field Name	Data
name:	Dream House
location:	no.45,jalan LEP 1/10,Taman Lestari Putra,43300,Seri Kembangan,Selangor
price:	2888
description:	ghost house
bedroom:	8
bathroom:	5
isRented:	false

6.2.3 Admin Test Data Set

- Includes Actor, Properties & User test data

Admin Data:

User 1

Field Name	Data
name:	HomeWoW Admin
password:	admin
email:	homewowadmin@gmail.com
icNum:	475303149953
phone:	0234222444
gender:	Male
userType:	Admin

User 2

Field Name	Data
name:	HomeWoW Admin 2
password:	admin2
email:	homewowadmin2@gmail.com
icNum:	987327443476
phone:	0984765444
gender:	Female
userType:	Admin

Admin Property Data:

Property 1

Field Name	Data
name:	Seri Maya Condominium
location:	Kuala Lumpur
price:	600
description:	Modern condominium with excellent facilities.
bedroom:	3
bathroom:	2
isRented:	false

Property 2

Field Name	Data
name:	Eco Majestic Terrace House
location:	Semenyih, Selangor
price:	550
description:	Spacious house in a gated community.
bedroom:	4
bathroom:	3
isRented:	false

Property 3

Field Name	Data
name:	Mutiara Ville
location:	Cyberjaya, Selangor
price:	650
description:	Waterfront condominium with stunning sea views
bedroom:	2
bathroom:	2
isRented:	false

Property 4

Field Name	Data
name:	Garden Plaza Serviced Apartment
location:	Cyberjaya, Selangor
price:	420
description:	Serviced apartment with lush green surroundings.
bedroom:	2
bathroom:	1
isRented:	false

Admin User Data:

User 1

Field Name	Data
name:	Patric Nuggets
password:	Patric
email:	pankers1@businessweek.com

icNum:	881213011234
phone:	0153715732
gender:	Male
userType:	Owner/Agent

User 2

Field Name	Data
name:	Brenna Baiden
password:	Brenna
email:	bbaiden8@biglobe.ne.jp
icNum:	442113011234
phone:	0157615734
gender:	Female
userType:	Owner/Agent

User 3

Field Name	Data
name:	Christal Edi
password:	Christal
email:	chawse0@odnoklassniki.ru
icNum:	950101145678
phone:	0123456789
gender:	Female
userType:	tenant

User 4

Field Name	Data
name:	Bank Prose
password:	Bank
email:	bprose2@bloomberg.com

icNum:	930303149876
phone:	0111222333
gender:	Male
userType:	tenant

6.3 Acceptance Tests

6.3.1 Acceptance Tests Owner/Agent

Use Case	Expected Results	% Fulfilled	Priority
1. Sign Up	The user creates a new account successfully.	100%	High
2. Login	The user is logged into the web app.	100%	High
3. Edit Profile	User successfully Changes his Personal information.	100%	High
4. Change Password	User successfully changes his old password with a new password & can use the new password to login into the system.	100%	High
5. Add Property	User adds a property to the system & can view property from my property section.	100%	High
6. Remove Property	User successfully removes the desired property from the system.	100%	High
7. Update Property Information	Property information is successfully updated in the system.	100%	High
8. Upload Property Documents	Upload property documents in PDF format for verification.	100%	High
9. View Property	Allows the user to view all his properties in the system with details & status.	100%	High
10. Agreement Requests	Successfully Agree or disagree with a tenancy agreement.	100%	High
11. Logout	User successfully logout from the web app.	100%	High
12. Updates from payment API	User can see all payment logs associated with the user.	10%	Low

13. Contact Support	User successfully interact with the support team for assistance.	50%	Low
14. Dashboard	User successfully accesses the dashboard and views relevant account information.	50%	High
15. Appointment	User successfully accept/reject an appointment request.	0%	Low

6.3.2 Acceptance Tests Tenant

Use Case	Expected Results	% Fulfilled	Priority
1. Sign Up	The user creates a new account successfully.	100%	High
2. Log In	Upon successful authentication, the user gains access to their account and the functionalities offered by the Web app.	100%	High
3. Edit Profile	Changes made to the profile are saved and reflected in the user's updated profile information.	100%	High
4. Change Password	The user's password is updated to the new one.	100%	High
5. Search Properties	The system displays a list of properties matching the specified search criteria.	0%	Low
6. View Properties	The system displays details of property for the tenant to view.	100%	High
7. Make an appointment	An appointment for viewing the property is scheduled between the user and the property owner/agent.	0%	Low
8. Sent agreement request	The system processes and forwards the user's request for access to the tenancy agreement to the property owner/agent.	100%	High
9. View Outstanding Balance	The system displays the details of the outstanding balance related to the user's tenancy.	100%	High
10. Pay bill	Make payment using the API	100%	High
11. View Invoices	The system displays a list of invoices associated with the user's account.	100%	High
12. Dashboard	Users successfully access the dashboard and view relevant account information.	50%	High
13. Contact	User successfully interacts with the support team for	50%	Low

Support	assistance.		
14. View Agreement	The system displays the finalized and approved tenancy agreement for the specified property to the user.	0%	Low
15. View Receipt	The system displays receipts of past transactions, providing details of payments made by the user.	0%	Low
16. Logout	The user's session is terminated, and they are redirected to the home page	100%	High

6.3.3 Acceptance Tests Admin

Use Case	Expected Results	% Fulfilled	Priority
1. Login	The user is logged into the web app.	100%	High
2. Dashboard	User successfully accesses the dashboard and views relevant account information.	50%	High
3. Updates from Payment API	View payment logs from the Payment Gateway system.	0%	Low
4. View All Users	Successfully access the user management and view all users in the system.	100%	High
5. Search User	successfully search and able to see the specific user.	0%	Low
6. Update User Info.	Update the information of registered users.	0%	High
7. Remove User	Remove a specific user that registered in the web app	100%	High
8. Remove Property	Successfully removes the desired property from the system.	100%	High
9. Contact Support	Interact with the support team for assistance.	50%	Low
10. Logout	Logs out from the web app.	100%	High
11. Update Property	Property information is successfully updated in the system.	100%	High

