Ideation Phase

Empathize & Discover

Date	13 Feb 2026
Team ID	LTVIP2026TMIDS87052
Project Name	House Hunt: Finding Your Perfect Rental Home
Maximum Marks	4 Marks

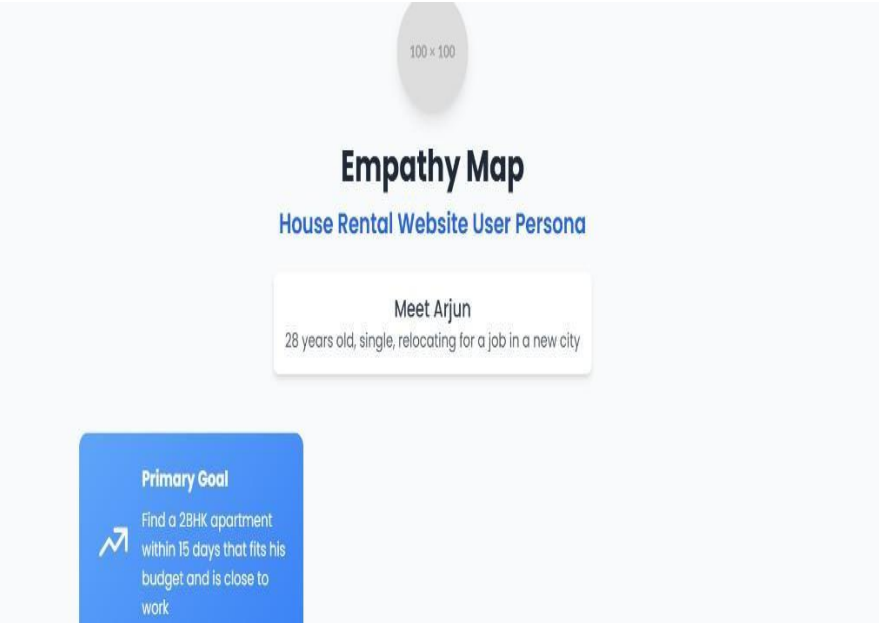
Empathy Map :

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.

It is a useful tool to helps teams better understand their users.

Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

EXAMPLE: **House Hunt: Finding Your Perfect Rental Home**



Say & Do

- "Why is it so hard to find genuine property listings?"
- Asks friends for recommendations/reviews
- Regularly checks notifications for new listings
- Saves listings to review later

Pain Points

- Fake or misleading property listings
- Poor filtering options that waste time
- Difficulty scheduling visits or contacting landlords

Goals

- Find a property within budget and timeline
- Minimize time spent on irrelevant listings
- Ensure accuracy of property descriptions
- Proximity to workplace

Key Insights for Rental Website Design

What Arjun Needs:

- Verified, accurate property listings
- Advanced filtering options
- Neighborhood safety and commute information
- Easy landlord communication system

Potential Solution Features:

- 360° virtual tours of properties
- Verified badge for trustworthy listings
- AI-powered recommendation system
- Integrated schedule viewing feature

House Rental Website Persona - Empathy Map for Design Thinking

Think & Feel

- "Will I find a property that suits my budget and preferences?"
- "Is the neighborhood safe and well-connected?"
- "What if the property looks different than the photos online?"
- Feels overwhelmed by options but anxious about missing deals

Hear

- "You need to act fast in this rental market" (friends/colleagues)
- "This website is great for verified listings" (social media)
- "The best properties get booked quickly"

See

- Listings that lack clarity or detailed descriptions
- Confusing navigation on competing platforms
- Ads for premium services

Reference: https://www.mural.co/templates/empathy-map-canvas

THINK and FEEL

- Will I find a property that suits my budget and preferences?
- is the neighborhood safe and well-connected?
- What if the property looks different than the photos online?

HEAR

- "You need to act fast in this rental market." (from friends./
- "This website is great for verified listings." (from social reviews)
- "The best properties get booked quickly

Arjun

A 28-year-old relocating for a job

SEE

- Listings that lack clarity or detailed descriptions
- Confusing navigation on competing platforms
- Ads for premium services on rental platforms

SAY and DO

- "Why is it so hard to fund genuine property listings?
- Asks friende for recommendations or reviews about the website.
- Regularly checks notifications for new property listings
- Saves listings to review later or to share with family/friends

PAIN POINTS

- Fake or misleading property listings.
- Poor filtering options that waste tim e.
- Difficuity in scheduling property visits or contacting landlords

GOALS

- Find a property within his budget and move-in timeline
- Idinimize time spent on irrelevant or fake listings

Full Stack Development Documentation

Project Title: House Hunt: Finding Your Perfect Rental Home

:

Submitted by
Team Id: **LTVIP2026TMIDS87052**

Table of Contents:

1. Introduction

In today's technology-driven environment, traditional processes like renting a home are being redefined through digital innovation. The process of finding rental properties—once is dominated by manual searches, broker involvement, and word-of-mouth—often proves inefficient, time-consuming, and lacking in transparency. To address these challenges, this project introduces a full-stack web application titled **"House Hunt – Finding Your Perfect Rental Home."**

This application is built using the **MERN stack**—**MongoDB**, **Express.js**, **React.js**, and **Node.js**—and is designed to create a streamlined platform where **tenants**, **property owners**, and **administrators** can interact seamlessly. The system enables users to register based on their roles, browse and list properties, book rental homes, and manage platform activities efficiently.

By incorporating modern web technologies, secure authentication mechanisms, and responsive user interface design, **House Hunt** offers a centralized solution that eliminates the need for intermediaries while enhancing accessibility, security, and ease of use.

1.1 Project Title:

House Hunt – Finding Your Perfect Rental Home

The title "House Hunt – Finding Your Perfect Rental Home" reflects the core objective of the project — to assist users in efficiently locating, listing, and managing rental properties through a centralized digital platform. It represents a real-time solution tailored for tenants, property owners, and administrators to handle the complexities of the rental process without the traditional challenges of manual searching and broker dependency.

The phrase "House Hunt" emphasizes the action-oriented nature of the application — searching, browsing, and shortlisting homes — while "Finding Your Perfect Rental Home" highlights the application's goal of delivering personalized, filter-based property discovery tailored to user preferences.

This title was chosen to be simple, descriptive, and relatable for users and stakeholders, making it easy to identify the purpose and value of the project at first glance.

1.2 Team Members

The development of this project was a collaborative effort by a team of four members, each contributing to specific areas of the MERN stack application development. The division of responsibilities was done to ensure smooth progress across all phases—design, development, testing, and documentation.

Team Composition

Name	Role
Janani	Frontend Developer & UI/UX Designer
Gayathri	Backend Developer
Reshma	Testing & Bug Fixing
Srija	Documentation & Project Coordination

2. Project Overview

2.1 Purpose

The primary purpose of the *House Hunt – Finding Your Perfect Rental Home* application is to streamline the property rental process by creating a centralized, digital platform for users to search, list, and manage rental properties. Traditional methods of house hunting often involve time-consuming processes, lack of updated information, broker dependency, and inefficient communication.

This project aims to solve these challenges by:

- Providing a role-based platform for renters, property owners, and administrators.
- Ensuring secure access and smooth interaction through modern web technologies.
- Enabling renters to filter properties based on location, price, and amenities.
- Empowering property owners to manage listings and approve booking requests.
- Allowing admins to oversee the platform, verify property details, and manage user activities.

Overall, the application is designed to deliver transparency, reliability, and user convenience in the property rental ecosystem.

2.2 Key Features

The application offers the following core features:

Feature	Description
User Registration & Login	Role-based registration for renters, owners, and admin with secure authentication.
Property Listing (Owner)	Owners can list properties with images, descriptions, price, and amenities.
Property Search (Renter)	Renters can browse and filter properties based on city, price, and more.
Booking System	Renters can send booking requests for listed properties.
Booking Approval (Owner/Admin)	Owners/admins can accept or reject booking requests.
Admin Dashboard	Admins can monitor platform activity, approve listings, and manage users.
Responsive UI	Clean and responsive interface compatible with various screen sizes.
JWT Authentication	JSON Web Token-based secure login session management.
MongoDB Database	Data is stored securely in a NoSQL format, enabling fast access and scalability.
Error Handling & Alerts	System alerts and validations for form inputs, errors, and confirmations.

3. Architecture

The technical architecture of the **House Hunt – Finding Your Perfect Rental Home** project is built upon the widely used **MERN stack**, structured around a **client-server model**. The frontend (client) interacts with users, while the backend (server) manages logic, data storage, and communication. This modular and layered architecture allows for ease of development, scalability, and future extensibility.

3.1 Frontend Architecture (React.js)

The frontend of the **House Hunt** application is developed using **React.js**, a powerful JavaScript library that enables the creation of fast, interactive, and dynamic user interfaces. The frontend serves as the client-side of the application and is responsible for rendering UI elements, capturing user input, and handling all visual interactions.

Key Features:

- **Component-Based Architecture**: Reusable React components (e.g., Header, Footer, PropertyCard, SearchFilter) improve modularity and maintainability.
- **Routing**: Implemented using **React Router DOM** to manage multiple views such as Home, Login, Register, Property Listings, and Admin Panel.
- **State Management**: Uses React's useState, useEffect, and optionally Context API to manage component-level and shared state.
- **API Integration**: **Axios** is used for making HTTP requests to backend APIs for user login, property fetching, bookings, etc.
- **Responsive Design**: Layouts adapt seamlessly across desktops, tablets, and mobile devices using **CSS3**, **Bootstrap**, and **Ant Design**.

3.2 Backend Architecture (Node.js & Express.js)

The backend of the application is built using **Node.js** and **Express.js**, which together handle business logic, data operations, authentication, and routing. The backend exposes a set of **RESTful APIs** that interact with the frontend and database.

Core Functionalities:

- **API Routing**: Defined using Express Router (e.g., /api/users, /api/properties, /api/bookings)
- **Authentication**: Implemented using **JWT (JSON Web Token)** for session management and **bcryptjs** for secure password hashing.
- **Middleware**:
    - authMiddleware.js → Protects private routes

- errorHandler.js → Handles server errors gracefully
- **Role-Based Access**: Separate logic for renters, owners, and admins (e.g., owners can add listings, admins can approve them).
- **File Uploads**: Uses **Multer** for handling image uploads (property photos).

