ABSTRACT

Rentify is a comprehensive property management platform built using the MERN (MongoDB, Express.js, React.js, Node.js) stack, designed to streamline the rental process for both landlords and tenants. The platform integrates essential features such as property listings, tenant management, rental payments, maintenance tracking, and real-time communication into one unified, user-friendly interface. Landlords can easily manage property listings, track rent payments, handle maintenance requests, and communicate with tenants directly through integrated messaging. Tenants can browse available properties, submit applications, make payments, and submit maintenance requests, all through a seamless interface. Rentify incorporates JWT-based authentication for secure user logins and role-based access control, ensuring that landlords, tenants, and admins have appropriate access to relevant features. Its scalable architecture supports a wide range of users, from individual landlords to large property management companies. The platform's mobile-responsive design ensures it is accessible on desktops, tablets, and smartphones. Rentify also integrates a secure payment gateway for processing rent payments and provides transparency between landlords and tenants. The backend uses MongoDB to store data securely, while the frontend offers a dynamic and responsive user interface. The application simplifies property management by automating key tasks and offering a streamlined rental experience, helping landlords manage properties more efficiently and allowing tenants to find and manage rentals with ease. Rentify's primary goal is to modernize and improve the rental experience, making it more efficient, secure, and user-friendly for both landlords and tenants.

TABLE OF CONTENTS

CHAP.NO.	TITLE	PAGE NO.
	ABSTRACT	
1	**INTRODUCTION** 1.1 PROJECT TITLE 1.2 PROBLEM STATEMENT 1.3 SPECIFIC OBJECTIVES 1.4 SCOPE 1.5 TEAM MEMBERS	**1**
2	**PROJECT OVERVIEW** 2.1 PURPOSE 2.2 FEATURES	**4**
3	**ARCHITECTURE** 3.1 FRONTEND 3.2 BACKEND 3.3 DATABASE 3.3.1 DATABASE SCHEMA 3.3.2 DATABASE INTERACTIONS 3.3.3 RELATIONSHIPS BETWEEN COLLECTIONS 3.3.4 INDEXES FOR PERFORMANCE OPTIMIZATION	**7**
4	**SETUP INSTRUCTIONS** 4.1 PREREQUISITES 4.2 INSTALLATION 4.3 CONFIGURE ENVIRONMENTAL VARIABLES 4.4 RUNNING THE APPLICATION	**15**
5	**FOLDER STRUCTURE** 5.1 CLIENT 5.2 SERVER	**19**
6	**RUNNING THE APPLICATION** 6.1 BACKEND SERVER 6.2 FRONTEND SERVER 6.3 ACCESSING THE APPLICATION	**22**
7	**API DOCUMENTATION**	**25**

8	**AUTHENTICATION** 8.1 AUTHENTICATION 8.2 AUTHORIZATION 8.3 MIDDLEWARE FUNCTIONS	**31**
9	**USER INTERFACE**	**33**
10	**TESTING** 10.1 TESTING STRATEGY 10.2 TOOLS USED FOR TESTING	**35**
11	**SCREENSHOTS**	**38**
12	**KNOWN ISSUES**	**41**
13	**FUTURE ENHANCEMENTS**	**42**

CHAPTER 1

INTRODUCTION

1.1 PROJECT TITLE

House Rent App Using MERN Stack

The management of rental properties often involves a multitude of complex tasks that can be inefficient, time-consuming, and prone to errors. Traditional property management relies on manual processes for managing tenants, collecting payments, maintaining properties, and handling communications, often leading to confusion, delays, and frustration for both landlords and tenants. Furthermore, existing property management systems tend to be fragmented, addressing only specific aspects like property listings or rent payments, but failing to offer a comprehensive solution that integrates all key functionalities into a single platform. This lack of cohesion results in inefficiencies, communication gaps, and ultimately, a subpar experience for all parties involved.

1.2 PROBLEM STATEMENT

Managing properties, tenants, and maintenance tasks involves numerous manual processes that are inefficient and prone to errors. Landlords often face challenges in tracking rent payments, resolving tenant issues, and maintaining effective communication. Similarly, tenants encounter difficulties in finding suitable properties, submitting maintenance requests, and interacting with landlords efficiently. Existing solutions are fragmented, focusing only on specific aspects like listings or payments without a holistic approach.

1.3 SPECIFIC OBJECTIVES

 Develop a secure, user-friendly platform for landlords to manage properties, tenants, and payments.

 Create a reliable search system for tenants to find suitable properties with recommendations.

 Facilitate real-time communication and efficient handling of maintenance requests.

 Ensure scalability and security to cater to multiple users simultaneously.

1.4 SCOPE

Rentify is designed for individual landlords, property managers, and tenants. It provides functionality to handle property listings, tenant onboarding, rent payments, maintenance requests, and real-time communication. The platform can also be extended to support third-party integration for payment gateways and AI-based property recommendations.

1.5 TEAM MEMBERS

- **TEAM LEAD: SHARMILA R (311521205049)**
  Role: Back-End Developer
  The back-end developer leads the team, ensuring effective planning, task allocation, and timely progress. They develop the server-side architecture using Node.js and Express.js**,** handling API design for user authentication, property management, and payments. Additionally, they integrate third-party tools like Stripe for secure transactions. They ensure data security, implement JWT-based authentication, and resolve critical technical challenges. Coordinating team collaboration and achieving milestones is also their responsibility.
- **TEAM MEMBER 1: PRIYADHARSHINI V(311521205040)**
  Role: Front-End Developer
  The Front-End Developer focuses on creating a responsive and intuitive user interface using React.js**.** They design and implement key components such as the home page, search functionality, and user dashboards. By integrating APIs, they ensure seamless communication between the front-end and back-end systems. Their role involves performance optimization for fast loading and smooth operation. They also test UI elements for compatibility across devices and browsers.

- **TEAM MEMBER 2: ROOPADHARSINI R J(311521205043)**

  Role: Database Administrator & Back-End Support

  The Database Administrator designs and manages the MongoDB database, creating schemas for users, properties, and transactions. They optimize data queries and ensure database scalability for large datasets. In collaboration with the back-end developer, they handle server-side logic and implement data relationships. Their role also includes monitoring database performance and resolving issues to maintain data integrity. They support real-time features like chat and notifications.

- **TEAM MEMBER 3: TANUJHAA G(311521205053)**

  Role: Quality Assurance & Documentation Specialist

  The Quality Assurance Specialist tests all components of the app, ensuring functionality, performance, and security. They conduct unit, integration, and end-to-end testing to identify and resolve bugs. They write comprehensive documentation, including technical references and user guides, to support deployment and maintenance. Additionally, they oversee version control to maintain code quality. Their role also includes preparing reports and presentations for project updates.

CHAPTER 2

PROJECT OVERVIEW

2.1 PURPOSE

The Rentify House Rental App was designed to revolutionize how rental properties are managed and rented by providing a centralized digital platform that benefits both landlords and tenants. Traditional rental management methods often involve inefficiencies, such as manual documentation, fragmented communication, and unclear processes for payments and maintenance tracking. The purpose of this project is to eliminate these bottlenecks by digitizing and automating these tasks within a single, user-friendly application.