Information			

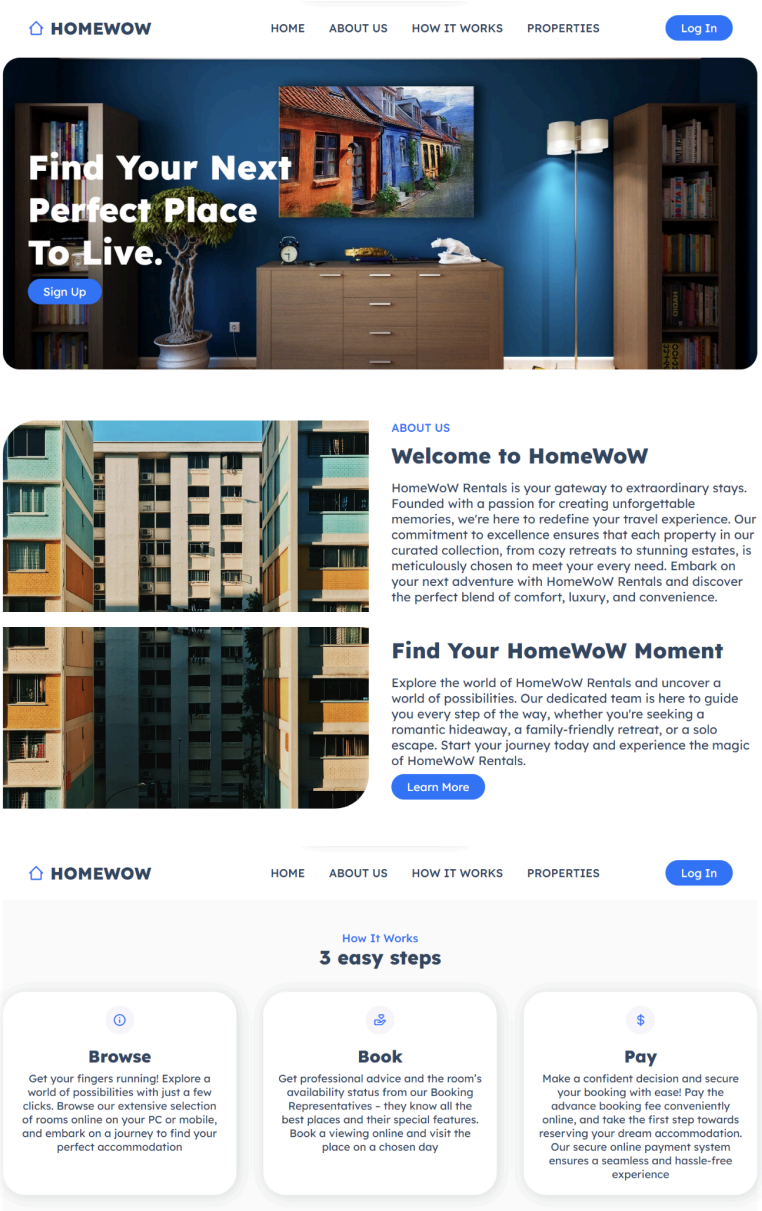
6.3.4 Acceptance Tests Searcher

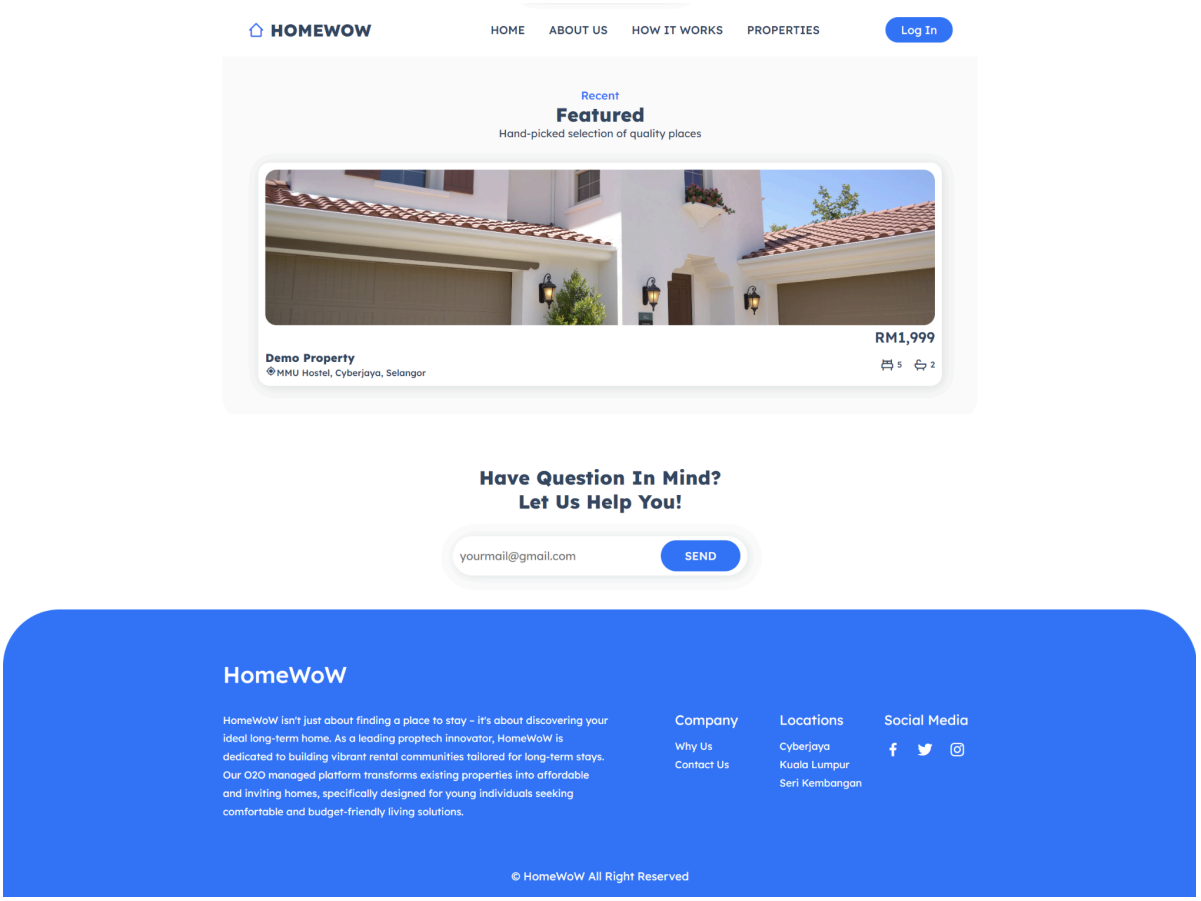
Use Case	Expected Results	% Fulfilled	Priority
1. Search	The user is logged into the web app.	100%	High
1. Dashboard	User successfully accesses the dashboard and views relevant account information.	50%	High