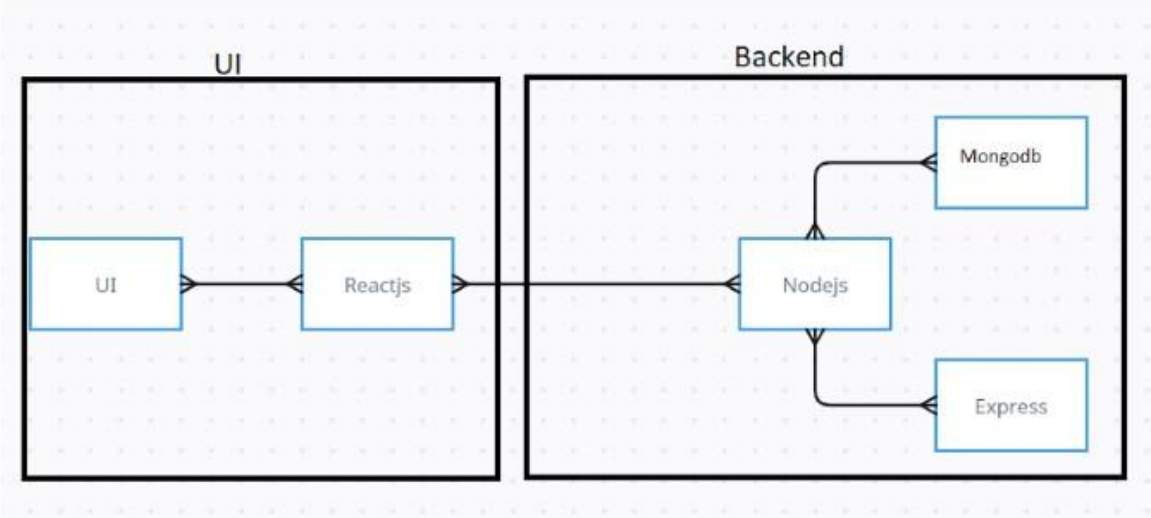
3.3 Database Design (MongoDB with Mongoose)

The database used for this application is **MongoDB**, a flexible NoSQL database well-suited for storing JSON-like data structures. **Mongoose**, an ODM (Object Data Modeling) library, is used to define and interact with schemas.

Collections and Their Roles:

1. **Users Collection**
   - Fields: name, email, password (hashed), role (renter/owner/admin)
   - Stores login credentials and user roles
2. **Properties Collection**
   - Fields: title, description, location, price, ownerId, images[], status
   - Represents rental property listings created by owners
3. **Bookings Collection**
   - Fields: propertyId, renterId, bookingDate, status (pending/approved/rejected)
   - Tracks booking requests and their approval status

TECHNICAL ARCHITECTURE



4. Setup Instructions

The following section provides a complete, step-by-step guide to setting up the House Hunt – Finding Your Perfect Rental Home application on a local development environment. The setup involves configuring both the frontend (React.js) and the backend (Node.js + Express.js), connecting to MongoDB, and preparing the necessary environment variables.

4.1 Prerequisites

Before setting up the House Hunt MERN application, ensure your development environment includes the following software and tools. These are essential for running both frontend and backend components smoothly.

Tool / Software	Purpose	Download / Notes
Node.js (v14 or above)	JavaScript runtime environment for backend & React tooling	https://nodejs.org/en/download
npm	Node package manager to install frontend/backend libraries	Comes bundled with Node.js
MongoDB	NoSQL database for storing all user/property/booking data	https://www.mongodb.com/try/download/community
Git	Version control system to manage project repository	https://git-scm.com/downloads
MongoDB Compass	GUI to view and test your MongoDB database	https://www.mongodb.com/products/compass
Visual Studio Code	Code editor (IDE) for writing and debugging source code	https://code.visualstudio.com/
Postman *(optional)*	API testing tool to verify backend endpoints	https://www.postman.com/downloads/
Nodemon *(optional)*	Auto restarts the backend server on file changes	Install via npm install -g nodemon

4.2 Installation Steps

Follow these steps to install all required dependencies and get the application running locally.

Step 1: Clone the Project Repository

Open your terminal or command prompt and run:

git clone https://github.com/your-name/house-rent.git

cd house-rent

You should now see the following folders:

house-rent/

├── frontend/ → React frontend application

└── backend/ → Node.js backend server with API logic

Step 2: Install Frontend Dependencies

cd frontend

npm install

This will install all React-based packages:

- axios – to make API calls
- react-router-dom – for page navigation
- bootstrap, antd, material-ui – for responsive design
- moment.js – for time/date handling
- mdb-react-ui-kit, react-bootstrap – for extra UI components

Step 3: Install Backend Dependencies

cd ../backend

npm install

This will install backend libraries such as:

- express – server framework
- mongoose – database interaction
- cors – to enable frontend-backend communication
- bcryptjs – password encryption
- jsonwebtoken – user authentication
- dotenv – to manage sensitive config variables
- multer – file/image upload handling
- moment – formatting dates
- nodemon *(optional)* – for auto server reloads during development

4.3 Environment Configuration

In your **backend folder**, create a new file named .env. This file stores environment variables securely and allows different configurations for dev vs production.

- Here's what to include:
- env
- PORT=5000
- MONGO_URI=mongodb://localhost:27017/househunt
- JWT_SECRET=yourSuperSecretKey
- **Note**: Keep this .env file in your .gitignore to avoid exposing it in public repositories.

MongoDB URI:

If you are using **local MongoDB**, the default URI is:

mongodb://localhost:27017/househunt

If using **MongoDB Atlas** (cloud database), replace MONGO_URI with:

mongodb+srv://<username>:<password>@cluster.mongodb.net/househunt?retryWrites=true&w=majority

Optional. env.example Template for GitHub:

Create a. env.example to help collaborators understand what variables they need to add.

env

PORT=5000

MONGO_URI=your_mongodb_connection_uri

JWT_SECRET=your_jwt_secret

By the end of this section, the development environment for the House Hunt – House Rent App is fully prepared for local execution. All necessary software tools, packages, and dependencies have been installed for both the frontend and backend, and environment variables have been configured securely using a .env file.

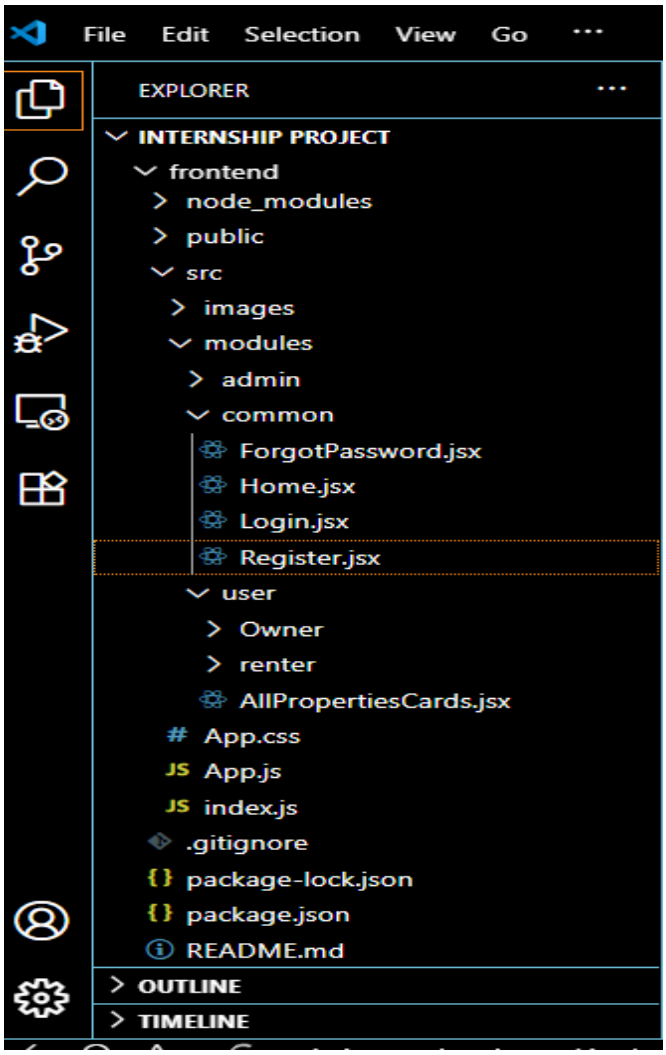
The project structure has been defined, and the MongoDB database is either set up locally or connected via MongoDB Atlas. This foundation ensures that developers can now run, test, and build upon the application with ease**.**

5. Folder Structure

The House Hunt project is organized into two primary directories — **frontend** and **backend**. Each directory follows a modular structure to ensure clear separation of concerns, ease of development, and maintainability.

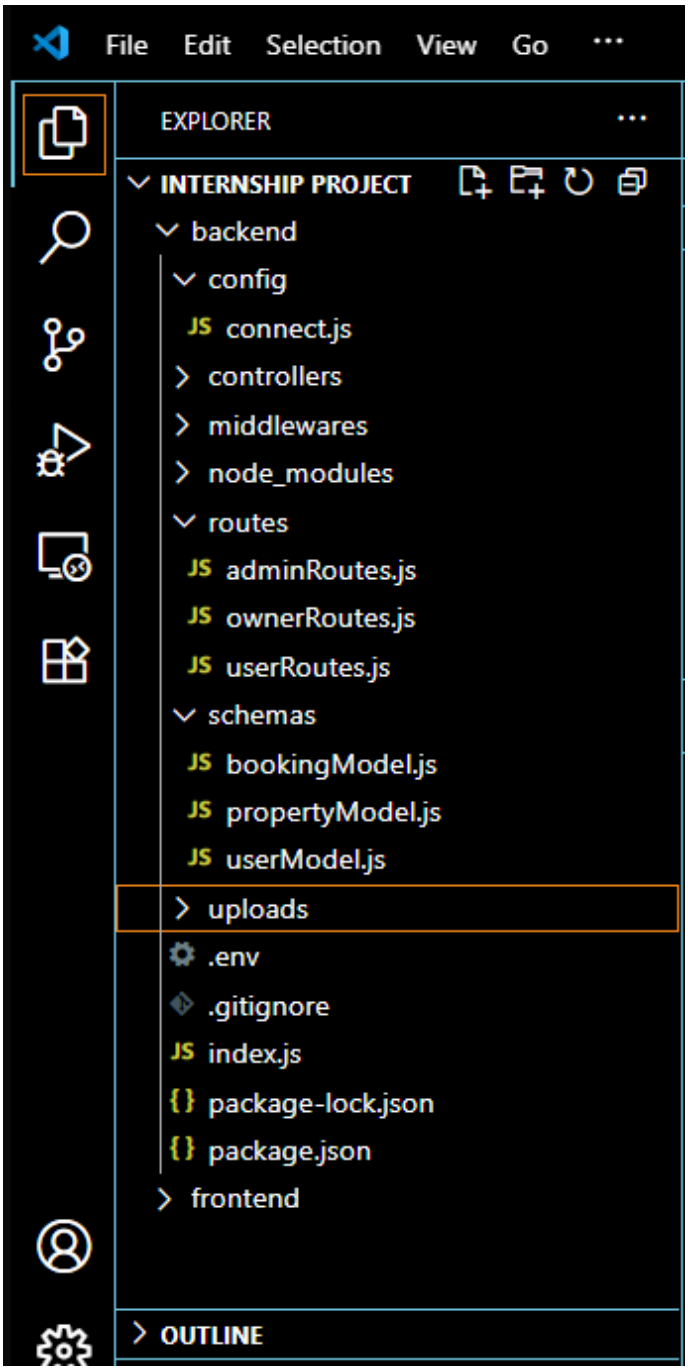
5.1 Client-Side (Frontend)

The frontend is built using **React.js**. It manages the user interface and handles communication with the backend through REST APIs using **Axios**.