The platform enables landlords to manage property listings, track rental payments, handle tenant interactions, and monitor maintenance requests efficiently. For tenants, it provides an easy-to-navigate interface for browsing properties, signing rental agreements, and maintaining direct communication with landlords. The inclusion of features like secure payments via a gateway, real-time chat, and tenant screening tools ensures a seamless and transparent experience for all users.

The primary goal of the app is to improve operational efficiency in property management while maintaining data security and user privacy. By leveraging the powerful MERN stack, the project emphasizes scalability, making it suitable for both individual landlords and large property management companies. Additionally, the app is designed to be responsive, ensuring accessibility across various devices, including desktops, tablets, and smartphones.

The project also aims to foster transparency and trust between landlords and tenants by providing clear documentation and communication channels. It serves as a one-stop solution for managing all aspects of the rental process, eliminating the need for multiple tools or platforms. Ultimately, the Rentify app aspires to modernize the rental experience, saving time, reducing costs, and ensuring a better experience for all stakeholders involved in the rental ecosystem.

2.2 FEATURES

1. Property Listings Management

- Landlords can easily add, update, or remove property listings.
- Tenants can browse detailed property descriptions, including images, amenities, and rental terms.

2. Advanced Search and Filtering

- Tenants can search properties based on location, price, size, and features.
- Dynamic filters ensure quick and precise results.

3. User Authentication and Role Management

- Secure user authentication with JWT ensures data privacy.
- Role-based access for landlords, tenants, and administrators.

4. Real-Time Chat System

- Facilitates direct communication between landlords and tenants.
- Enables efficient handling of inquiries, requests, or complaints.

5. Secure Payment Gateway

- Integrated payment system for tenants to pay rent and deposits online.
- Supports payment tracking and receipts for landlords.

6. Maintenance Tracking

- Tenants can report maintenance issues, which landlords can track and resolve.
- A dedicated dashboard for managing maintenance requests.

7. Detailed Property Information

- Interactive pages for detailed property viewing, including maps and nearby amenities.
- Property comparison for tenants to evaluate options.

8. Tenant Management

- Tools for landlords to manage tenant profiles, agreements, and payment histories.
- Automated reminders for upcoming payments or contract renewals.

9. Add Property Module

- Simple forms for landlords to add property details, images, and rental terms.
- Image storage handled via Firebase Storage for reliability.

10. Mobile-Responsive Design

- Ensures the app works seamlessly on desktops, tablets, and smartphones.

11. Dashboard for Insights

- Landlords and tenants get a personalized dashboard for tracking activities.
- Visual insights on payments, maintenance, and property status.

12. Scalable Architecture

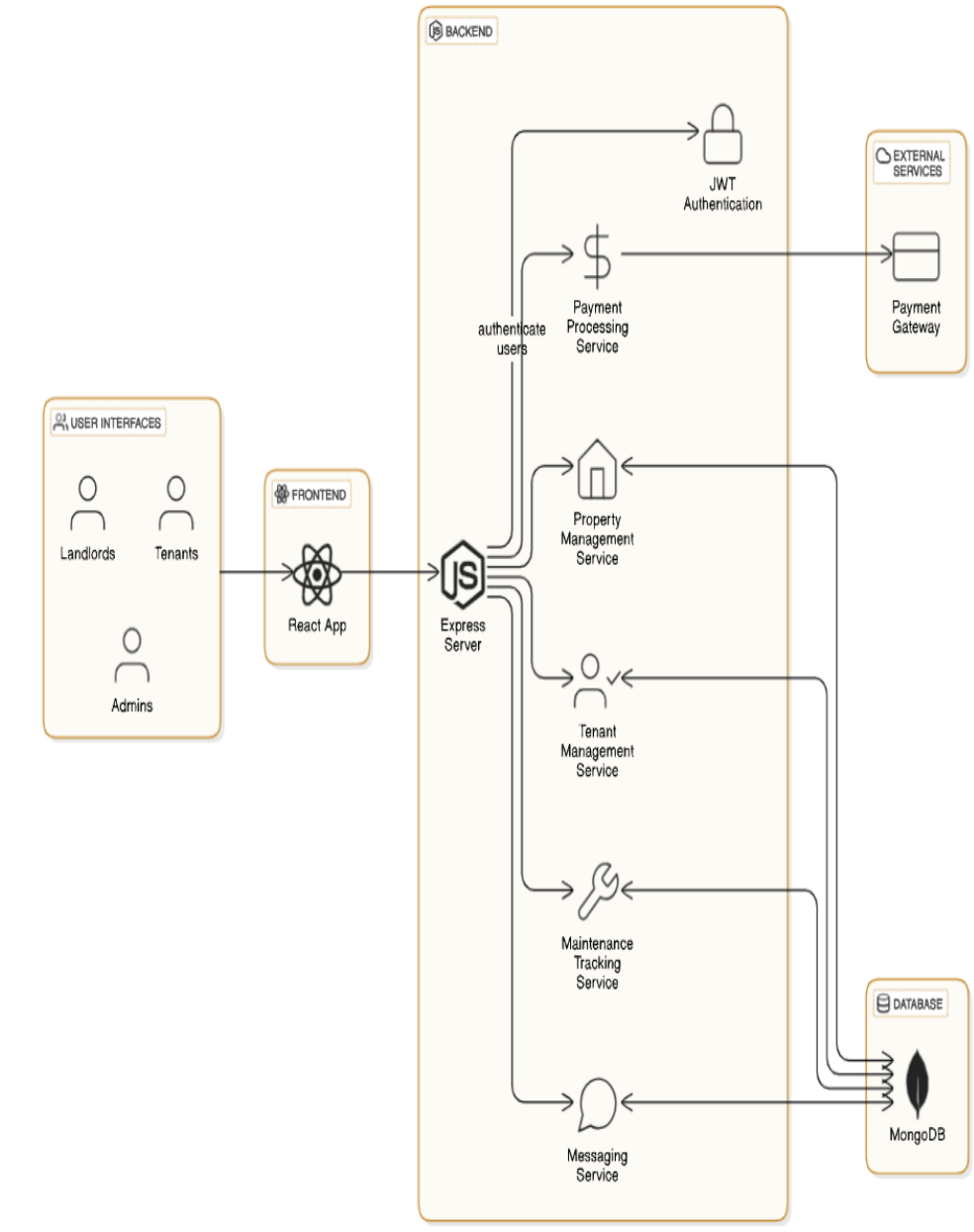
- Built on the MERN stack, the app is scalable to support growing user bases.

13. Search History and Recommendations

- Tenants can view recent searches and receive recommendations for properties.
- Smart algorithms suggest properties based on preferences and browsing behaviour.

CHAPTER 3

ARCHITECTURE



3.1 FRONTEND

The front-end of the Rentify House Rental App is developed using React.js, a powerful JavaScript library for building user interfaces. The architecture follows a component-based design to ensure modularity, reusability, and scalability. Below is a detailed description of the front-end architecture:

1. Component-Based Structure

The app is divided into small, reusable components, each responsible for a specific functionality. This modular approach simplifies maintenance and testing.