7 Sample Screens

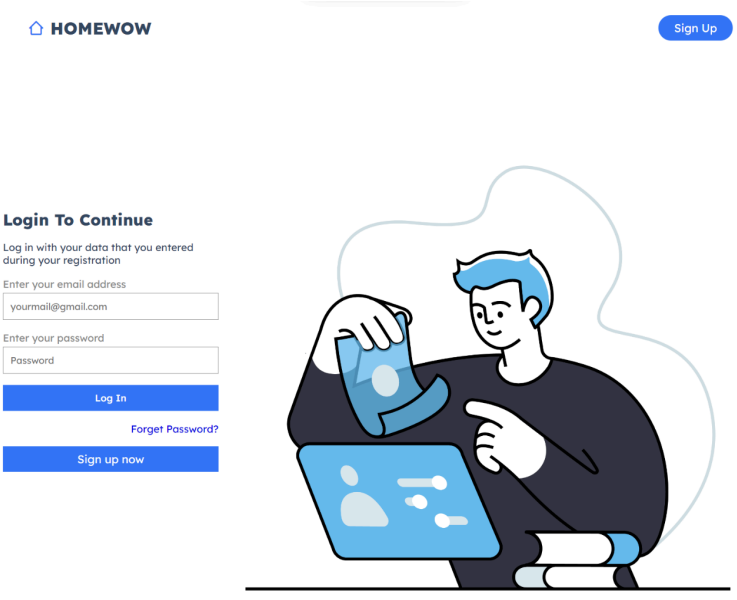
7.1 Main Screens

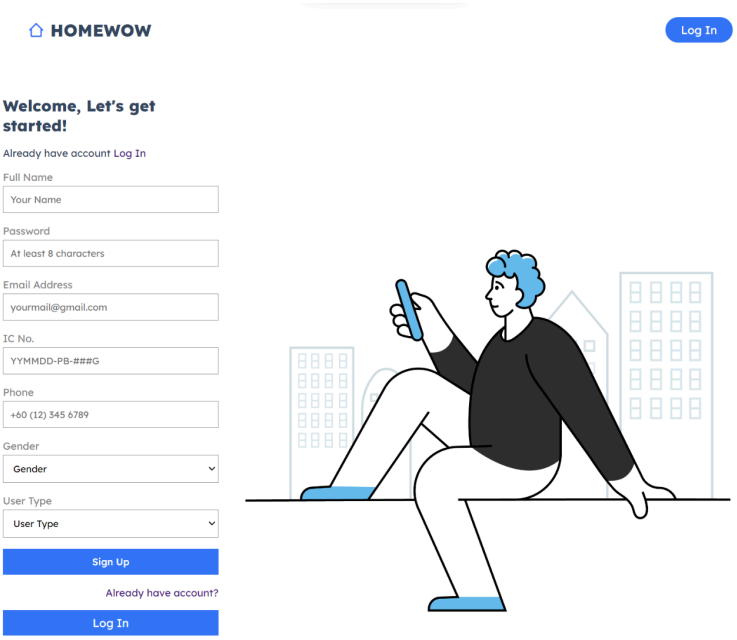
- ***Home page***



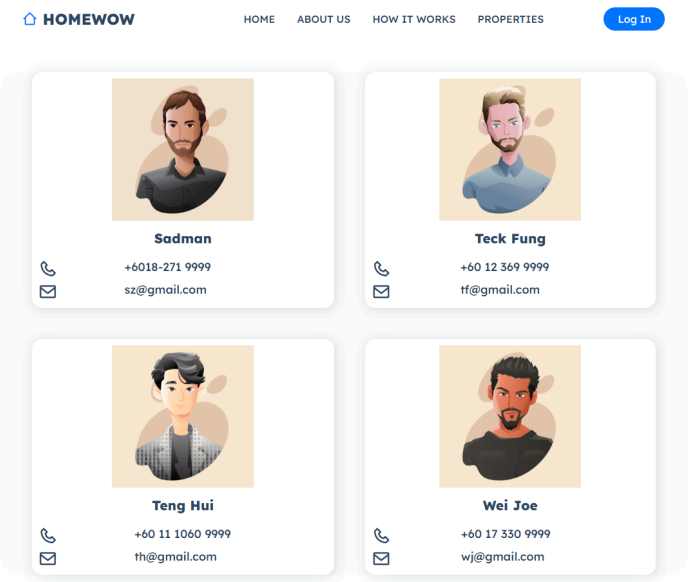


- ***Login / Signup page***

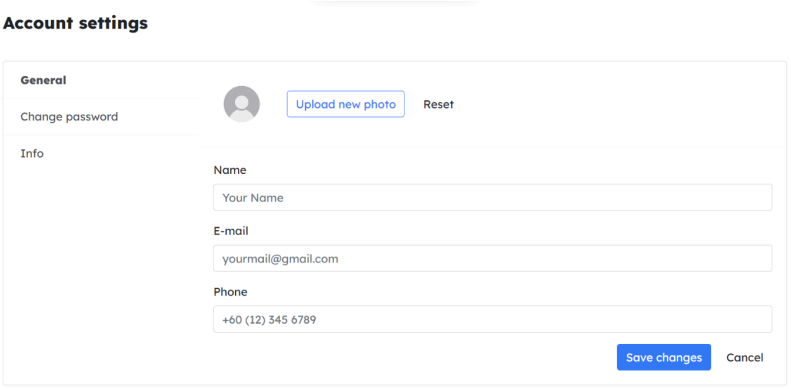


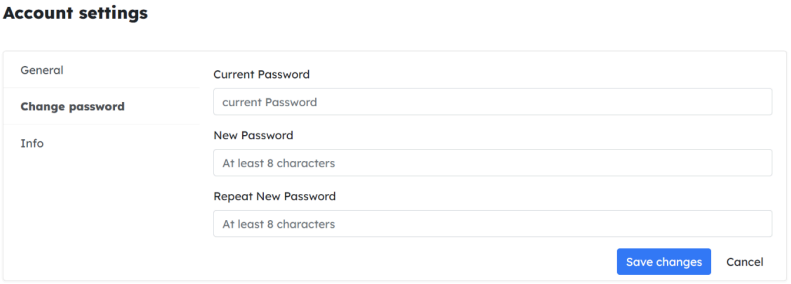


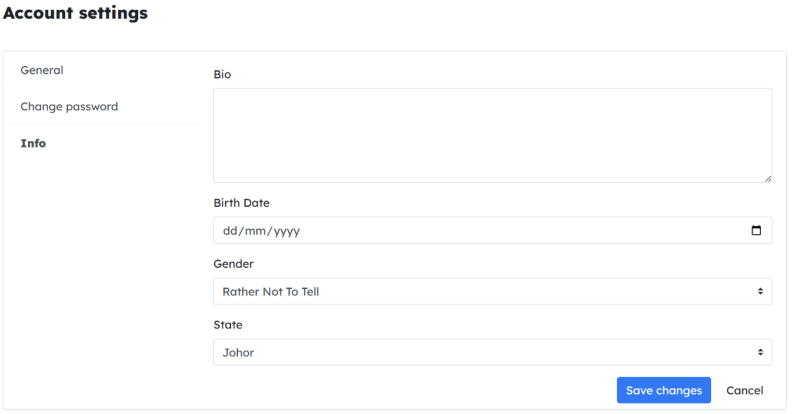
- ***Contact Page***



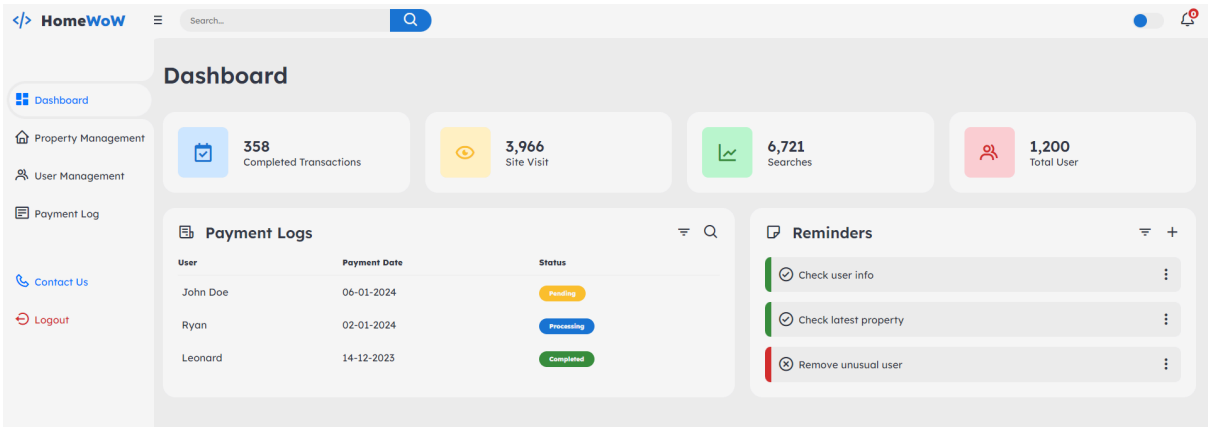
- ***Edit Profile for Owner/Agent & Tenant***

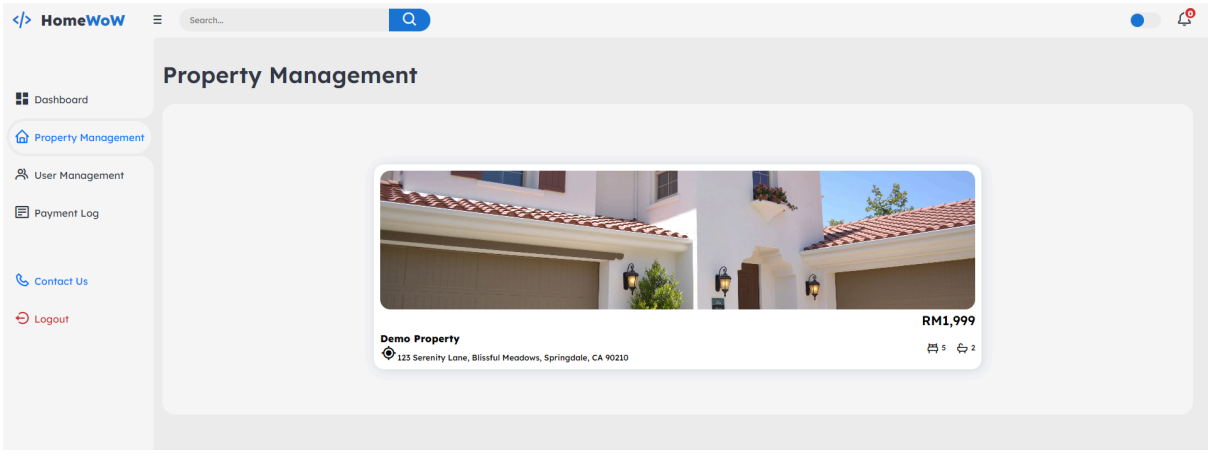

Account settings
General
Change password
Info
Upload new photo
Reset
Name
Your Name
E-mail
yourmail@gmail.com
Phone
+60 (12) 345 6789
Save changes
Cancel