5.2 Server-Side (Backend)

The backend is built using **Node.js** with the **Express.js** framework. It provides APIs for authentication, property listings, and bookings, and connects to the **MongoDB** database.



The folder structure ensures:

- A clear **separation of frontend and backend**
- Organized handling of **API logic, middleware, models**, and **UI components**
- Easy onboarding for new developers and easier debugging

This modular project layout enhances **scalability**, **readability**, and **collaboration**.

6. Running the Application

After completing the setup and configuration steps, the application is ready to be executed in a local development environment. This section explains how to run both the **frontend (React.js)** and **backend (Node.js + Express.js)** servers and validate that the system is working properly.

6.1 Frontend Server

The frontend is developed using **React.js** and handles all user interactions, form submissions, and interface rendering. It communicates with the backend using **Axios** to fetch or send data.

Steps to Run the Frontend:

1. Open a new terminal window.
2. Navigate to the frontend directory:

cd frontend

3. Install any remaining packages (if not already done):

npm install

4. Start the React development server:

npm start

Output:

- This command will automatically launch the app in your default browser.
- The frontend runs at:
  http://localhost:3000

Frontend Features:

- User login/registration interface
- Property listing and booking interface
- Responsive UI (Bootstrap + Ant Design)
- API integration using Axios

6.2 Backend Server

The backend server is built using **Node.js** with the **Express.js** framework. It is responsible for handling API routes, database communication, user authentication, and business logic.

Steps to Run the Backend:

1. Open a **separate terminal window**.
2. Navigate to the backend directory:

```
cd backend
```

3. Install required backend dependencies (if not already done):

```
npm install
```

4. Start the server using:

```
nodemon index.js
```

If nodemon is not installed globally:

```
npm install -g nodemon
```

Output:

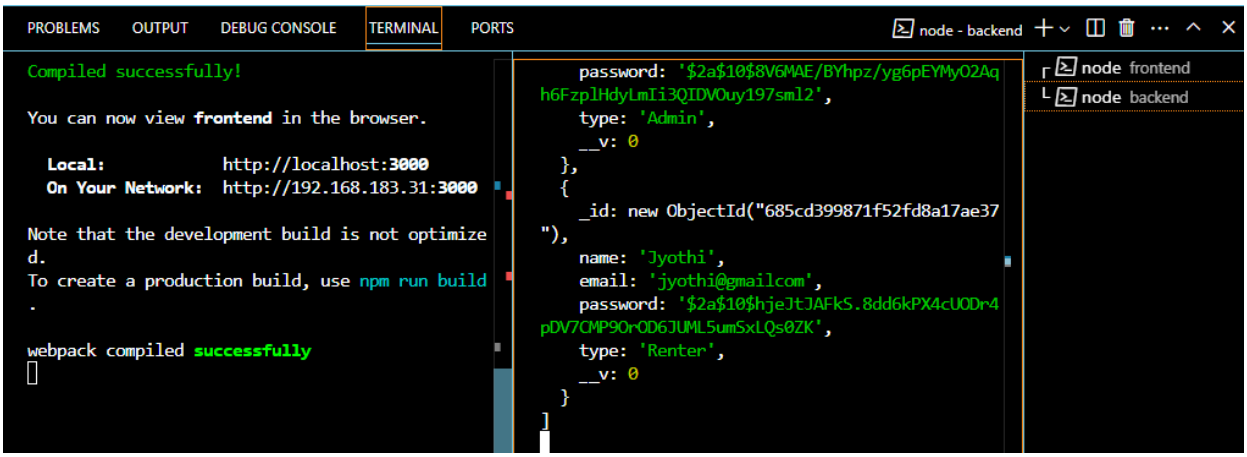
- The backend server runs at:
  http://localhost:5000
- It connects to the MongoDB database and exposes secure APIs.

Backend Features:

- RESTful API for authentication, bookings, properties, admin controls
- Token-based security using JWT
- Image/file uploads using Multer
- Role-based access (renter, owner, admin)

Confirmation of Successful Execution

Component	Expected URL	Output
Frontend	http://localhost:3000/	Loads the House Hunt UI
Backend	http://localhost:5000/	Responds to API requests (Postman/browser)



7. API Documentation

The backend of the House Hunt application exposes a set of **RESTful API endpoints** that handle authentication, property management, booking requests, and admin operations. These APIs are accessed by the frontend using **Axios** and are secured using **JWT-based authentication**.

7.1 Endpoint Descriptions

Here's a table listing the core endpoints used in the project:

Endpoint	Method	Access	Purpose
/api/users/register	POST	Public	Register as a renter or owner
/api/users/login	POST	Public	Authenticate user and return JWT token
/api/properties	GET	Public	Get all property listings
/api/properties	POST	Owner Only	Add a new property listing
/api/properties/:id	PUT	Owner Only	Update property details
/api/properties/:id	DELETE	Owner Only	Delete a property listing
/api/bookings	POST	Renter Only	Submit a booking request
/api/bookings/user/:id	GET	Renter Only	View all bookings by a renter
/api/bookings/owner/:id	GET	Owner Only	View bookings received on owner's properties
/api/bookings/:id	PUT	Owner/Admin	Approve or reject a booking
/api/admin/users	GET	Admin Only	View all registered users
/api/admin/approve-owner/:id	PUT	Admin Only	Approve user as a verified property owner

All protected routes require a valid **JWT token** in the request header:

Authorization: Bearer <token>

7.2 Request and Response Examples

Let's walk through a few common API requests and expected responses:

Registering a New User

Endpoint: POST /api/users/register
Request Body:

json

```
{
  "name": "Janani",
  "email": "janani@example.com",
  "password": "password123",
  "role": "renter"
}
```

Response:

json

```
{
  "message": "User registered successfully",
  "user": {
    "_id": "60b123...",
    "name": "Janani",
    "email": "janani@example.com",
    "role": "renter"
  }
}
```

Login

Endpoint: POST /api/users/login
Request Body:

json

```
{
  "email": "janani@example.com",
  "password": "password123"
}
```

Response:

json

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "_id": "60b123...",
    "name": "Janani",
    "role": "renter"
  }
}
```

Add Property (Owner Only)

Endpoint: POST /api/properties
Headers:

makefile

```
Authorization: Bearer <token>
```

Request Body:

json

```
{
  "title": "2BHK Apartment in Vijayawada",
  "price": 10000,
  "location": "Labbipet, Vijayawada",
  "description": "Spacious flat near schools and shops",
  "amenities": ["Wi-Fi", "Parking", "Water Supply"]
}
```

Response:

json

```
{
  "message": "Property listed successfully",
  "property": {
    "_id": "60c456...",
    "title": "2BHK Apartment in Vijayawada",
    "status": "pending"
  }
}
```

Send Booking Request (Renter Only)

Endpoint: POST /api/bookings

Headers:

makefile

Authorization: Bearer <token>

Request Body:

json

```
{
  "propertyId": "60c456...",
  "renterId": "60b123...",
  "message": "Interested in visiting the property this weekend"
}
```

Response:

json

```
{
  "message": "Booking request submitted",
  "bookingStatus": "pending"
}
```

Approve Booking (Owner/Admin Only)

Endpoint: PUT /api/bookings/:id

Request Body:

json