Core Components:

1. Navbar: Provides navigation across pages such as Home, Profile, and Manage Properties.
2. Chat Component: Enables real-time messaging between landlords and tenants.
3. Property Cards: Displays property details, including images, prices, and descriptions.
4. Search Bar: Allows users to filter properties based on various criteria like location, price, and type.

Pages:

Home Page, Search Results Page, Manage Properties Page, Profile Page, etc., are built using combinations of components.

2. State Management

State management is crucial for handling dynamic data like user information, property details, and search filters.

1. React's Local State: Manages component-specific states (e.g., form inputs, modal visibility).
2. Redux:A global state management library is used for sharing data across components, such as user authentication status, property listings, and chat data.

3. Routing with React Router

The app uses React Router for handling navigation and routing between pages:

1. Public Routes: Accessible to all users (e.g., Home, Property Search).
2. Private Routes: Restricted to authenticated users (e.g., Manage Properties, Payments).
3. Dynamic routes like `/detailed-propertyInfo/:propertyId` fetch and display property-specific data.

4. Responsive Design

The front-end is designed to be mobile-responsive using CSS3 and frameworks like Bootstrap or custom media queries. This ensures seamless functionality across devices, including desktops, tablets, and smartphones.

5. API Integration

Components communicate with the back-end using RESTful APIs.

1. Axios is used for HTTP requests to fetch and send data like property details, user profiles, and payments.
2. The app handles API responses using React's asynchronous features like `useEffect` and `useState`.

6. Front-End Libraries

To enhance functionality and design, several libraries are utilized:

1. Material-UI or Chakra UI for modern and consistent UI components.
2. Firebase for handling image uploads in property listings.
3. Chart.js for visualizing data like payment history and maintenance requests on dashboards.

7. Security Features

1. Sensitive data like JWT tokens are stored securely in HTTP-only cookies or local storage.
2. The app prevents unauthorized access using protected routes.

8. Performance Optimization

1. Lazy Loading: Components and images are loaded only when needed, improving performance.
2. Code Splitting: The app uses Webpack for breaking code into smaller chunks, ensuring faster load times.
3. Caching: Static assets are cached to minimize API calls.

9. Continuous Integration

The front-end code is integrated with GitHub Actions or similar CI/CD tools for testing and deployment.

3.2 BACKEND

The backend architecture for the Rentify project is built with Node.js and Express.js, enabling core functionalities such as property management, user authentication, payment processing, and real-time chat. The design follows a modular structure for maintainability and scalability, with distinct directories for configurations, controllers, middleware, models, routes, services, and utilities.

1. Database Design

The application uses MongoDB as its database, with Mongoose for schema modelling. Key entities include:

- **User**: Represents landlords, tenants, and administrators.
- **Property**: Manages property listings and details.
- **Booking**: Tracks rental agreements and payment transactions.
- **Message**: Stores real-time chat data for persistence.

2. Authentication & Authorization

Authentication is implemented using JSON Web Tokens (JWT) for secure, session less access. Role-based access control ensures that different user types (landlords, tenants, admins) have the appropriate permissions. Middleware handles token validation for protected routes.

3. API Endpoints

The API is organized into well-defined endpoints for various functionalities:

- **/api/auth:** Manages user login and registration.
- **/api/properties:** Handles property listing creation, retrieval, updates, and deletion.
- **/api/bookings:** Manages rental agreements and retrieves booking details.
- **/api/payments:** Facilitates secure payment processing.
- **/api/chat:** Provides endpoints for fetching and sending real-time chat messages.

4. Middleware

Middleware components enhance functionality and security:

- **Authentication Middleware**: Validates JWT tokens and user roles.
- **Error Handling Middleware**: Formats and returns HTTP error responses.
- **Request Validation**: Ensures incoming data complies with defined schemas using libraries like Joi or Express Validator.

5. Business Logic Services

The backend logic is encapsulated in modular services:

- **User Service**: Manages user data and authentication processes.
- **Property Service**: Handles property-related CRUD operations.
- **Booking Service**: Processes booking requests and validations.
- **Payment Service**: Integrates with payment gateways like Stripe or PayPal.
- **Chat Service**: Manages real-time messaging and data retrieval.

6. Real-Time Chat System

The chat system is powered by Socket.IO for real-time communication. It handles events for joining chat rooms, sending messages, and delivering notifications. All messages are stored in the database for persistence and later retrieval.

7. Deployment Setup

The backend uses environment-specific configurations stored in .env files. Deployment platforms like AWS, Heroku, or DigitalOcean host the application, while MongoDB Atlas is used for database hosting. The system is designed to work seamlessly in development, testing, and production environments.

8. Scalability and Security

Scalability is addressed through techniques like load balancing, database indexing for faster queries, and asynchronous task processing using BullMQ for handling background jobs, such as email notifications. Security measures include input sanitization to prevent SQL/NoSQL injection, HTTPS for secure communication, and role-based access control to protect sensitive operations.

3.3 DATABASE

3.3.1 DATABASE SCHEMA

The database schema for Rentify is designed with key collections to support the core functionalities of the application. These include:

- **User**

  Stores details about users, such as landlords, tenants, and administrators. Key fields include name, email, password, role, profile picture, and contact number. Role-based differentiation ensures access control for various user types.
- **Property**

  Manages property listings with details such as title, description, address, city, price, associated landlord, and availability status. It also stores an array of image URLs to showcase the property.
- **Booking**

  Tracks rental agreements, associating each booking with a property and a tenant. Key fields include the rental period (start and end dates), total payment amount, and payment status.

- **Message**

  Facilitates real-time chat between users by storing messages, including the sender, receiver, chat room identifier, message content, and timestamps.

3.3.2 DATABASE INTERACTIONS

- **User**

  Includes operations like registering a new user, finding a user by email for authentication, and updating user profiles. Role-based queries ensure the system only retrieves or modifies appropriate data.
- **Property**

  Covers creating property listings, fetching properties for browsing, and updating property availability after a successful booking.
- **Booking**

  Manages booking creation, retrieving bookings for tenants or landlords, and updating payment statuses after processing payments.
- **Message**

  Handles real-time message storage and retrieval, ensuring messages are correctly grouped by chat room and sorted chronologically for seamless communication.

3.3.3 RELATIONSHIPS BETWEEN COLLECTIONS

- **User to Property**

  Each property is linked to a landlord, ensuring only authorized users manage their properties.
- **Property to Booking**

  Each booking references a specific property, tying rental agreements to the appropriate listing.
- **User to Message**

  Messages are associated with both the sender and receiver, enabling accurate chat room management and retrieval.

3.3.4 INDEXES FOR PERFORMANCE OPTIMIZATION

- **User**

  A unique index on the email field ensures efficient authentication and avoids duplicate registrations.
- **Property**

  Indexes on fields like city and availability improve search and filtering speed when browsing properties.
- **Message**

  Indexes on chat room identifiers and timestamps ensure efficient retrieval of chat histories.