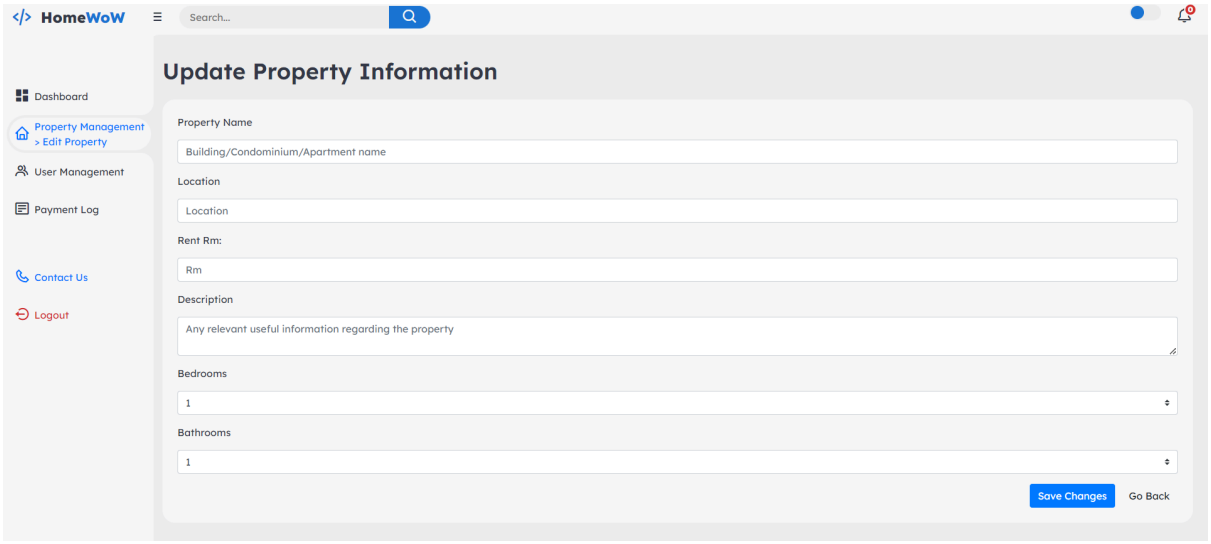

Account settings
General
Change password
Info
Current Password
current Password
New Password
At least 8 characters
Repeat New Password
At least 8 characters
Save changes
Cancel

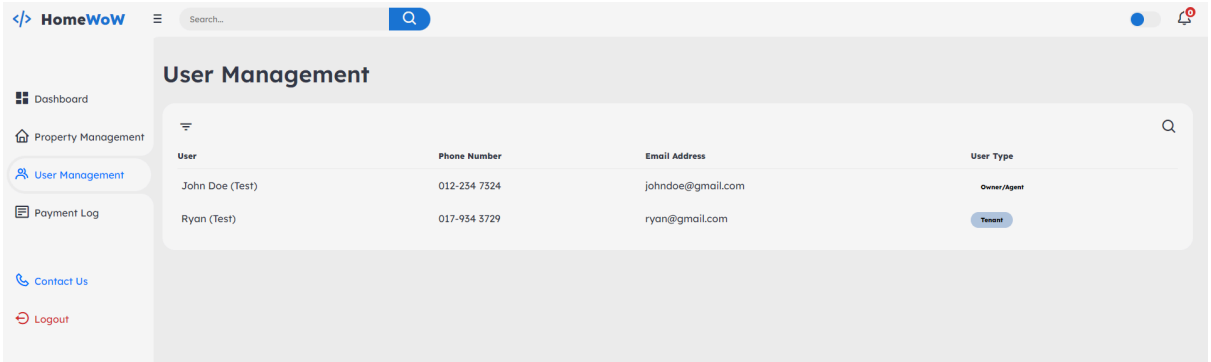

Account settings
General
Change password
Info
Bio
Birth Date
dd/mm/yyyy
Gender
Rather Not To Tell
State
Johor
Save changes
Cancel

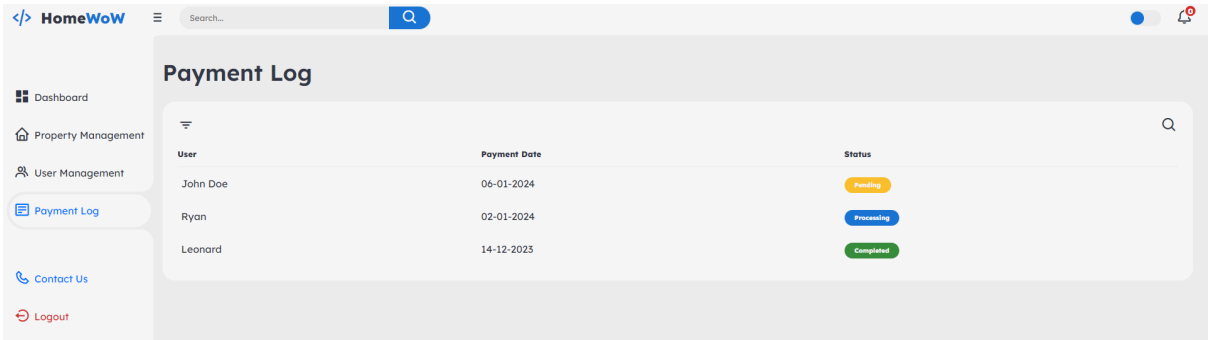
7.2 Admin


HomeWoW
Search...
Dashboard
Property Management
User Management
Payment Log
Contact Us
Logout
Dashboard
358
Completed Transactions
3,966
Site Visit
6,721
Searches
1,200
Total User
Payment Logs
User
Payment Date
Status
John Doe
06-01-2024
Pending
Ryan
02-01-2024
Processing
Leonard
14-12-2023
Completed
Reminders
Check user info
Check latest property
Remove unusual user

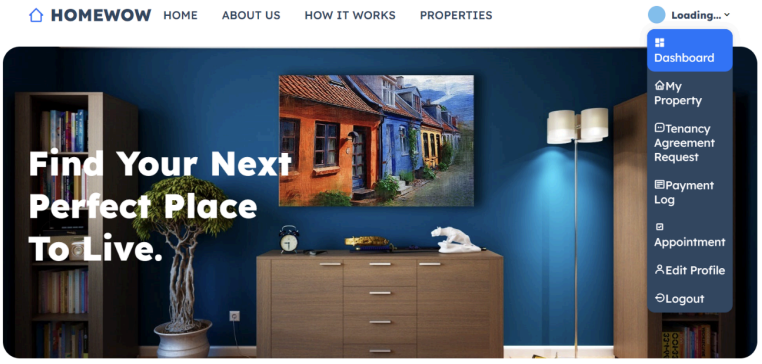

HomeWoW
Search...
Property Management
Dashboard
Property Management
User Management
Payment Log
Contact Us
Logout
RM1,999
Demo Property
123 Serenity Lane, Blissful Meadows, Springdale, CA 90210