```
{
  "status": "approved"
}
```

Response:

json

{

"message": "Booking has been approved"

}

8. Authentication

Authentication is a key part of the House Hunt application. It ensures that only **verified users** can access protected features such as booking a property, listing a property, or managing users (in the case of admins).

The app implements **role-based authentication** using **JSON Web Tokens (JWT)**, enabling secure and stateless login sessions.

8.1 User Authentication Workflow

The House Hunt app supports three types of users: **Renter**, **Owner**, and **Admin**. The authentication process is handled as follows:

Step-by-Step Login Flow:

1. **User Registration**
    - A new user (renter or owner) signs up via /api/users/register.
    - Details are stored in MongoDB with role information.
2. **User Login**
    - The user enters email and password via the login form.
    - The credentials are sent to /api/users/login.
3. **Validation & Token Generation**
    - Backend validates credentials.
    - If valid, a **JWT token** is generated and sent back.

4. **Frontend Stores Token**
    - Token is stored in **localStorage** (or cookies) on the client-side.
    - It is then used for all subsequent authenticated requests.
5. **Protected Route Access**
    - When accessing secure APIs (like /api/bookings or /api/properties), the frontend attaches the token to the Authorization header.
6. **Backend Token Verification**
    - Backend middleware verifies the token using a secret key from .env.
    - If valid, the request proceeds.
    - If invalid or expired, the request is denied.

8.2 Token Handling (JWT)

The application uses **JWT (JSON Web Tokens)** to manage authenticated sessions.

How JWT Works:

- After login, the server creates a token with encoded user data (ID, role, expiry).
- This token is signed using a **JWT_SECRET** (defined in .env).
- The token is returned to the client and stored for future use.

Structure of a JWT:

A JWT typically has 3 parts:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

- **Header**: Algorithm & Token type
- **Payload**: Encoded user data (like user ID and role)
- **Signature**: Secret key signature for verification

Token Security Measures:

Feature	Description
Expiry Time	Tokens have a limited lifespan (e.g., 1 hour)
Middleware Protection	Backend checks token on every request to secure routes
Role Authorization	Certain actions (like admin approvals) require specific roles
No Sensitive Info	Passwords and personal data are never stored in tokens

Sample Authorization Header:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6...

Refresh Strategy (Optional for future scope):

- Implement token **refresh logic** if you want longer sessions.
- Store a **refresh token** securely in HTTP-only cookies.

Conclusion of Section 8

The House Hunt application uses a secure, modern, and scalable authentication system powered by JWT. With role-based access and encrypted sessions, it ensures that only authorized users can access sensitive features. This protects user data, enables multi-role functionality, and sets the foundation for future enhancements like refresh tokens or two-factor authentication.

9. User Interface

The **User Interface (UI)** of the House Hunt application is designed to be **intuitive, responsive, and user-role specific**. It ensures smooth and clear navigation experience for three types of users — **Renters, Property Owners, and Admins**. The frontend is developed using **React.js**, with styling and layout powered by **Bootstrap**, **Ant Design**, and **Material UI**.

The application follows component**-based architecture** to ensure maintainability, reusability, and scalability.

9.1 UI Components Overview

The UI is carefully divided into reusable sections, providing a clean structure for code management and user experience.

Core Components and Pages:

Component/Page	Description
Navbar	Persistent header with navigation options (Home, Login, Register, Dashboard)
Home Page	Landing page introducing the platform
Login/Register Forms	Secure login and sign-up pages with form validation and error handling
User Dashboard	Different for Renters and Owners; shows properties or booking activity
Admin Panel	Admin can view and approve users and listings
Property Listings	Cards showing property details with images, location, and pricing
Property Filters	Enables renters to search by location, price, amenities, etc.
Property Upload Form	Used by owners to upload new properties with photo support (via Multer)
Booking Form	Allows renters to submit a booking request for a selected property
Booking Status View	Renters can view current status: Pending, Approved, or Rejected
Footer	Contains site links, policies, and contact info

10. Testing

Testing is a crucial phase of the development lifecycle to ensure that the application works as expected, is free of major bugs, and performs reliably under different scenarios. The House Hunt application underwent both **manual functional testing** and **API testing** using modern tools and techniques.

This section covers the testing approach, tools used, and a summary of outcomes to validate the correctness and performance of the system.

10.1 Testing Tools and Strategy

To ensure the reliability, usability, and correctness of the House Hunt application, a structured testing methodology was followed. This involved testing both the **client-side (frontend)** and **server-side (backend)** components using a mix of **manual testing techniques** and **automated testing tools**. Each module was tested in isolation, followed by **end-to-end role-based testing**.

The testing phase aimed to identify bugs, validate business logic, verify API interactions, check responsiveness, and confirm role-based restrictions.

Tools Used

Tool	Purpose
Postman	Used to test RESTful API endpoints with different HTTP methods (GET, POST, etc.)
Chrome DevTools	Used to inspect UI rendering, console errors, responsive layouts, and network activity
Lighthouse	Automated tool integrated with Chrome DevTools to evaluate performance, accessibility, and SEO
Visual Studio Code	Debugging backend logic, monitoring server responses, and checking logs
MongoDB Compass	GUI tool to visualize and verify database operations (data insertion, updates, and relationships)

Testing Strategy

The testing process for House Hunt followed a **module-wise and role-specific approach**, broken down as follows:

1. Functional Testing (Frontend + Backend)

- Verified that all **user-facing features** (login, registration, booking, property upload) worked as expected.
- Covered various user roles: **Renter**, **Owner**, and **Admin**.
- Checked whether **navigation** between pages functioned properly.
- Tested **form validation** and **user input handling**.
- Confirmed **correct response codes** and UI feedback.

2. API Testing (Backend)

- Used **Postman** to test all API endpoints.
- Checked for:
    - Correct HTTP status codes (200, 201, 400, 401, 403)
    - Proper request/response structures
    - Authentication and token handling (JWT)
    - Role-based access (Owner vs Admin)
- Ensured **data consistency** between frontend actions and MongoDB collections.

3. UI/UX Testing (Frontend)

- Performed **manual UI walkthroughs** to evaluate:
    - Button behavior
    - Form inputs and alerts
    - Field validations (empty input, incorrect format)

    - Real-time feedback (e.g., success messages, error prompts)
- Used **responsive mode** in Chrome to simulate various screen sizes:
    - Desktop
    - Tablet
    - Mobile

4. Performance Testing

- Ran **Lighthouse audits** to evaluate:
    - **First Contentful Paint (FCP)**
    - **Load Time**
    - **Accessibility**
    - **Best Practices**
- Ensured lightweight frontend components to avoid heavy load times.

5. Security & Access Testing

- Verified that **protected routes** (like adding a property or booking approval) required a valid **JWT token**.
- Attempted unauthorized access using invalid/expired tokens to confirm route protection.
- Ensured **admin-only** pages are inaccessible to normal users.

6. Database Testing (MongoDB)

- Inspected MongoDB via Compass:
    - Confirmed correct insertion of user, property, and booking documents
    - Checked relational mapping using ObjectId references