CHAPTER 4

SETUP INSTRUCTIONS

Follow these steps to set up the project on your local machine for development or testing purposes

4.1 PREREQUISITES

Before you begin, make sure the following software is installed on your system:

1. **Node.js**
    - Required for running the server and React frontend.
    - Download Nodejs
    - Recommended version: 14+
2. **MongoDB**
    - Used as the database for storing property and user data.
    - Download MongoDB
    - Ensure it runs on localhost.
3. **Git**
    - Used to clone the project repository.
    - Download Git
4. **npm (Node Package Manager) or yarn**
    - Comes with Node.js. Use either for installing dependencies.
5. **Code Editor**
    - Visual Studio Code or another preferred IDE.

4.2 INSTALLATION

Follow these steps to set up the project.

Step 1: Clone the Repository

Open a terminal or command prompt and run the following command to clone the Rentify project from GitHub:

git clone

Step 2: Navigate to the Project Directory

Change the directory to the project folder:

cd rentify

Step 3: Install Backend Dependencies

Move to the backend folder and install the necessary packages:

cd ../frontend
npm install

Step 4: Install Frontend Dependencies

Now, move to the frontend folder and install the required packages:

cd ../frontend
npm install

4.3 CONFIGURE ENVIRONMENTAL VARIABLE

Create a .env file inside the backend folder to store environment variables.

Add the following content to your .env file:

```
# MongoDB connection string (URL to your MongoDB database)
MONGO_URI=mongodb://localhost:27017/rentify

# Port number where the backend server will run
PORT=5000

# Secret key used for JWT token generation
JWT_SECRET=your_jwt_secret

# API keys for the payment gateway (if integrated)
PAYMENT_API_KEY=your_payment_api_key
PAYMENT_SECRET_KEY=your_payment_secret_key

# Cloud storage service API keys (if applicable)
CLOUDINARY_API_KEY=your_cloudinary_api_key
CLOUDINARY_SECRET_KEY=your_cloudinary_secret_key
```

Replace the placeholder values (your_jwt_secret, your_payment_api_key, etc.) with your actual values.

4.4 RUNNING THE APPLICATION

Step 1: Start MongoDB

Ensure MongoDB is running on your system. In most cases, you can simply run:

```
mongod
```

Step 2: Start the Backend Server

Move to the backend directory and start the server:

```
cd backend
npm run dev
```

This will start the backend server on http://localhost:5000.

Step 3: Start the Frontend

```
cd ../frontend
npm start
```

This will start the frontend on http://localhost:3000.

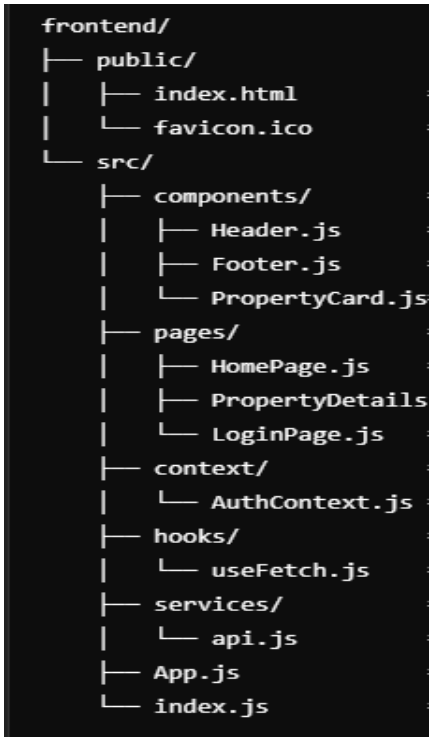
CHAPTER 5

FOLDER STRUCTURE

This section provides a detailed explanation of the project folder structure, divided into two main parts: the Client (React frontend) and the Server (Node.js backend).

5.1 CLIENT

The frontend folder contains the source code for the React application.



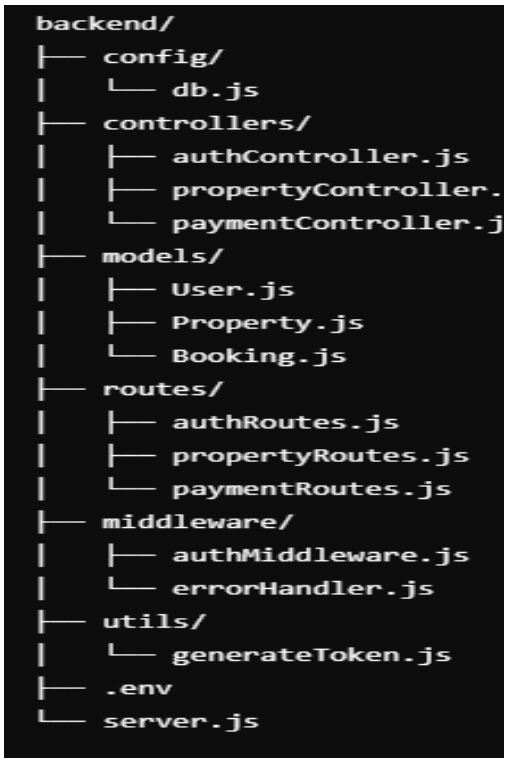
Explanation:

- **components/:** Contains small, reusable UI components used across different pages (e.g., buttons, forms, cards).
- **pages/:** Contains full-page components that correspond to different routes (e.g., HomePage, LoginPage).
- **context/:** Handles global state (e.g., user authentication) using React's Context API.
- **hooks/:** Custom hooks to encapsulate logic, making components cleaner and reusable.

- **services/:** Contains functions that interact with backend APIs (e.g., fetching data or posting forms).
- **App.js:** The main component where routes and global providers are defined.

5.2 SERVER

The backend folder contains the code for the backend server built with Node.js and Express.



Explanation:

- **config/:** Contains configuration files, such as database connection settings.
- **controllers/:** Contains functions that handle the business logic for different API endpoints.
- **models/:** Defines Mongoose schemas and models for MongoDB collections (e.g., User, Property).
- **routes/:** Defines Express routes that map to specific controller functions.
- **middleware/:** Contains custom middleware for tasks like authentication and error handling.

- **utils/:** Contains utility functions (e.g., token generation for authentication).
- **server.js:** The main file that initializes the Express server and connects to the database.

CHAPTER 6

RUNNING THE APPLICATION

To run the Rentify application, you need to start both the frontend and backend servers separately. Each server performs a specific role:

- **Frontend (React):** This is the user interface of the application, where users interact with features like searching for properties, booking rentals, and making payments.
- **Backend (Node.js):** This is the server-side logic that handles requests from the frontend, processes data, interacts with the database (MongoDB), and returns responses.

6.1 BACKEND SERVER

The backend server is built using Node.js and uses Express.js to manage API endpoints. It connects to a MongoDB database to store and retrieve data such as user information, property listings, and bookings.

Steps to Start the Backend:

1. **Navigate to the Backend Directory:**
   First, you need to move into the folder where the backend code is stored. This folder typically contains files like server.js, the models, routes, and controllers directories.
2. **Start the Backend Server:**
   Use a command to launch the server. This starts the backend application, making it listen for incoming requests on a specific port (default is usually 5000).
3. **What Happens When the Backend Starts:**
   - The backend server connects to the MongoDB database.
   - It sets up routes (URLs) to handle incoming API requests (e.g., /api/properties to fetch property listings).

- It listens for HTTP requests, processes them, and returns appropriate responses such as JSON data or error messages.