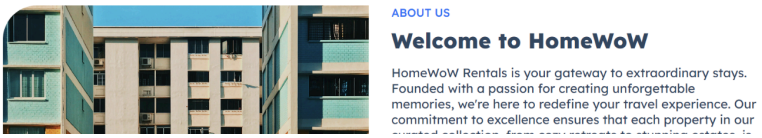

HomeWoW
Search...
Update Property Information
Dashboard
Property Management > Edit Property
User Management
Payment Log
Contact Us
Logout
Property Name
Building/Condominium/Apartment name
Location
Location
Rent Rm:
Rm
Description
Any relevant useful information regarding the property
Bedrooms
1
Bathrooms
1
Save Changes
Go Back

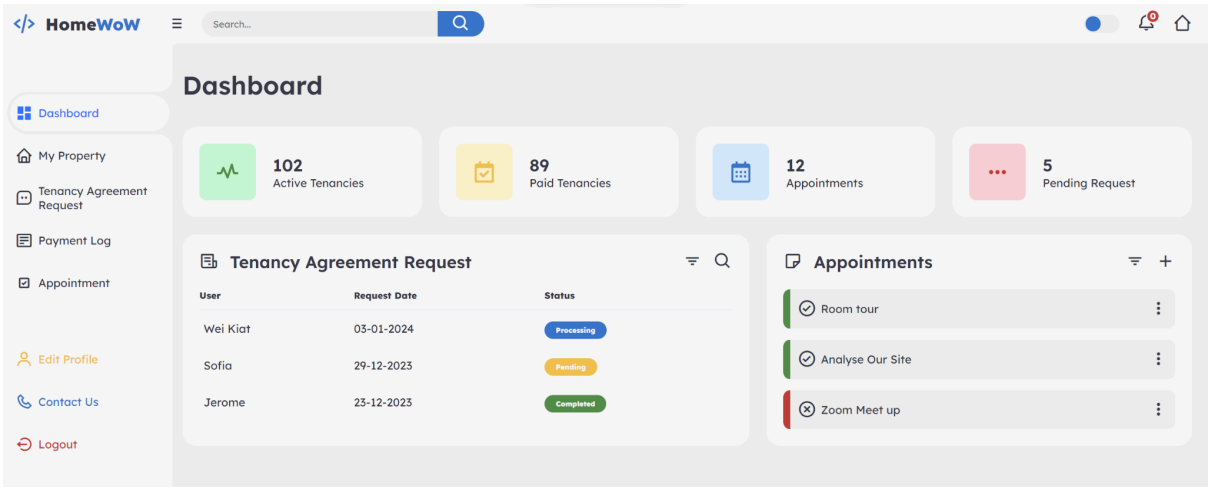

HomeWoW
Search...
User Management
Dashboard
Property Management
User Management
Payment Log
Contact Us
Logout
User
Phone Number
Email Address
User Type
John Doe (Test)
012-234 7324
johndoe@gmail.com
Owner/Agent
Ryan (Test)
017-934 3729
ryan@gmail.com
Tenant

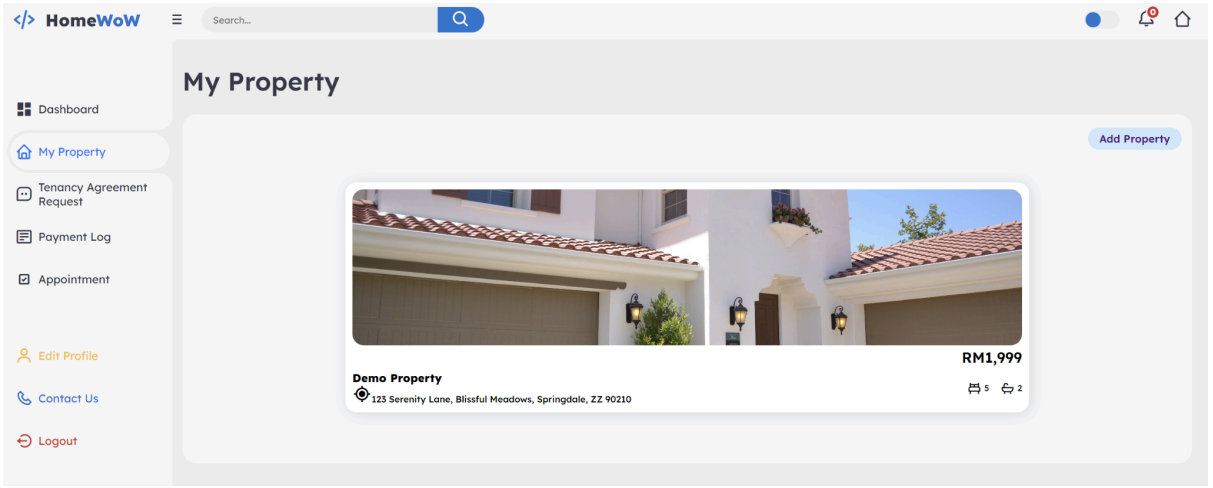

HomeWoW
Search...
Payment Log
Dashboard
Property Management
User Management
Payment Log
Contact Us
Logout
User
Payment Date
Status
John Doe
06-01-2024
Pending
Ryan
02-01-2024
Processing
Leonard
14-12-2023
Completed