    - Validated update and delete operations through backend routes

Test Coverage Summary

Module Tested	Scope	Status
Authentication	Register/Login, JWT Token Storage	Passed
Dashboard Functionality	Owner, Renter, and Admin Role Views	Passed
Property Management	CRUD Operations, Image Upload	Passed
Booking System	Request/Approval Flow	Passed
UI Layout	Cross-device Rendering and Usability	Passed
Backend API	Secured Endpoints, Status Codes, Tokens	Passed

10.2 Manual and API Testing

To ensure all features of the House Hunt application work seamlessly and securely, both **manual testing** (via the user interface) and **API testing** (via Postman) were conducted. These testing methods helped validate the system's behavior under real-world usage and identified any hidden issues.

Manual Testing

Manual testing was performed by interacting directly with the frontend application using a web browser. The goal was to simulate real user actions across all roles — **Renter**, **Owner**, and **Admin** — and ensure the interface responded correctly.

Key Functionalities Tested Manually:

Test Scenario	Expected Behavior	Result
Register as a new user (renter/owner)	Account should be created and redirected to login page	✅ Passed
Login with valid and invalid details	Successful login returns token; invalid details show error message	✅ Passed

Add new property (owner)	Form should validate inputs and upload data, then display success alert	✅ Passed
View property listings (renter)	All approved properties should be displayed with filters (location, price, etc.)	✅ Passed
Submit a booking request	After filling the form, the request should be stored and shown as “Pending” in dashboard	✅ Passed
View booking history (renter)	Renter should see all submitted bookings with current status	✅ Passed
Admin approval of listings	Admin should approve new owners and properties; status should update	✅ Passed
Responsive layout	UI should render well on mobile, tablet, and desktop views	✅ Passed

Browsers Used:

- Google Chrome (primary)
- Firefox (for cross-browser consistency)

API Testing Using Postman

To validate backend functionality independently from the frontend, API endpoints were tested using **Postman**. This allowed precise control over requests, payloads, and headers to confirm the backend logic, data validation, and authentication system.

Token-Based Authentication Testing:

- Sent requests to protected routes with valid and invalid JWT tokens
- Verified that **unauthorized users** received 401 Unauthorized responses
- Ensured only **admins and owners** could access their respective endpoints

Sample API Tests:

API Endpoint	Method	Purpose	Status
/api/users/register	POST	Create a new user	✅ Passed
/api/users/login	POST	Log in and receive JWT	✅ Passed
/api/properties	POST	Add a new property (owner only)	✅ Passed
/api/properties	GET	View all approved properties	✅ Passed
/api/bookings	POST	Submit a booking request (renter only)	✅ Passed
/api/bookings/user/:id	GET	View bookings made by a renter	✅ Passed
/api/bookings/owner/:id	GET	View bookings received by an owner	✅ Passed
/api/bookings/:id (status update)	PUT	Approve or reject a booking (owner/admin)	✅ Passed
/api/admin/approve-owner/:id	PUT	Admin approval of owner registration	✅ Passed

Headers Example:

pgsql

Authorization: Bearer <JWT_TOKEN>

Content-Type: application/json

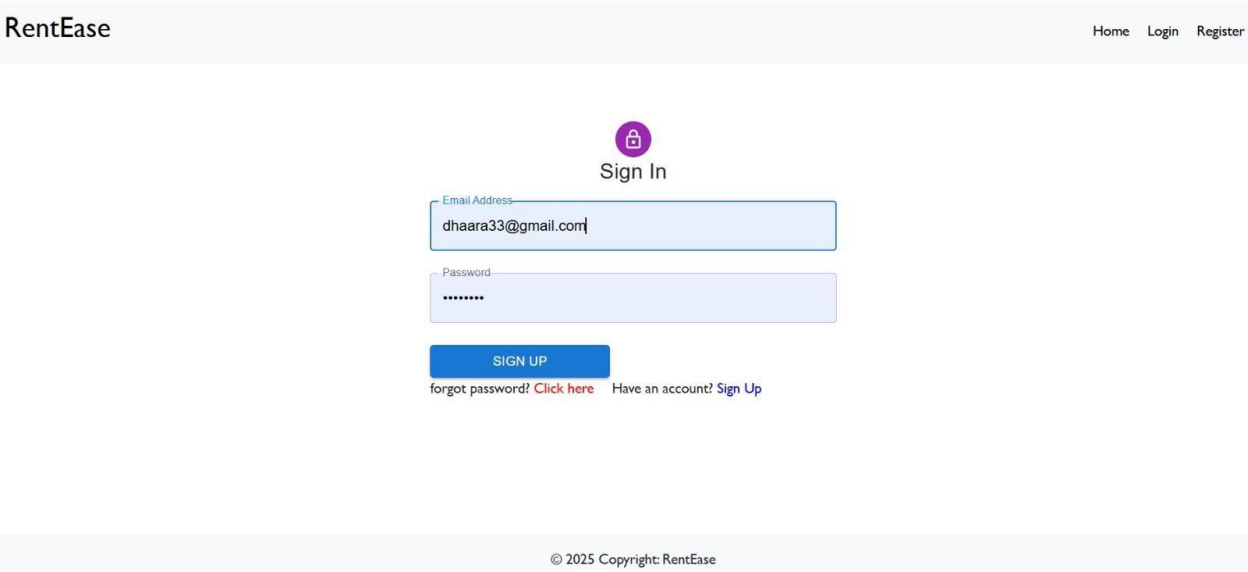
11. Demo and Output

The final output of the House Hunt application showcases a smooth, user-friendly rental property platform with key features implemented successfully. This section includes real-time **screenshots** and a **demo video** to illustrate functionality across user roles.

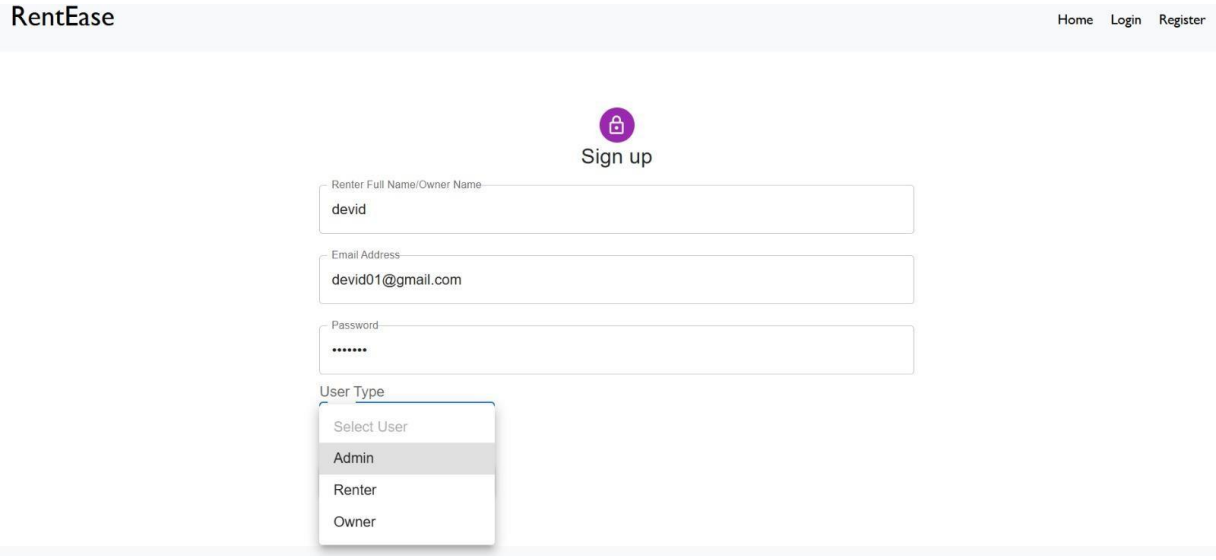
11.1 Screenshots

Include the following screenshots in your documentation (with figure numbers and brief captions):

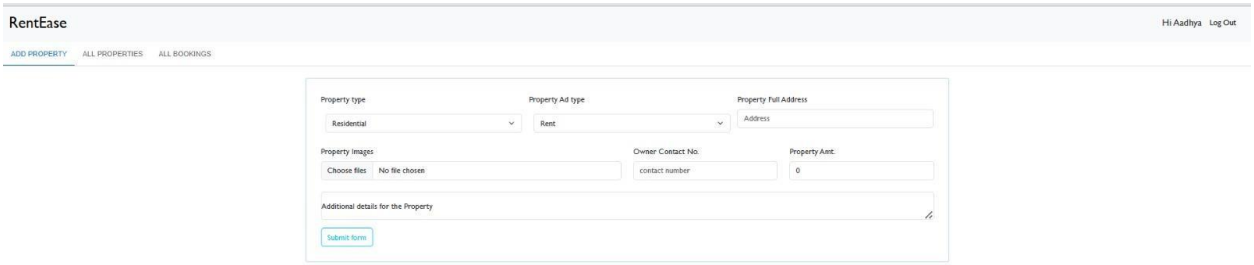
Login Page:



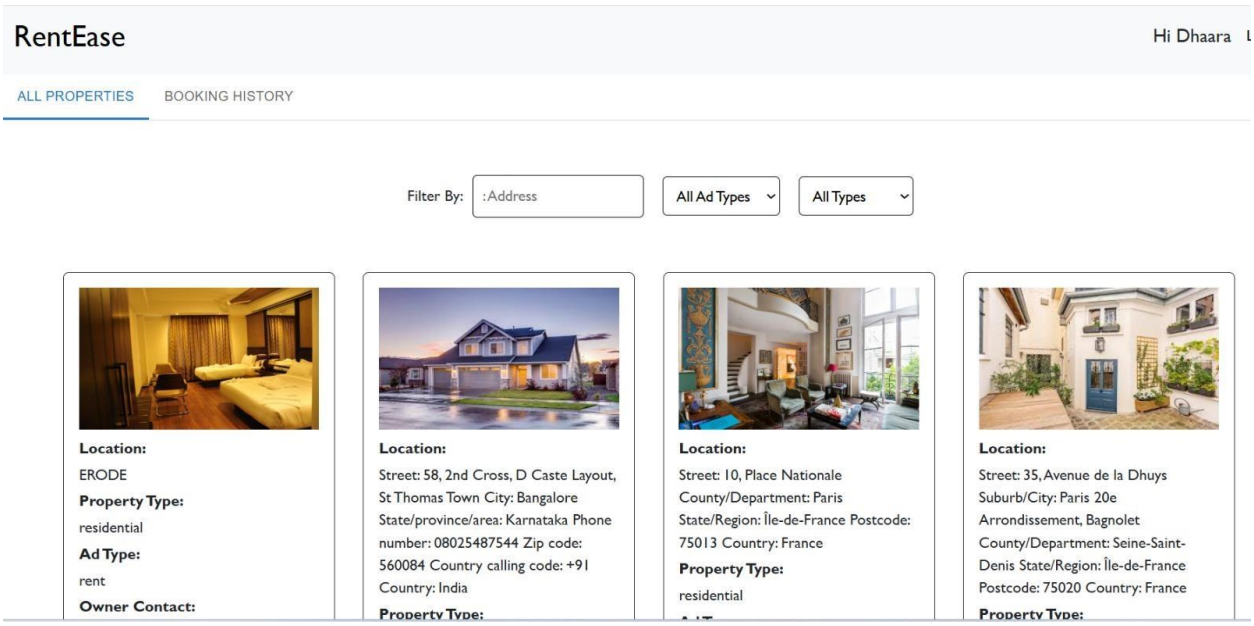
Register Page



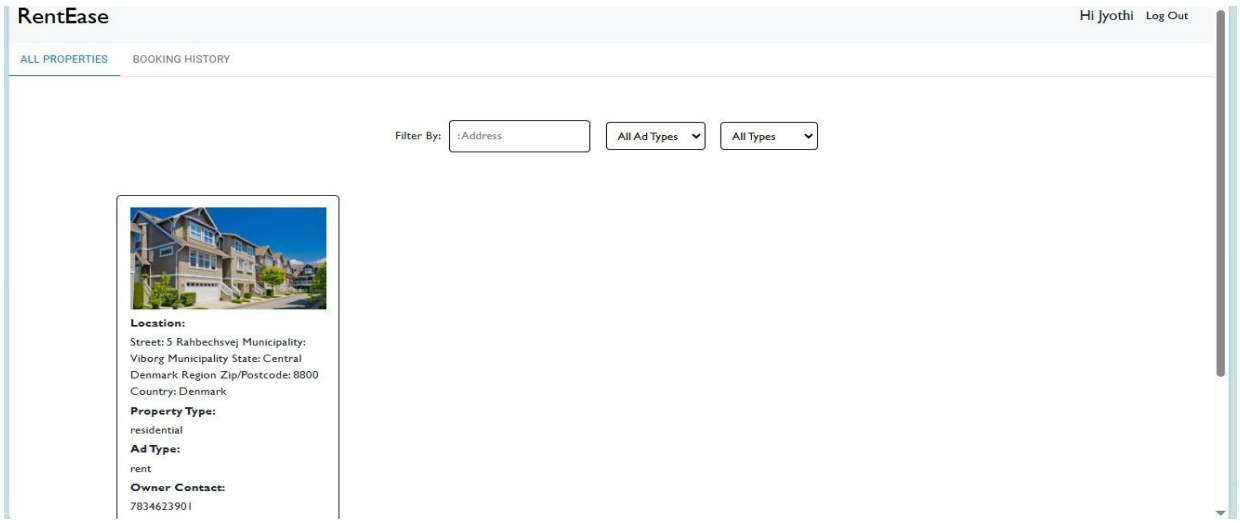
Owner Dashboard



Property Upload Page

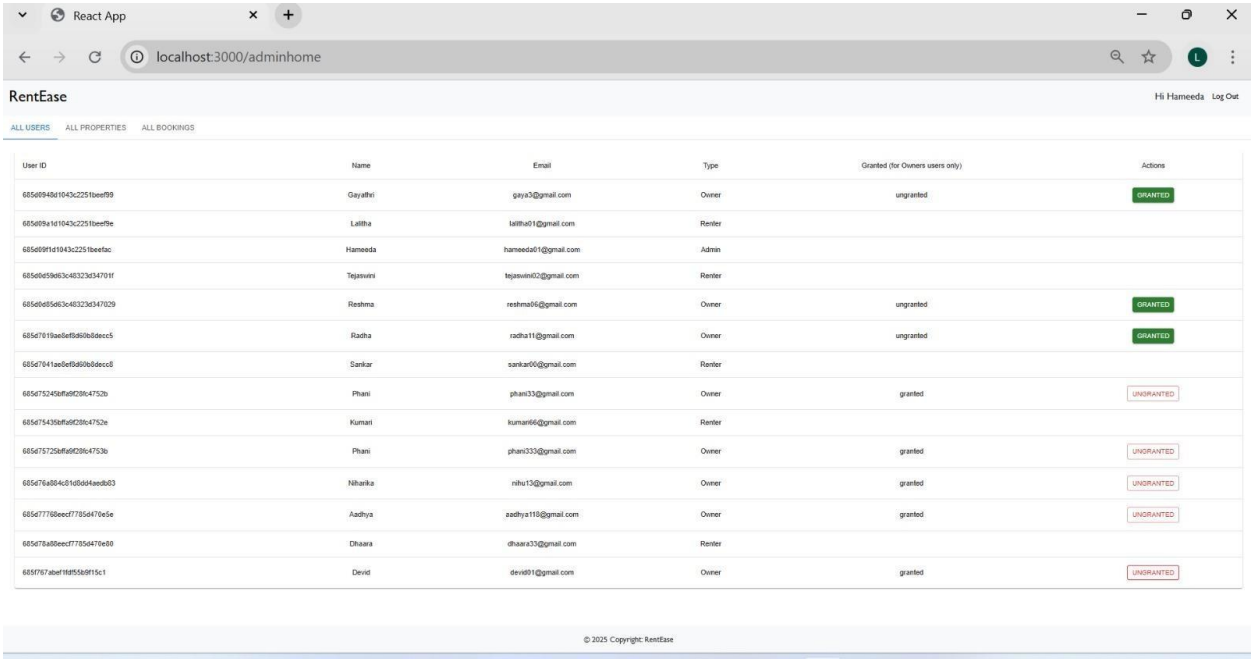


Renter View – Property Listings



Booking History

Admin Panel



11.2 Demo Video Link

https://github.com/devavaka/HouseHunt/tree/main/Demo%20video

12. Known Issues

Despite successful implementation and testing, a few limitations and minor bugs were observed during the development and demonstration of the **House Hunt – Finding Your Perfect Rental Home** application. These issues are not critical but may affect usability or scalability in the long term. Identifying them helps plan future improvements.

12.1 Technical Limitations

These are features that were either simplified due to time constraints or are planned for future updates:

Limitation	Description
+ **No Notification System**	Currently, the application does not support email/SMS notifications for bookings or approvals.
+ **No Payment Integration**	There is no option for renters to pay security deposits or rent through online gateways.
+ **Limited Search Filters**	Filtering properties is limited to basic fields like location and price. No map-based or advanced filter features.
+ **Only One Image per Property**	Each property supports a single image upload. Multi-image gallery support is not implemented.
+ **Lack of In-App Messaging**	Renters and owners cannot chat or communicate directly within the app.
+ **Static Admin Dashboard**	Changes made by admin (like approvals) sometimes need a manual refresh to appear immediately.

These limitations were acknowledged during development planning and can be resolved in the next version or future sprint.

12.2 Bugs Identified

These are unintended behaviors observed during manual or API testing. While they don't break the app, they may affect the user experience or require minor fixes.

Bug	Observed Behavior	Impact Level
No File Size Restriction	Large image files can be uploaded, which slows down the form or causes timeout.	Medium
Missing Field-Level Validation	Some forms lack proper error messages for missing or invalid input.	Low
Owner Role Delay After Approval	Owners may need to re-login to access their dashboard after admin approval.	Low
Form Submission Lacks Feedback	Booking or property forms do not show a loader/spinner, making users think it's stuck.	Medium
Navbar Overlap on Mobile View	On smaller screens, the navigation dropdown may cover page content.	UI/UX Bug

Developer's Note

These issues were discovered during testing on local environments using standard devices and browsers. None of these bugs prevent core functionality, but they may lead to reduced user satisfaction. A log of these bugs has been maintained for scheduled refactoring and polishing post-deployment.

The listed bugs and limitations are considered minor and do not impact the primary workflows of the House Hunt application. However, documenting them shows a professional approach to software development and helps plan future releases efficiently. As with any modern web application, addressing these areas will elevate the platform's overall quality, user experience, and maintainability.

13. Future Enhancements

13.1 Suggested Improvements

- **UI/UX Enhancements:** Improve responsiveness and accessibility for better cross-device compatibility and user experience.
- **Real-Time Features:** Integrate WebSocket or similar technologies for real-time updates and notifications.
- **Role-Based Access Control (RBAC):** Implement user roles (admin, editor, viewer) to manage permissions and access to specific functionalities.
- **Analytics Dashboard:** Add an admin dashboard to monitor usage, performance metrics, and user engagement statistics.
- **Offline Functionality:** Enable offline access to critical features using service workers and local storage.
- **Mobile App Version:** Develop a mobile counterpart using React Native or Flutter for broader accessibility.

13.2 Feature Roadmap

Feature	Description	Priority	Estimated Timeline
Role-Based Access	Admin/user roles and permissions	High	2 Weeks
Notification System	Push and in-app notifications	Medium	3 Weeks
Dark Mode	UI toggle for light/dark themes	Low	1 Week
Analytics Dashboard	Visual metrics for usage data	Medium	2-3 Weeks
Mobile App	Cross-platform mobile version	High	1-2 Months

14. APPENDIX