Indicators of a Successful Backend Start:

- You'll see a message like:
  - *"Server running on port 5000..."*
  - *"Connected to MongoDB..."*

If these messages appear, it means the backend is ready to handle requests.

6.2 FRONTEND SERVER

The frontend server is built using React, a JavaScript library for building user interfaces. It interacts with the backend server through API calls to display data and perform actions like user authentication or property booking.

Steps to Start the Frontend:

1. **Navigate to the Frontend Directory:**
   Move to the folder where the frontend code is located. This folder contains the src directory with React components, App.js, and index.js.
2. **Start the Frontend Server:**
   Use a command to start a development server for the React application. This launches the frontend on a specific port (default is usually 3000).
3. **What Happens When the Frontend Starts:**
   - The React application compiles and runs a development server.
   - It opens a web browser window to display the application's user interface.
   - It establishes communication with the backend to fetch and display dynamic data, such as property listings or user profiles.

Indicators of a Successful Frontend Start:

- You'll see a message like:
  - *"Compiled successfully!"*
  - *"You can now view the application in your browser at http://localhost:3000."*

If this message appears, the frontend is running and ready for interaction.

6.3 ACCESSING THE APPLICATION

Once both servers are running:

- Open your browser and go to http://localhost:3000. This will display the frontend user interface, where you can explore different features of the Rentify app.
- The frontend communicates with the backend through API requests, typically sent to http://localhost:5000. The backend processes these requests and returns the necessary data.

CHAPTER 7

API DOCUMENTATION

The Rentify backend exposes several API endpoints to manage properties, users, bookings, and payments. Below is the detailed documentation of each endpoint, including the request methods, parameters, and example responses.

1. Authentication

These endpoints manage user registration, login, and authentication.

POST /api/auth/register

Registers a new user.

- **Request Body:**

```
{
  "name": "James Auther",
  "email": "james2000@gmail.com",
  "password": "james-2000"
}
```

- **Response:**

```
{
  "message": "User registered successfully",
  "user": {
    "id": "28j8aut2000000",
    "name": "James Auther",
    "email": "james2000@gmail.com",
    "token": "jwt_token_here"
  }
}
```

POST /api/auth/login

Logs in a user and returns a token.

- **Request Body:**

```
{
  "email": "james2000@gmail.com",
  "password": "james-2000"
}
```

- **Response:**

```
{
  "message": "Login successful",
  "user": {
    "id": "28j8aut2000000",
    "name": "James Auther",
    "email": "james2000@gmail.com",
    "token": "jwt_token_here"
  }
}
```

2. Property

These endpoints manage property listings.

GET /api/properties

Fetches all available properties.

- **Response:**


```
[
  {
    "id": "28j8aut2000000",
    "title": "Cozy Apartment",
    "description": "A beautiful apartment in the city center",
    "location": "New York, NY",
    "pricePerNight": 120,
    "available": true
  }
]
```


POST /api/properties

Creates a new property listing (Admin only).

- **Request Body:**


```
{
  "title": "Luxury Villa",
  "description": "A luxury villa with a pool",
  "location": "Los Angeles, CA",
  "pricePerNight": 500
}
```


- **Response:**


```
{
  "message": "Property created successfully",
  "property": {
    "id": "63h6the789012",
    "title": "Luxury Villa",
    "description": "A luxury villa with a pool",
```

```
    "location": "Los Angeles, CA",
    "pricePerNight": 500,
    "available": true
  }
}
```


3. Booking

These endpoints handle booking properties.

POST /api/bookings

Creates a new booking.

- **Request Body:**


```
{
  "propertyId": "63h6the789012",
  "userId": "28j8aut2000000",
  "checkInDate": "2024-12-01",
  "checkOutDate": "2024-12-07"
}
```


- **Response:**


```
{
  "message": "Booking created successfully",
  "booking": {
    "id": "63h6the789012",
    "propertyId": "63f6abc123456",
    "userId": "28j8aut2000000",
    "checkInDate": "2024-12-01",
    "checkOutDate": "2024-12-07",
    "status": "Confirmed"
  }
}
```

GET /api/bookings/:userId

Fetches all bookings for a specific user.

- **Response:**

```
[
  {
    "id": "63f7ghj890123",
    "property": {
      "title": "Cozy Apartment",
      "location": "New York, NY"
    },
    "checkInDate": "2024-12-01",
    "checkOutDate": "2024-12-07",
    "status": "Confirmed"
  }
]
```

4. Payment

These endpoints handle payments for bookings.

POST /api/payments

Processes a payment for a booking.

- **Request Body:**

```
{
  "bookingId": "63f7ghj890123",
  "amount": 840,
  "paymentMethod": "Credit Card"
}
```

- **Response:**


```
{
  "message": "Payment processed successfully",
  "payment": {
    "id": "63f8pay901234",
    "bookingId": "63f7ghj890143",
    "amount": 840,
    "paymentMethod": "Credit Card",
    "status": "Completed"
  }
}
```


5. User Profile

These endpoints manage user profiles and account settings.

GET /api/users/:userId

Fetches user profile information.

- **Response:**


```
{
  "id": "28j8aut2000000",
  "name": "James Auther",
  "email": "james2000@gmail.com",
  "bookings": [
    {
      "id": "63f7ghj890123",
      "property": "Cozy Apartment",
      "status": "Confirmed"
    }
```

CHAPTER 8

AUTHENTICATION

In the Rentify project, authentication and authorization are essential components that ensure users can securely access the platform and perform actions based on their roles (e.g., regular user or admin). This system is implemented using JSON Web Tokens (JWT) for stateless authentication.

8.1 AUTHENTICATION

Authentication verifies a user's identity through a token-based system, allowing for secure and stateless communication between the client and server.

1. **User Registration (Sign Up):**
    - Users provide their name, email, and password during registration.
    - Passwords are hashed using bcrypt before being stored in the database.
2. **User Login:**
    - Users log in with their email and password.
    - Upon successful authentication, a JWT is generated and returned to the client.
3. **Token Structure:**

    The JWT includes:
    - **Header:** Specifies the token type and hashing algorithm.
    - **Payload:** Contains user details (e.g., user ID and role).
    - **Signature:** Ensures the token's integrity.
4. **Token Storage:**

    Tokens are stored on the client side in:
    - **Local Storage** (persistent storage).
    - **HTTP-Only Cookies** (for enhanced security).

8.2 AUTHORIZATION

Authorization controls user access to specific resources and actions based on their role.

1. **User Roles:**
    - **Regular User:** Can browse properties, make bookings, and manage their profile.
    - **Admin:** Can manage properties, view all bookings, and access admin-specific features.
2. **Role-Based Access Control (RBAC):**
   Access to certain API endpoints is restricted based on user roles.

8.3 MIDDLEWARE FUNCTIONS

Authentication Middleware:

Verifies the JWT token and attaches the user's information to the request.

Authorization Middleware:

Checks if the user has the required role to access specific endpoints.

CHAPTER 9

USER INTERFACE

1. Landing Page

Purpose:

The landing page serves as the first interaction point for users, providing an overview of the platform's features and a navigation option to either explore properties or log in.