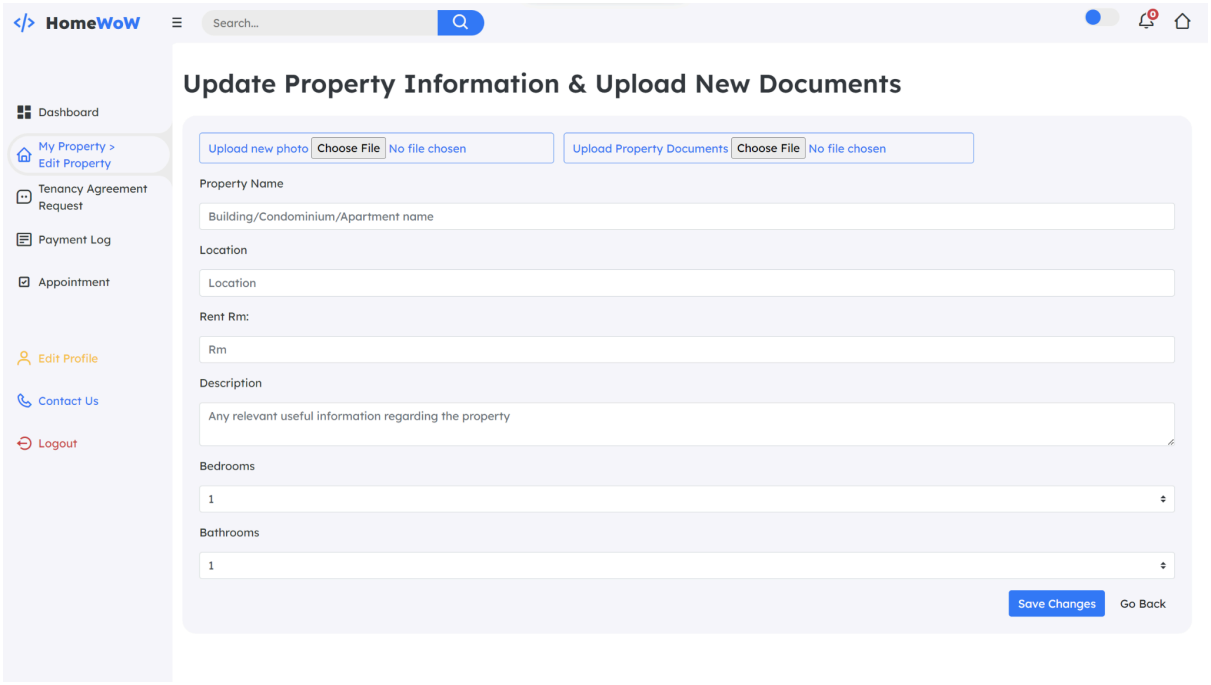
7.3 Owner/Agent

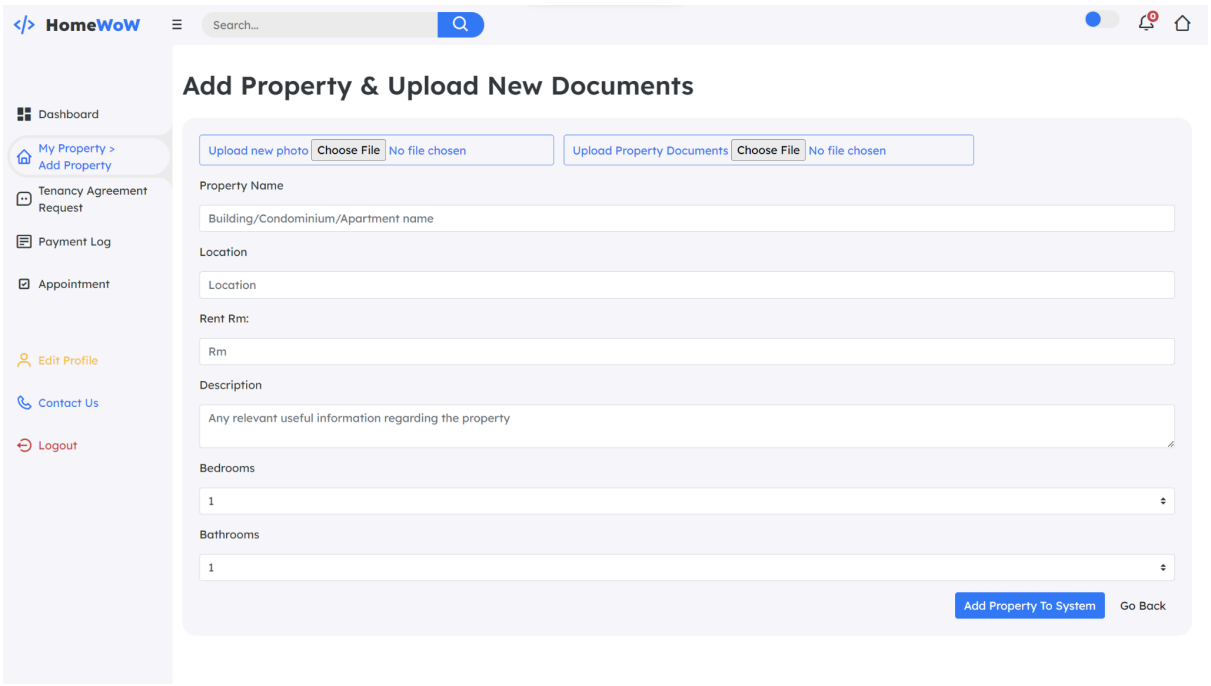


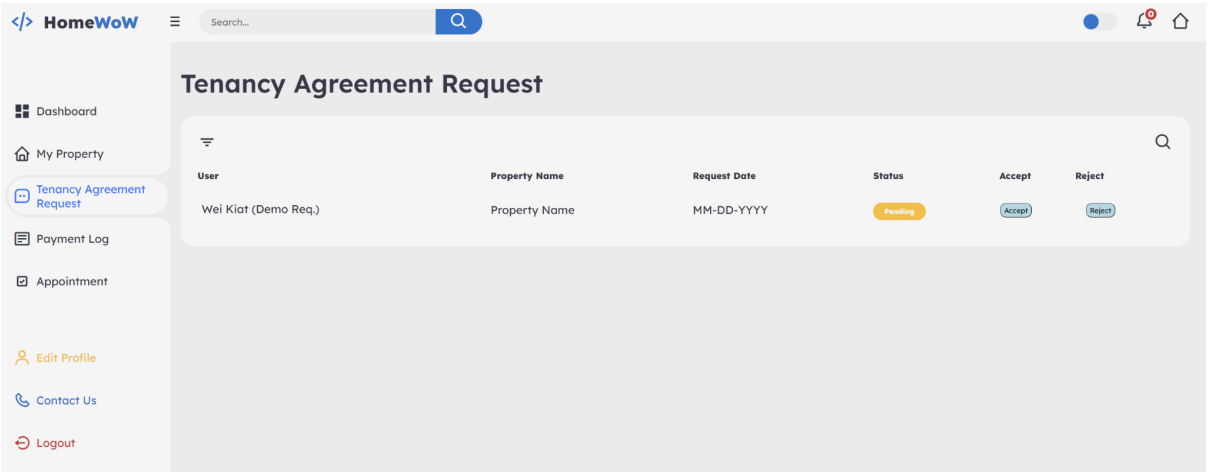


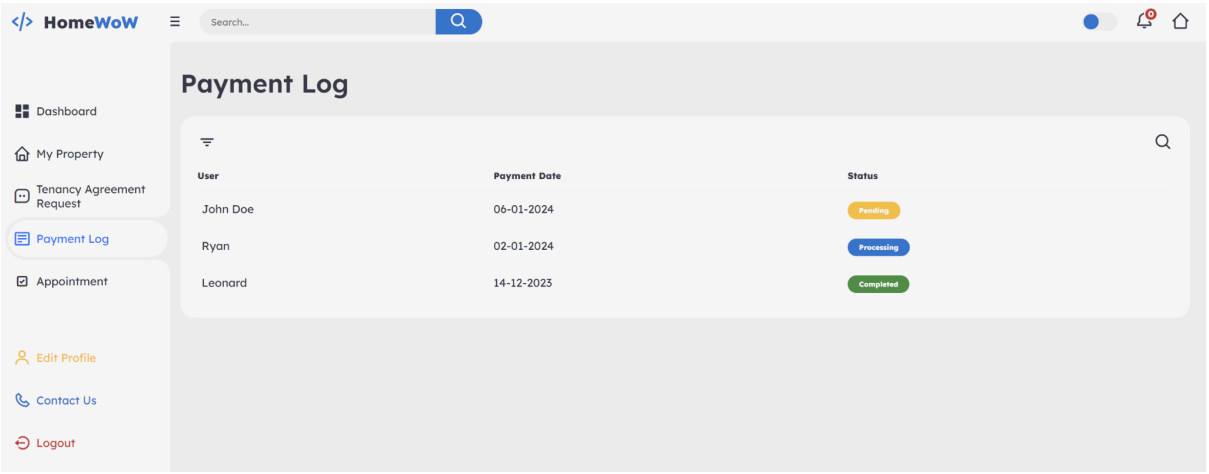





HomeWoW
Search...
Update Property Information & Upload New Documents
Dashboard
My Property > Edit Property
Tenancy Agreement Request
Payment Log
Appointment
Edit Profile
Contact Us
Logout
Upload new photo
Choose File
No file chosen
Upload Property Documents
Choose File
No file chosen
Property Name
Building/Condominium/Apartment name
Location
Location
Rent Rm:
Rm
Description
Any relevant useful information regarding the property
Bedrooms
1
Bathrooms
1
Save Changes
Go Back

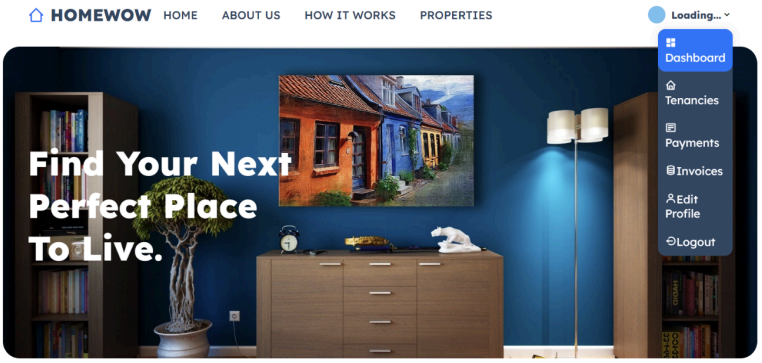

HomeWoW
Search...
Add Property & Upload New Documents
Dashboard
My Property > Add Property
Tenancy Agreement Request
Payment Log
Appointment
Edit Profile
Contact Us
Logout
Upload new photo
Choose File
No file chosen
Upload Property Documents
Choose File
No file chosen
Property Name
Building/Condominium/Apartment name
Location
Location
Rent Rm:
Rm
Description
Any relevant useful information regarding the property
Bedrooms
1
Bathrooms
1
Add Property To System
Go Back