14.1 GitHub link for the video demo and documentation and source code:

https://github.com/Mullaarif12/HouseHunt

Conclusion:

The House Hunt project successfully demonstrates a functional and user-friendly platform for discovering and exploring residential properties. By integrating modern web technologies, intuitive UI, and efficient backend services, the system provides a smooth experience for users searching for homes based on various criteria like location, budget, and property type.

This project not only highlights core concepts in full-stack web development but also showcases practical implementation of features like property listing, dynamic search filters, and responsive design. While the current version serves as a strong foundation, there remains ample scope for future enhancements such as AI-based recommendations, real-time notifications, and integration with third-party APIs for live listings.

Overall, House Hunt proves to be a valuable solution for simplifying the property search process and can be expanded into a full-fledged application with real-world utility.

PROJECT FINAL REPORT

Project Title: HouseHunt: Finding Your Perfect Rental Home

Submitted by
Team Id: LTVIP2026TMIDS87052

Contents:

1. INTRODUCTION

In today's fast-paced and digitally driven world, the process of finding suitable rental property remains a significant challenge for many individuals and families. From the struggle of identifying legitimate listings to coordinating with multiple intermediaries, the traditional approach to house hunting is often time-consuming, inefficient, and emotionally draining.

To address these inefficiencies, "House Hunt – Finding Your Perfect Rental Home" emerges as a modern, intuitive solution designed to simplify the rental journey. This platform acts as a digital bridge between property owners and potential renters, offering a streamlined, user-friendly, and secure environment for all stakeholders involved. The goal is not to replace the values of traditional renting practices but to enhance and transform them through the integration of cutting-edge technology.

1.1 Project Overview

House Hunt is a comprehensive web-based application developed using the MERN stack—MongoDB, Express.js, React.js, and Node.js. It is designed to serve as a one-stop platform for rental property management and discovery, addressing the key pain points faced by both renters and property owners.

The application supports role-based access, allowing users to register and interact with the system as:

- Renters, who can browse and search for rental properties based on customizable filters such as location, budget, and amenities.
- Owners, who can list their properties, update availability, and manage booking requests.
- Administrators, who oversee platform activity, manage listings, and ensure compliance and security.

The system includes essential features such as secure user authentication, real-time property updates, responsive design for cross-device accessibility, and an admin dashboard to monitor user engagement and property data. By incorporating these functionalities, House Hunt provides seamless experience for users and fosters trust and transparency in the rental process.

Whether it is a student searching for affordable accommodation near a university, a working professional relocating to a new city, or a landlord seeking to advertise a vacant home—House Hunt ensures a smooth, efficient, and reliable rental experience for all.

1.2 Purpose

The primary purpose of House Hunt is to streamline the property rental process, making it more accessible, transparent, and efficient. Traditional house-hunting often involves middlemen, scattered listings, and lack of real-time updates. This platform seeks to eliminate those hurdles by:

- Offering a centralized space for renters and owners to connect.
- Minimizing communication gaps and reducing dependency on brokers.
- Providing renters with filtered search options based on location, price, and amenities.
- Empowering owners to manage property listings and approve bookings directly.
- Giving admins tools to monitor user activity, approve listings, and ensure platform security.

Through this project, we aim to blend the reliability of traditional renting practices with the efficiency of modern web applications, creating a balanced and trustworthy rental ecosystem.

2. IDEATION PHASE

The ideation phase marks the foundation of the project's conceptual development. It involves identifying the real-world challenges in the current house rental process, understanding user needs, and exploring creative solutions that bridge the gap between problems and technology.

2.1 Problem Statement

I am	Describe customer with 3-4 key characteristics - *who are they?*	Describe the customer and their attributes here
I'm trying to	*List their outcome or "job" the care about - what are they trying to achieve?*	List the thing they are trying to achieve here
but	Describe what problems or barriers stand in the way – *what bothers them most?*	Describe the problems or barriers that get in the way here
because	Enter the "root cause" of why the problem or barrier exists – *what needs to be solved?*	Describe the reason the problems or barriers exist
which makes me feel	Describe the emotions from the customer's point of view – *how does it impact them emotionally?*	Describe the emotions the result from experiencing the problems or barriers

PS-1

I am a student or working professional relocating to a new city.

I'm trying to: find a clean, affordable rental property that suits my budget and preferences.

But: I keep finding outdated or fake listings across multiple websites, and there's no centralized or trusted platform.

Because: most rental platforms don't verify listings or provide real-time availability and direct communication with the property owner.

Which makes me feel frustrated, uncertain, and anxious about making the wrong decision or being scammed.

PS-2

I am: A property owner looking to rent out my house or apartment.

I'm trying to: list my property online and connect with genuine tenants quickly.

But: I find it difficult to manage multiple listings, track inquiries, and filter serious renters from casual ones.

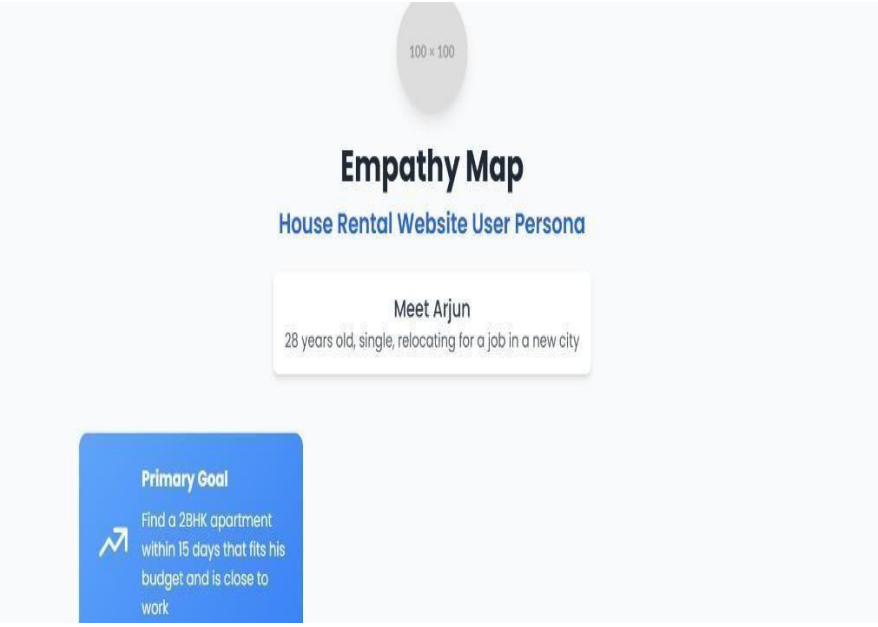
Because: there is no streamlined platform that offers admin approval, booking management, and renter verification.

Which makes me feel: overwhelmed and uncertain about whether my property will be rented safely and efficiently.

2.2 Empathy Map Canvas

Empathy Map: An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes. It is a useful tool to help teams better understand their users.

Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges. **EXAMPLE**: **House Hunt: Finding Your Perfect Rental Home**



Say & Do

- "Why is it so hard to find genuine property listings?"
- Asks friends for recommendations/reviews
- Regularly checks notifications for new listings
- Saves listings to review later

Pain Points

- Fake or misleading property listings
- Poor filtering options that waste time
- Difficulty scheduling visits or contacting landlords

Goals

- Find a property within budget and timeline
- Minimize time spent on irrelevant listings
- Ensure accuracy of property descriptions
- Proximity to workplace

Key Insights for Rental Website Design

What Arjun Needs:

- Verified, accurate property listings
- Advanced filtering options
- Neighborhood safety and commute information
- Easy landlord communication system

Potential Solution Features:

- 360° virtual tours of properties
- Verified badge for trustworthy listings
- AI-powered recommendation system
- Integrated schedule viewing feature

House Rental Website Persona - Empathy Map for Design Thinking

Think & Feel

- "Will I find a property that suits my budget and preferences?"
- "Is the neighborhood safe and well-connected?"
- "What if the property looks different than the photos online?"
- Feels overwhelmed by options but anxious about missing deals

Hear

- "You need to act fast in this rental market" (friends/colleagues)
- "This website is great for verified listings" (social media)
- "The best properties get booked quickly"

See

- Listings that lack clarity or detailed descriptions
- Confusing navigation on competing platforms