Key Features:

- Hero section with a call-to-action (CTA) to explore properties.
- Quick search functionality for location-based property browsing.
- Overview of benefits for both landlords and tenants.

2. Sign-Up Page

Purpose:

Allows new users to register on the platform by providing personal information.

Key Features:

- Input fields for name, email, and password.
- Role selection (Tenant or Landlord).
- Integration with validation and error feedback for missing or incorrect inputs.

3. Login Page

Purpose:

Enables existing users to log in and access their dashboards.

Key Features:

- **E**mail and password fields.

- "Forgot Password" link for resetting credentials.
- Secure authentication with JWT.

4. Property Detailed View Page

Purpose:

Displays comprehensive details about a selected property.

Key Features:

- High-resolution images of the property.
- Property details such as price, amenities, location, and availability.
- Contact options for landlords and a "Book Now" button.

5. Maintenance Request Page

Purpose:

Allows tenants to report maintenance issues to the landlord.

Key Features:

- Input form to describe the issue.
- Status tracking of maintenance requests.
- Real-time chat support for clarification or updates.

6. Manage Property Page

Purpose:

A dashboard for landlords to manage their property listings.

Key Features:

- List of all properties owned by the landlord.
- Options to add, update, or delete property listings.
- Viewing property performance, such as occupancy rates and payment history.

CHAPTER 10

TESTING

The Rentify project follows a robust testing strategy to ensure the reliability, functionality, and performance of both the frontend and backend components. This strategy includes unit testing, integration testing, and end-to-end (E2E) testing using modern testing frameworks and tools.

10.1 TESTING STRATEGY

The testing strategy for Rentify is divided into the following levels:

a. Unit Testing

Focuses on testing individual components, functions, and modules in isolation to ensure they behave as expected.

- **Frontend:** Tests React components, utility functions, and hooks.
- **Backend:** Tests individual functions, API routes, and database operations.

b. Integration Testing

Verifies that different modules and components work together as expected.

- **Frontend:** Ensures the integration between components and external APIs.
- **Backend:** Tests the interaction between the server, database, and third-party services.

c. End-to-End (E2E) Testing

Simulates user interactions with the application to test the entire system from the user interface to the backend.

10.2 TOOLS USED FOR TESTING

Frontend Testing Tools

1. **Jest**
    - A popular JavaScript testing framework used for unit and integration testing of React components.
    - **Features:**
        - Snapshot testing to detect UI changes.
        - Mocking functions and API requests.
2. **React Testing Library**
    - A library built on top of Jest to test React components by simulating user interactions and checking for expected DOM changes.
    - **Features:**
        - Focus on user-centric testing.
        - Queries DOM elements using accessibility roles.
3. **Cypress (for E2E Testing)**
    - A powerful end-to-end testing framework that simulates user interactions in a real browser environment.
    - **Features:**
        - Fast and reliable test execution.
        - Real-time reloading for faster development.

Backend Testing Tools

1. **Mocha**
    - A feature-rich JavaScript testing framework for Node.js applications.
    - **Features:**
        - Asynchronous testing with descriptive test cases.
        - Works well with assertion libraries like Chai.

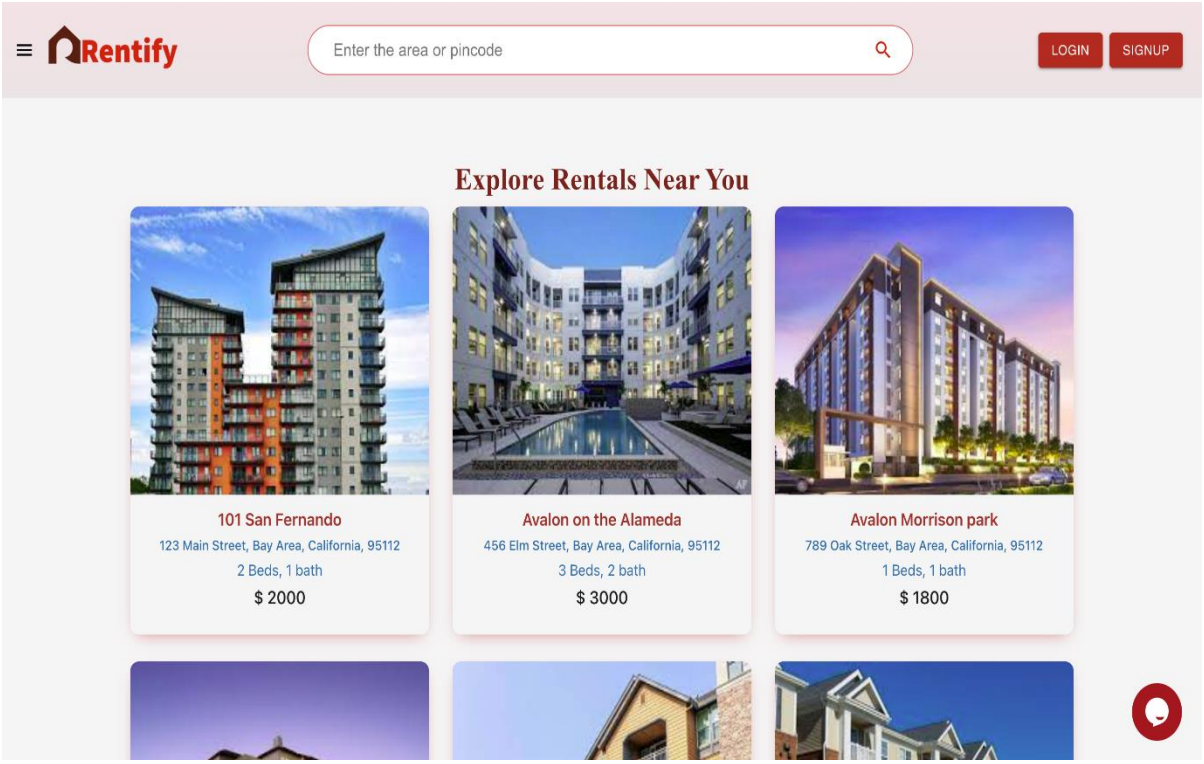
2. **Chai**
    - An assertion library used with Mocha to write human-readable test cases.
    - **Features:**
        - Provides expect, should, and assert styles for writing tests.

3. **Supertest**
    - A library for testing HTTP endpoints in Node.js applications.
    - **Features:**
        - Simulates API requests and verifies responses.
        - Easily integrates with Mocha and Chai.
4. **MongoDB Memory Server**
    - A tool to run a temporary in-memory MongoDB instance for testing database interactions.
    - **Features:**
        - Fast and isolated testing environment.
        - No need for an actual MongoDB instance during tests.