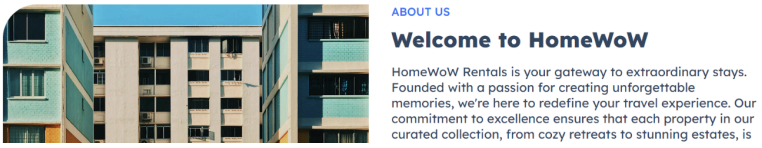

HomeWoW
Search...
Tenancy Agreement Request
Dashboard
My Property
Tenancy Agreement Request
Payment Log
Appointment
Edit Profile
Contact Us
Logout
User
Property Name
Request Date
Status
Accept
Reject
Wei Kiat (Demo Req.)
Property Name
MM-DD-YYYY
Pending
Accept
Reject

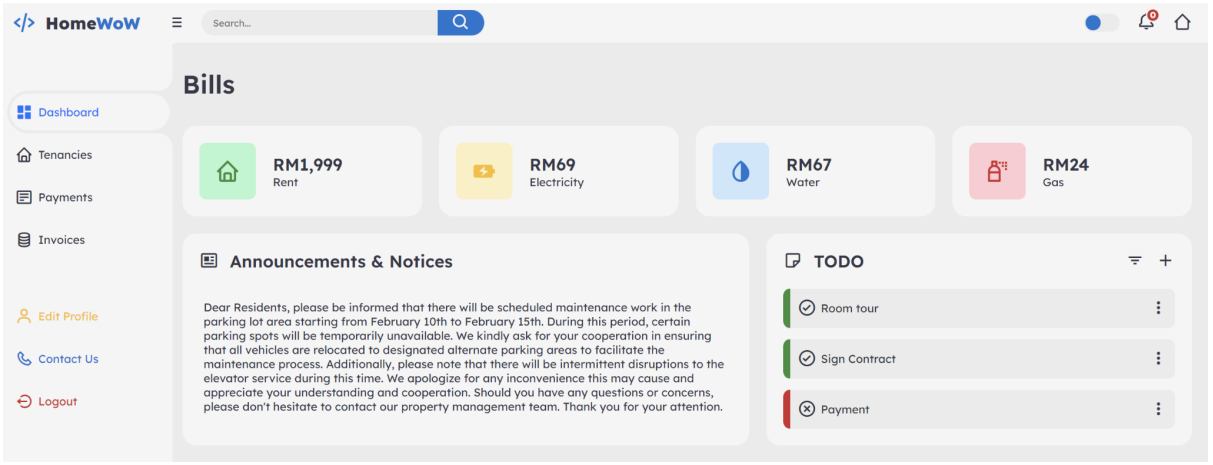

HomeWoW
Search...
Payment Log
Dashboard
My Property
Tenancy Agreement Request
Payment Log
Appointment
Edit Profile
Contact Us
Logout
User
Payment Date
Status
John Doe
06-01-2024
Pending
Ryan
02-01-2024
Processing
Leonard
14-12-2023
Completed

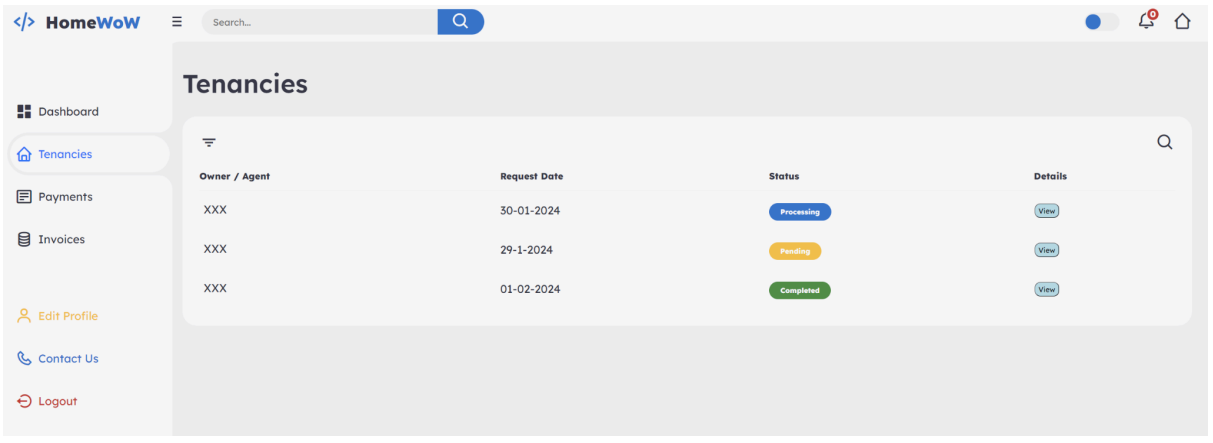

HomeWoW
Search...
Appointment
Dashboard
My Property
Tenancy Agreement Request
Payment Log
Appointment
Edit Profile
Contact Us
Logout
Room tour
Analyse Our Site
Zoom Meet up

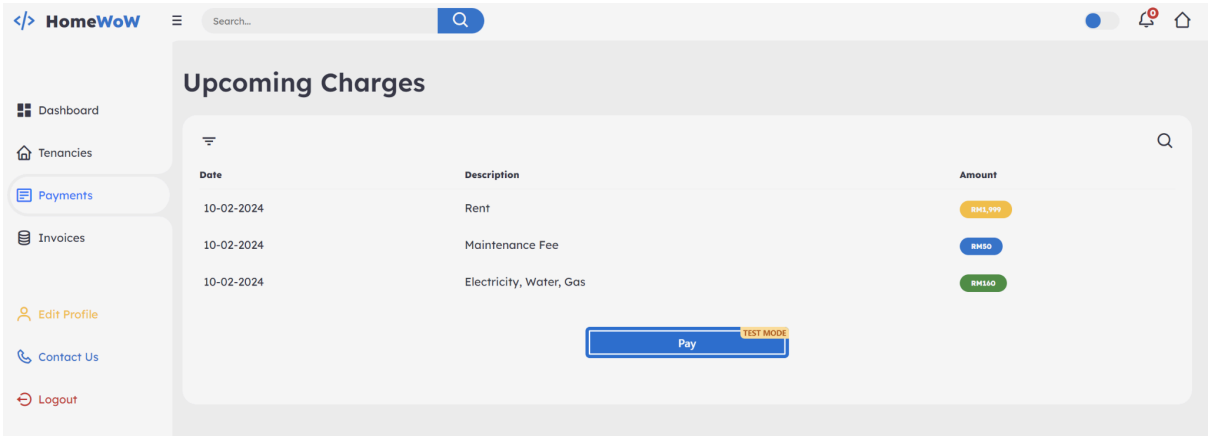
7.4 Tenant

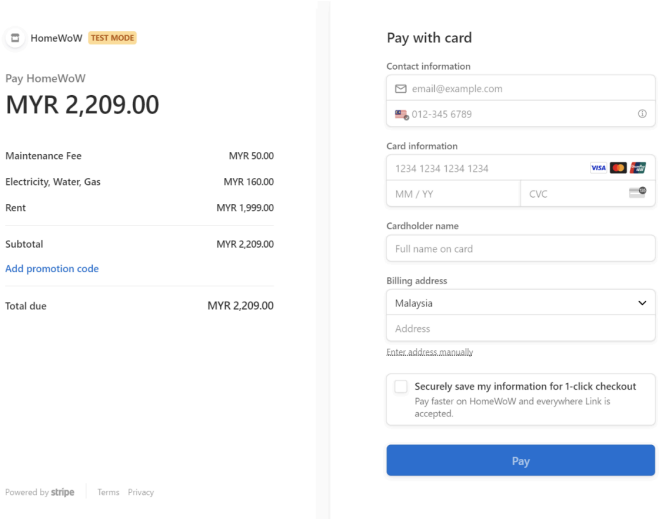

HOMEWOW
HOME
ABOUT US
HOW IT WORKS
PROPERTIES
Loading...
Dashboard
Tenancies
Payments
Invoices
Edit Profile
Logout
Find Your Next Perfect Place To Live.