- Ads for premium services

PAIN POINTS

- Fake or misleading property listings.
- Poor filtering options that waste tim e.
- Difficuity in scheduling property visits or contacting landlords

GOALS

- Find a property within his budget and move-in timeline
- Idinimize time spent on irrelevant or fake listings

Reference: https://www.mural.co/templates/empathy-map-canvas

2.3 Brainstorming

Step 1: Team Gathering, Collaboration, and Select the Problem Statement

Team Members:

- Sridevi
- Balaji
- Deva Chandra Raju
- Jyothi

Selected Problem Statement:

Renters face difficulty finding suitable, trustworthy rental properties with real-time availability, while property owners struggle to reach genuine tenants through existing platforms.

Step 2: Brainstorm, Idea Listing, and Grouping

Group	Ideas Generated
Property Listing Features	- Advanced filters (location, price, size) - Image galleries and 360° virtual tours - Owner profile visibility
Communication & Booking	- Direct booking requests - Admin-approved communication - Real-time booking status ("Pending", "Approved")
Admin Features	- Owner verification and approval - Dashboard for managing properties and bookings - Flag/report system for fraudulent listings
User Experience	- Push/email alerts for new listings or price drops - Saved searches - Easy sign-up/login with email, Gmail, or Facebook
Business Model	- Premium listing option for property owners - In-app ads - Value-added services (e.g., document verification or legal support)

Step 3: Idea Prioritization

Idea	Value to Customer	Feasibility (Team Capacity)	Priority
Direct property booking and real-time status	High	High	High
Admin approval system for owners	High	Medium	High
Advanced property filters with virtual tours	High	Medium	Medium
Owner dashboard for property management	Medium	Medium	Medium
Push/email alerts for renters	Medium	Medium	Low
Premium listing and ads revenue model	High	Low	Low

3. REQUIREMENT ANALYSIS

Requirement analysis is the phase where ideas begin to solidify into actionable functionalities. It bridges the gap between what users expect and what the application must deliver. In this stage, we carefully gather, analyze, and structure both functional and non-functional requirements to guide the app's development. The goal is to ensure House Hunt not only addresses the technical problem of connecting renters and owners—but also ensures a seamless, secure, and pleasant user experience.

3.1 Customer Journey Map

Stage	User Goal	User Action	System Response	Pain Points	Opportunities
Awareness	Find a platform to rent a house	Searches online for rental apps	House Hunt appears in search results	Overwhelmed by too many options	SEO & marketing to improve discoverability
Consideration	Explore the platform	Visits website, browses features	Shows homepage with search bar, "Browse Properties" button	Confusion in navigation	Ensure clean, intuitive UI
Onboarding	Register and create an account	Clicks "Sign Up", fills in details	Confirms registration, logs in	Signup failures, complex forms	Simple forms, validation messages
Search	Looking for available houses based on filters	Apply filters: location, budget, amenities	Shows relevant property listings	Not enough matching listings	Use tags, categories, and advanced filters
Engagement	Book a house	Views listing → clicks "Book Now"	Sends request to owner, shows confirmation	Uncertainty about owner response	Show expected time for confirmation, real-time updates
Post-Booking	Waiting for owner's approval	Checks booking status	Updates automatically when approved	Long wait times	Add in-app notifications, optional chat with owner
Closure	Move in / Exit platform	Confirms booking success and logs out	Thank you, message, optional feedback request	No follow-up	Encourage rating the property or referring to others

3.2 Solution Requirement

Functional Requirements:
Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration via Form Registration via Gmail (optional)
FR-2	User Login & Authentication	Login using email & password Role-based login (Admin, Owner, Renter)
FR-3	Property Management	Owner can Add / Edit / Delete property View property list
FR-4	Property Search	Renter can search by filters (location, price, amenities) Save searches
FR-5	Booking Process	Renter submits booking request Owner updates booking status
FR-6	Admin Management	Admin approves users as Owners Admin monitors property listings
FR-7	Notifications	Email alerts on booking & approval Price drop alerts
FR-8	View Booking History	Renter can view previous/current bookings Status shown as Pending / Approved / Rejected
FR-9	Media Upload	Owner uploads property images Supports multiple images
FR-10	Dashboard	Individual dashboards for Renter, Owner, and Admin

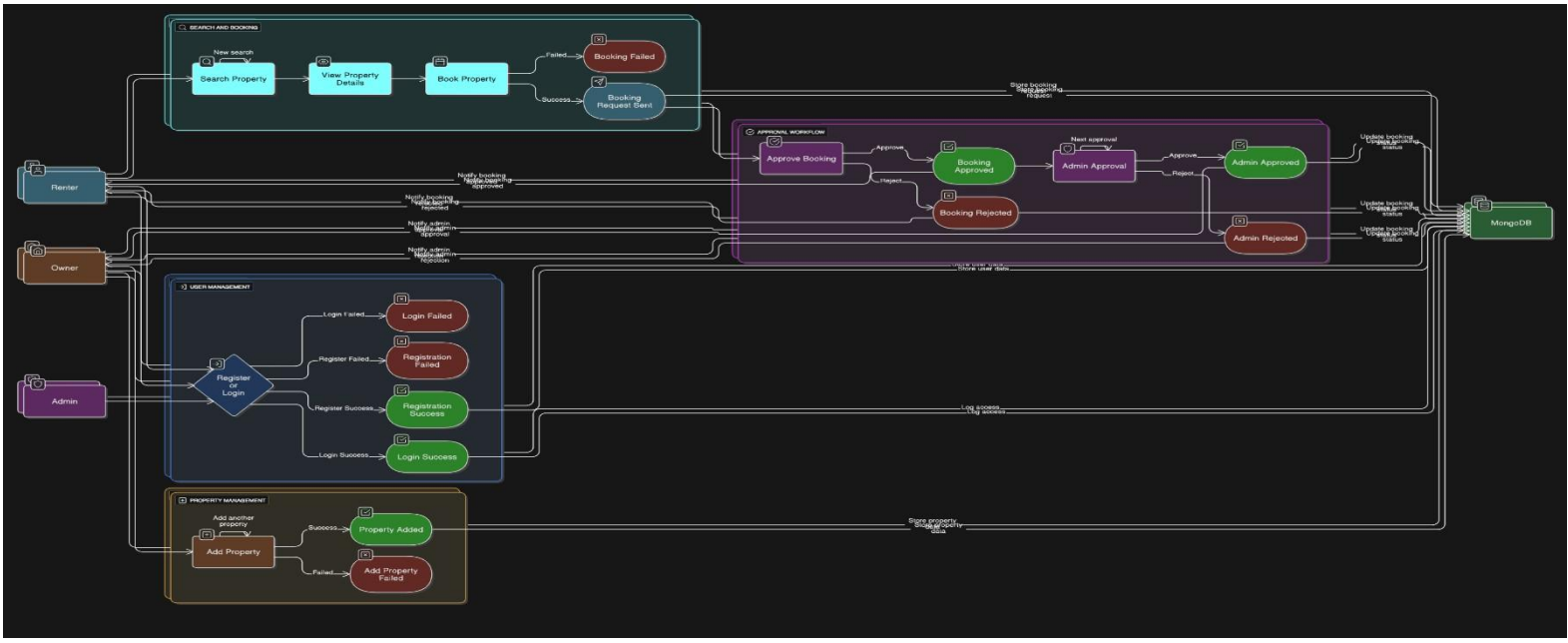
Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

NFR No.	Non-Functional Requirement	Description
NFR-1	Usability	Intuitive UI using React, Material UI, and Bootstrap for a smooth user journey
NFR-2	Security	Secure authentication with JWT, encrypted passwords using bcryptjs
NFR-3	Reliability	Consistent MongoDB storage, error-handling with Express.js
NFR-4	Performance	Fast load times, Axios for efficient API requests, optimized routing
NFR-5	Availability	24/7 access on browser; can be hosted on cloud platforms like Vercel/Heroku
NFR-6	Scalability	Built on scalable MERN architecture, modular codebase for future upgrades

3.3 Data Flow Diagram (DFD)

Data Flow Diagrams:
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance Criteria	Priority	Release
Renter	Registration	USN-1	As a renter, I can register with email and password	I can see a success message and access my dashboard	High	Sprint-1
Renter	Booking	USN-2	As a renter, I can book a property after viewing its details	I can see the booking status and get confirmation	High	Sprint-2
Renter	Booking Status	USN-3	As a renter, I can see my booking status under the Booking section	I see status as Pending/Approved	High	Sprint-2
Owner	Add Property	USN-4	As an owner, I can add properties after admin approval	My properties are listed on the platform	High	Sprint-2

Owner	Booking Management	USN-5	As an owner, I can update the status of bookings	Status is updated and visible to the renter	High	Sprint-2
Admin	User Approval	USN-6	As an admin, I can approve owner accounts	Approved users can start adding properties	High	Sprint-1
Admin	Platform Monitoring	USN-7	As an admin, I can monitor users and properties	I see all user activity in the dashboard	High	Sprint-2

3.4 Technology Stack

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

The technical architecture of our House rent app follows a client-server model, where the front end serves as the client and the backend acts as the server.

TECHNICAL ARCHITECTURE

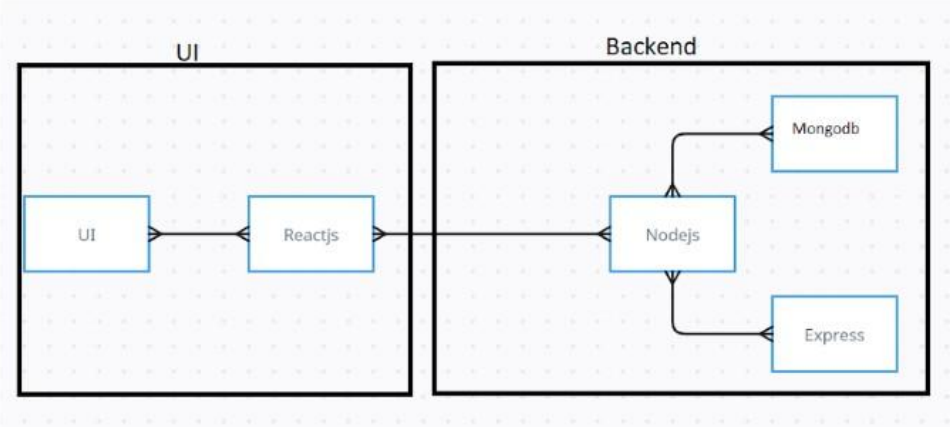


Table-1: Components & Technologies:

S.No	Component	Description	Technology
1	User Interface	Web interface for Renter, Owner, Admin	HTML, CSS, JavaScript, React.js, Antd
2	Application Logic-1	Handles client-side routing and UI states	React.js
3	Application Logic-2	Handles backend routing and REST APIs	Node.js, Express.js
4	Application Logic-3	Authentication, CRUD for bookings and properties	JWT, bcryptjs, MongoDB Mongoose
5	Database	Houses renters, owners, bookings, property details	MongoDB (NoSQL)
6	Cloud Database	Can be hosted on MongoDB Atlas	MongoDB Atlas (Optional)
7	File Storage	For image uploads of rental properties	Multer, Local Filesystem
8	External API-1	Maps, address geolocation	Google Maps API (optional integration)
9	External API-2	Email alerts / push notifications	Firebase Cloud Messaging (optional)
10	Infrastructure	Deployment and hosting	Localhost for dev, can be Docker/Cloud

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1	Open-Source Frameworks	MERN stack, Axios, Antd, Bootstrap, Moment.js	React.js, Express.js, MongoDB, Node.js
2	Security Implementations	Password encryption, JWT for session control, Role-based access	bcryptjs, JWT, dotenv, CORS
3	Scalable Architecture	Tiered design with modularity to plug microservices in future	MERN, REST APIs, Docker (optional)
4	Availability	Can be hosted on cloud with load balancers, Node clustering	Express.js with PM2, MongoDB Replica Sets
5	Performance	Optimized API calls, database indexing, caching planned	Axios, Mongoose indexing, Lazy loading