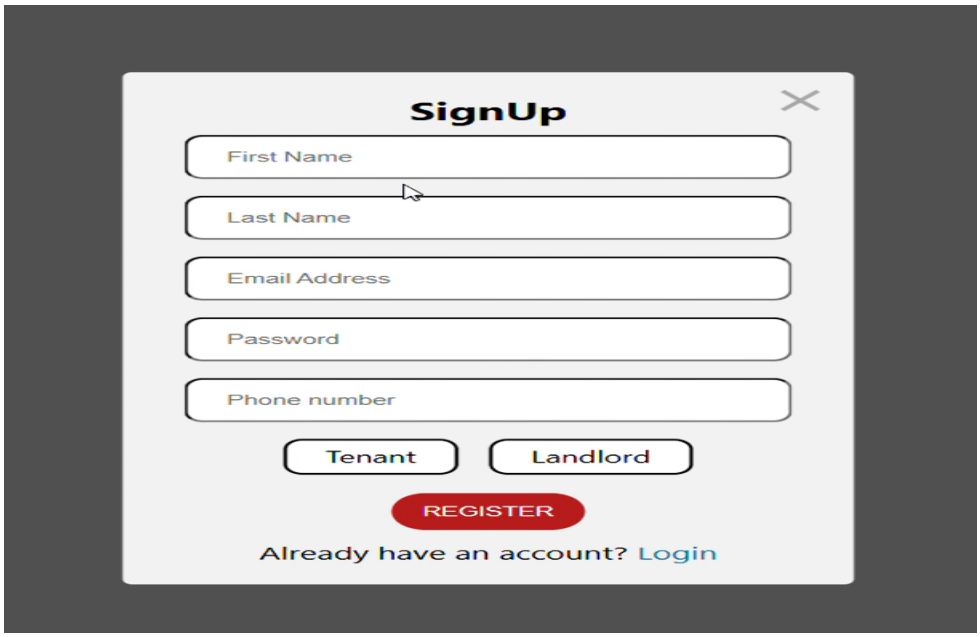
CHAPTER 11

SCREENSHOTS

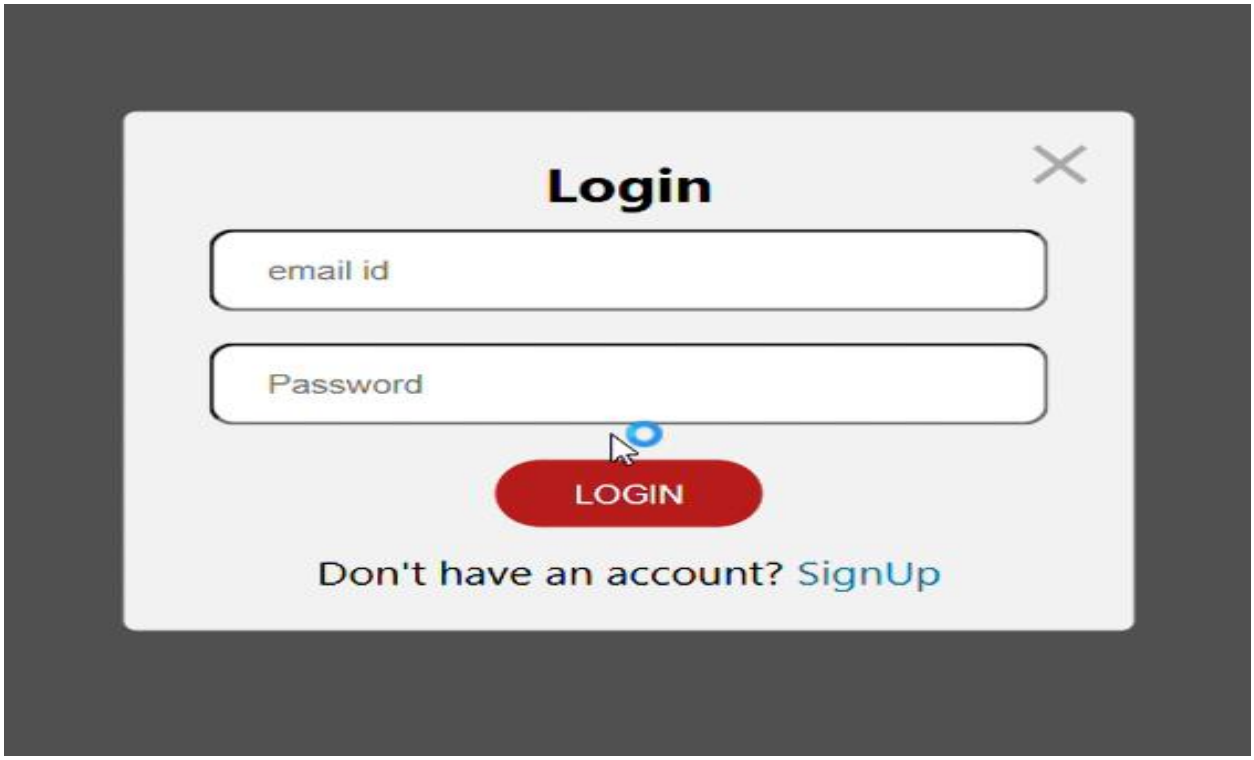
Landing page



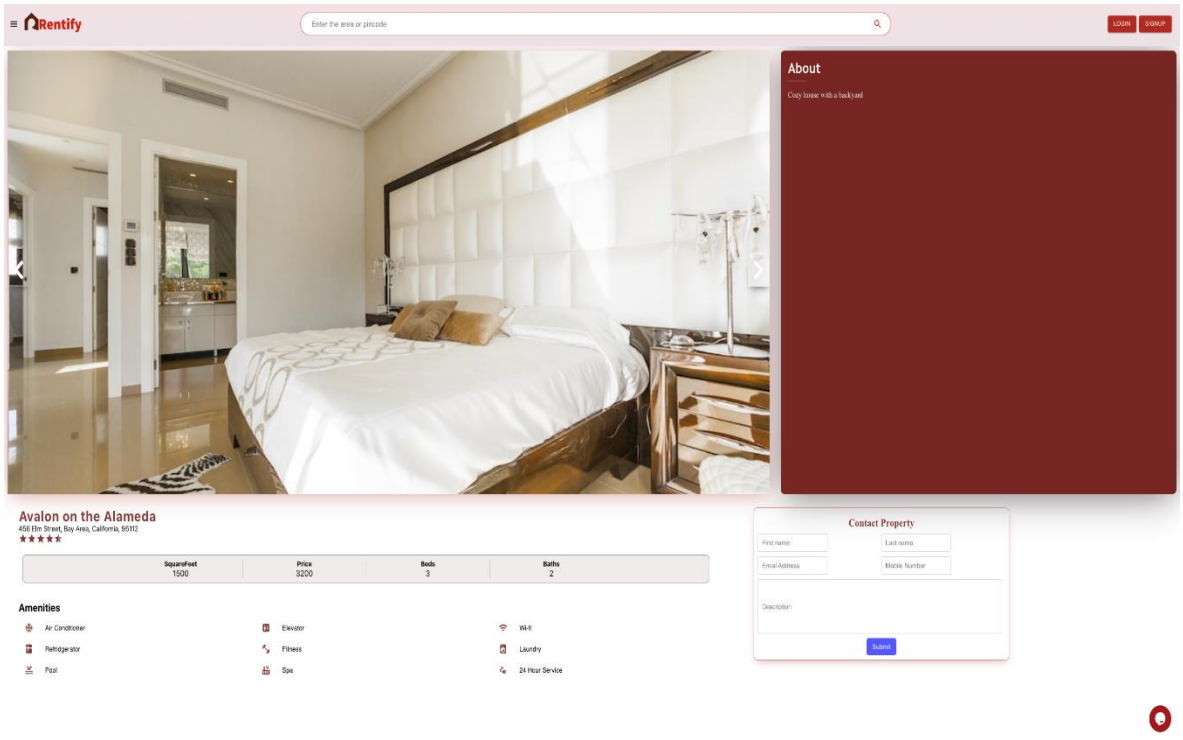
Sign up page



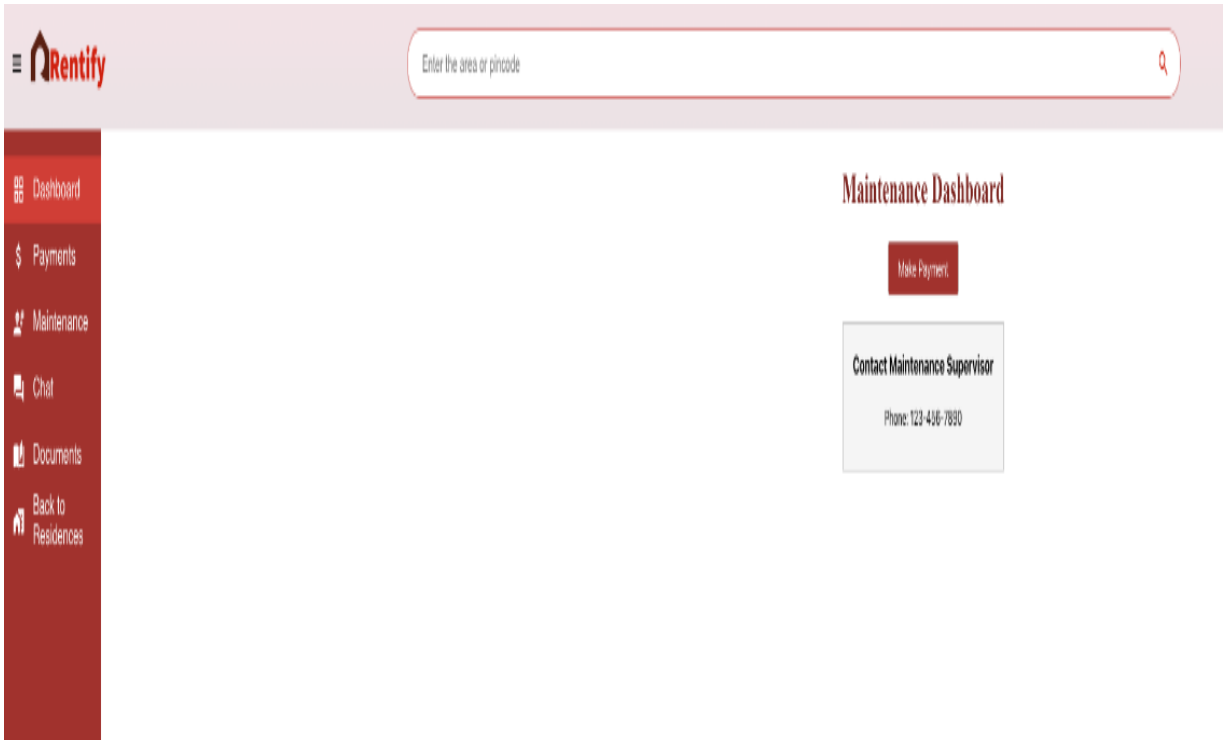
Login page



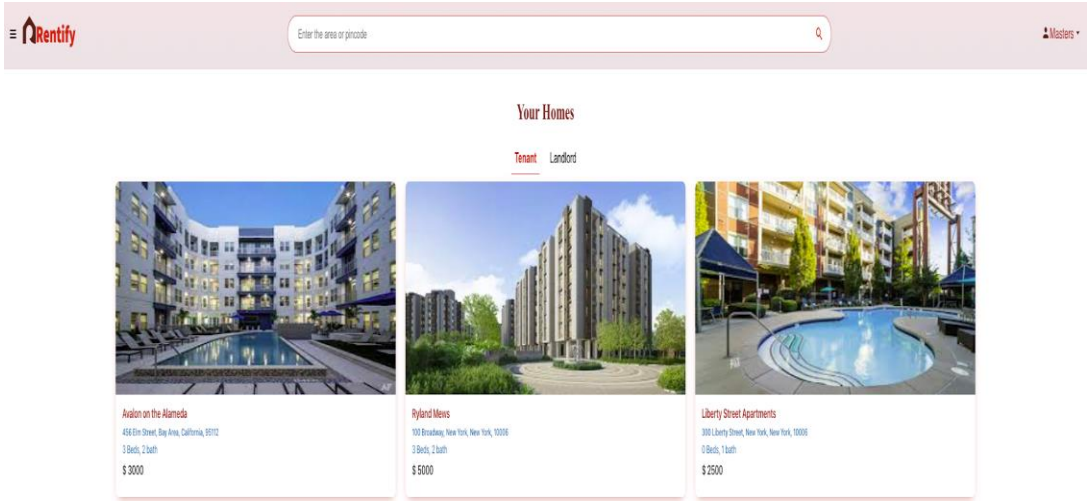
Property detailed view page



Maintenance page



Manage property page



CHAPTER 12

KNOWN ISSUES

This section lists the current known bugs and issues in the Rentify project, along with their potential impact and any temporary workarounds. Developers and users should be aware of these issues while using or contributing to the project.

1. **JWT Expiry Handling**
    - **Issue:** No automatic token refresh mechanism.
    - **Impact:** Users are logged out unexpectedly when their session expires.
    - **Workaround:** Users must manually log in again. A future update will implement token refresh functionality.
2. **Image Upload Size Limitation**
    - **Issue:** Uploads larger than 5MB fail without an error message.
    - **Impact:** Users may not understand why their upload failed.
    - **Workaround:** Compress images before uploading.
3. **Booking Calendar Responsiveness**
    - **Issue:** Calendar is not mobile-friendly.
    - **Impact:** Mobile users may face difficulties selecting dates.
    - **Workaround:** Use desktop devices; mobile optimization is planned.
4. **Delayed Notifications**
    - **Issue:** Notifications for new messages are sometimes delayed.
    - **Impact:** Users may miss important updates.
    - **Workaround:** Manual refresh; real-time service optimization is ongoing.

CHAPTER 13

FUTURE ENHANCEMENTS

Future enhancements for the Rentify platform could include the development of mobile applications for iOS and Android, providing users with a more personalized, on-the-go experience. Integrating AI-powered property recommendations would help tenants find properties that match their preferences more efficiently. The platform could also automate lease agreement generation with customizable templates and digital signatures, simplifying the rental process. Advanced analytics and reporting tools for landlords would offer insights into property performance, while expanding payment options, including cryptocurrency and international gateways, would broaden the platform's accessibility. Additionally, integrating smart home devices, enabling tenant and landlord reviews, and offering document management and e-signing would further improve user experience. Expanding the maintenance workflow, adding multi-language support, offering tenant insurance, and fostering community-building features would make Rentify a more robust, globally accessible, and user-friendly platform, meeting the needs of both tenants and landlords while driving future growth.