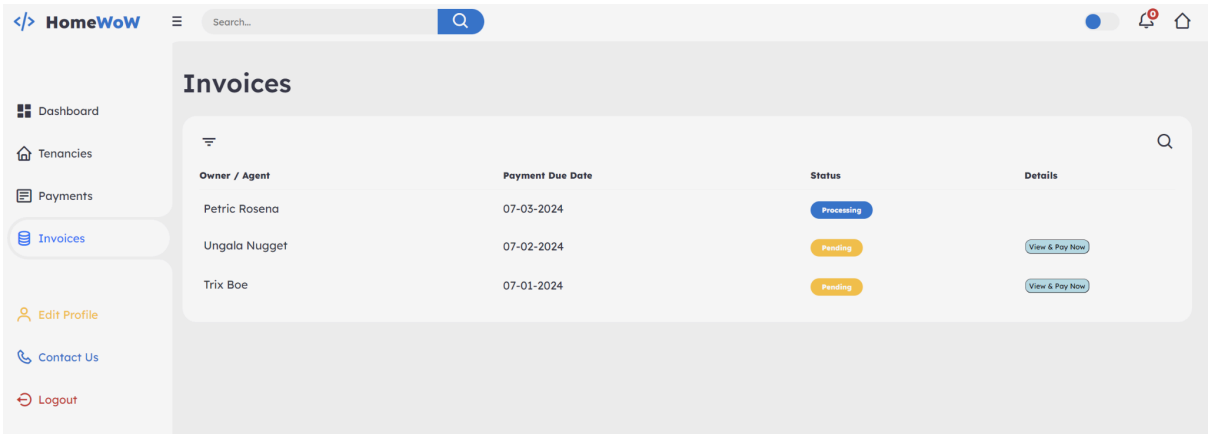

ABOUT US
Welcome to HomeWoW
HomeWoW Rentals is your gateway to extraordinary stays. Founded with a passion for creating unforgettable memories, we're here to redefine your travel experience. Our commitment to excellence ensures that each property in our curated collection, from cozy retreats to stunning estates, is


HomeWoW
Search...
Dashboard
Tenancies
Payments
Invoices
Edit Profile
Contact Us
Logout
Bills
RM1,999
Rent
RM69
Electricity
RM67
Water
RM24
Gas
Announcements & Notices
Dear Residents, please be informed that there will be scheduled maintenance work in the parking lot area starting from February 10th to February 15th. During this period, certain parking spots will be temporarily unavailable. We kindly ask for your cooperation in ensuring that all vehicles are relocated to designated alternate parking areas to facilitate the maintenance process. Additionally, please note that there will be intermittent disruptions to the elevator service during this time. We apologize for any inconvenience this may cause and appreciate your understanding and cooperation. Should you have any questions or concerns, please don't hesitate to contact our property management team. Thank you for your attention.
TODO
Room tour
Sign Contract
Payment


HomeWoW
Search...
Dashboard
Tenancies
Payments
Invoices
Edit Profile
Contact Us
Logout
Tenancies
Owner / Agent
Request Date
Status
Details
XXX
30-01-2024
Processing
View
XXX
29-1-2024
Pending
View
XXX
01-02-2024
Completed
View


HomeWoW
Search...
Upcoming Charges
Dashboard
Tenancies
Payments
Invoices
Edit Profile
Contact Us
Logout
Date
Description
Amount
10-02-2024
Rent
RM1,999
10-02-2024
Maintenance Fee
RM50
10-02-2024
Electricity, Water, Gas
RM160
TEST MODE
Pay


HomeWoW
TEST MODE
Pay HomeWoW
MYR 2,209.00
Maintenance Fee
MYR 50.00
Electricity, Water, Gas
MYR 160.00
Rent
MYR 1,999.00
Subtotal
MYR 2,209.00
Add promotion code
Total due
MYR 2,209.00
Powered by stripe
Terms
Privacy
Pay with card
Contact information
email@example.com
012-345 6789
Card information
1234 1234 1234 1234
MM / YY
CVC
Cardholder name
Full name on card
Billing address
Malaysia
Address
Enter address manually
Securely save my information for 1-click checkout
Pay faster on HomeWoW and everywhere Link is accepted.
Pay


HomeWoW
Search...
Invoices
Dashboard
Tenancies
Payments
Invoices
Edit Profile
Contact Us
Logout
Owner / Agent
Payment Due Date
Status
Details
Petric Rosena
07-03-2024
Processing
Ungala Nugget
07-02-2024
Pending
View & Pay Now
Trix Boe
07-01-2024
Pending
View & Pay Now

8 Conclusion

8.1 Completion of Software

Around 50% of software has been completed unfortunately not all the requirements are fulfilled due to their complexity & dependencies on each other, however some requirements are fulfilled 100%

8.2 Software Quality Assurance

Requirements Analysis (Sadman)

- Thoroughly analyze and understand the project requirements to ensure they are clear, complete, and achievable.
- Conduct regular reviews with team members to address any issues and change requirements as needed.

Design Review: (Teck Fung)

- Evaluate the software architecture and design against established requirements, ensuring adherence to best practices.
- Emphasizing on scalability, maintainability, and performance, so that it is easier to ensure the webpage for long-term usage.

Coding Standards and Guidelines (Teck Fung)

- Established to promote consistency and readability for the code.

Regular Code Review (Teng Hui)

- To identify and rectify the deviations from the standards to help our group for better collaboration on the webapp.

Performance Testing (Joe)

- Conduct testing to evaluate the new feature implemented to the code
- Optimize the code base on the performance test result

Task Prioritization (Sadman)

- To set the priority of each task to be worked on.

Regular Learning (Joe)

- To ensure more knowledge gain to implement the better work along the way working on the webapp.

Documentation (Teng Hui)

- Maintain up-to-date documentation
- Make sure is accessible and understandable to everyone in the group

8.3 Group Collaboration

Name	Actor	Contribution
SADMAN ZULFIQUER	**Owner/Agent**	**Everyone contributed equally in the project**
TAN TENG HUI	**Tenant**	
TENG WEI JOE	**Admin**	
HO TECK FUNG	**Searcher & Admin**	

8.4 Problems Encountered

We encountered a significant challenge when we realized that our existing frontend design for the website wasn't meeting our expectations. In response, we made a strategic decision to embark on a complete redesign and recoding process to enhance the user interface and overall user experience. One of the hurdles we faced was ineffective time management, leading to a need for accelerated progress. This situation prompted us to reevaluate our project timelines and implement more efficient time allocation strategies to ensure a smoother development process. Connecting the backend with the frontend, especially the process of storing data into our database, presented another noteworthy challenge. Specifically, understanding how to integrate Node.js with MongoDB seamlessly became a focal point. To address this, we dedicated time to extensive research, consulting various resources, and watching informative YouTube videos. This proactive approach enabled us to gain the insights needed to establish robust connections between our backend and frontend, ensuring the proper storage and retrieval of data. Despite these challenges, our commitment to overcoming obstacles and improving our project's elements has been a valuable learning experience. It has strengthened our problem-solving skills and enhanced our ability to navigate complex technical issues in the development process.

8.5 Remarks/Comments

- Note that for the user Admin the is added manually from the database, in Sign Up the userType Admin is available to make testing easy only however in real world scenario this does not apply.

9 User Guide (optional)

- Please refer end of 5.1 Development Environment on how to start Sever & how to access it