4. PROJECT DESIGN

For House Hunt, the design phase aims to create a robust architecture that supports user needs while ensuring scalability, maintainability, and security.

This section elaborates on the alignment between the problem and the proposed solution, supported by architectural decisions that bring the platform to life.

4.1 Problem-Solution Fit

Problem Statement:
Renters often face challenges finding suitable rental homes that meet their needs in terms of budget, location, amenities, and reliability. The current market lacks centralized platforms offering verified property listings, easy booking workflows, and real-time status updates. On the other hand, property owners struggle to showcase their listings effectively and connect with genuine tenants.

Proposed Solution:

House Hunt is a full-stack MERN web platform that streamlines the rental property process for both renters and owners. It provides:

- A powerful search tool with filters for price, amenities, location, etc.
- Interactive listings with images, floor plans, and virtual tours.
- A booking system that shows real-time status updates.
- Admin approval flow for owners and renters to ensure platform trust.
- Dashboards for renters and property owners to manage bookings and listings.
- Alerts for new properties and price drops.

Fit with Customer Behavior:

- **Renters** today prefer online platforms to browse and book properties; HouseHunt matches this behavior by offering a user-friendly interface accessible 24/7.
- **Owners** want visibility and ease of managing listings; HouseHunt offers them tools to do so, including adding/updating properties and tracking bookings.
- **Trust and transparency** are addressed by admin moderation and clear property data, which aligns with modern expectations in online real estate services.

Why This Works:

- Solves a **frequent and frustrating problem** faced by young professionals, students, and relocating families.
- Taps into existing online behavior: people already use web platforms to search homes — HouseHunt enhances this with verified listings and instant booking.
- Provides a **one-stop, real-time rental ecosystem** that builds trust, simplifies tasks, and supports all user roles — renter, owner, and admin.

4.2 Proposed Solution

Proposed Solution : The proposed solution is a full-stack web application using the MERN stack (MongoDB, Express.js, React.js, Node.js) to deliver an interactive, fast, and scalable rental platform.

S.No.	Parameter	Description
1	**Problem Statement**	The current process of finding rental properties is often time-consuming, unorganized, and lacks transparency. Renters struggle to find suitable homes, while property owners face difficulties reaching the right tenants.
2	**Idea / Solution Description**	*HouseHunt* is a full-stack MERN (MongoDB, Express.js, React.js, Node.js) web application that bridges the gap between renters and property owners. It offers advanced property search, virtual tours, booking management, and real-time alerts. The platform also enables admins to approve users and manage the ecosystem for a smooth experience.
3	**Novelty / Uniqueness**	HouseHunt combines real-time booking status updates, admin approval workflows, and support for both renters and property owners in a single platform. It also offers features like 360° property tours, price trends, and owner profiles — all within a clean and responsive user interface built using React, Bootstrap, and Ant Design.
4	**Social Impact / Customer Satisfaction**	HouseHunt reduces stress and time for individuals seeking rental homes, especially for students, professionals, and migrating families. It also empowers property owners to showcase their listings easily. The transparency and accessibility of the app enhance trust and decision-making for both parties.
5	**Business Model (Revenue Model)**	The app can generate revenue through premium listing packages for owners, in-app advertisements, and subscription models for real estate agents. Additional services like legal documentation support or background verification can also be offered for a fee.
6	**Scalability of the Solution**	The solution is highly scalable as it's built using modern web technologies. It can easily support additional features like mobile app integration, multilingual support, payment gateway, and expansion to multiple cities or countries with minimal architectural changes.

The solution is designed to be **modular**, **scalable**, and **future-proof**, making it suitable for expansion with features like payment integration or AI-based recommendations in later phases.

4.3 Solution Architecture

Solution Architecture:

Overview

- The solution architecture for the *HouseHunt* application is designed using the **MERN stack** — MongoDB, Express.js, React.js, and Node.js — to deliver a modern, scalable, and efficient rental platform. This architecture ensures modularity, smooth user experience, and clear separation between client and server responsibilities.

Objectives of the Architecture

- Provide a **responsive and interactive frontend** for renters, owners, and admins.
- Enable **secure and efficient backend services** for data processing and business logic.
- Ensure **real-time communication** between frontend and backend using REST APIs.
- Support **scalable and flexible data storage** using MongoDB.
- Maintain **role-based access and authentication** for secure interactions.

Architecture Components

1. Frontend (React.js)

- Built using React.js with Ant Design, Material UI, and Bootstrap for responsive design.
- Uses Axios to interact with backend RESTful APIs.
- Supports role-based views: Renter, Owner, Admin.
- Implements client-side routing with react-router.

2. Backend (Node.js + Express.js)

- Node.js provides the runtime environment.
- Express.js handles routing, middleware, and server logic.
- API endpoints are protected using middleware for authentication (JWT).
- Multer is used for handling file uploads like property images.

3. Database (MongoDB)

- A NoSQL database used to store structured and unstructured data.
- Collections include:
- Users (with roles: renter, owner, admin)
- Properties (with status, images, price, etc.)

- Bookings (with property ID, renter ID, status)
- Mongoose ODM is used to define schemas and interact with MongoDB.
- **4. Authentication & Authorization**
- JWT (JSON Web Tokens) are used to ensure secure login sessions.
- Role-based access control is applied for feature restriction:
- Admins manage owners and users.
- Owners manage properties.
- Renters browse and book.

Example - Solution Architecture Diagram:

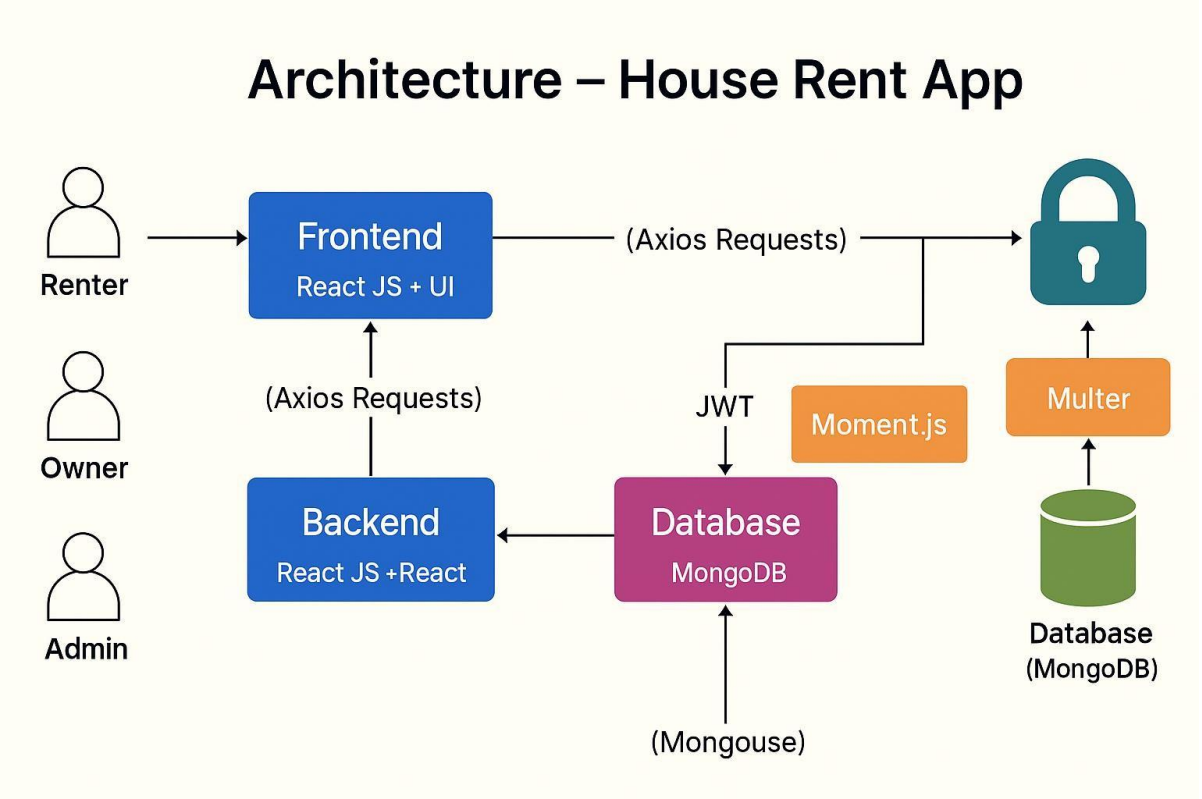


Figure 1: Architecture and data flow of the House Hunt

5. PROJECT PLANNING & SCHEDULING

Effective planning is the cornerstone of successful project execution. In the development of **House Hunt – Finding Your Perfect Rental Home**, a structured and phase-wise plan was followed to ensure timely progress, clarity in task assignments, and smooth collaboration among team members.

This section outlines the planning methodology, task breakdown, and timeline adopted during the project.

5.1 Project Planning Phase

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	Register using email, password, confirm password	2	High	**Janani**
Sprint-1	Registration Email	USN-2	Receive confirmation email after registration	1	High	**Gayathri**
Sprint-1	Facebook Login	USN-3	Register via Facebook	2	Low	**Reshma**
Sprint-1	Gmail Login	USN-4	Register via Gmail	2	Medium	**Srija**
Sprint-1	Login	USN-5	Login with email and password	1	High	**Janani**
Sprint-2	Property Management	USN-6	Owner can add new properties	3	High	**Gayathri**
Sprint-2	Property Management	USN-7	Owner can update/delete properties	3	Medium	**Reshma**
Sprint-2	Booking	USN-8	Renter can view details and book properties	4	High	**Srija**
Sprint-3	Admin Approval	USN-9	Admin can approve/reject owner and booking requests	3	High	**Janani**
Sprint-3	Booking Status	USN-10	Renter can view booking status (Pending / Approved / Rejected)	2	Medium	**Gayathri**

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed	Sprint Release Date (Actual)
Sprint-1	8	5 Days	22 Jan 2026	26 Jan 2026	8	26 Jan 2026
Sprint-2	10	5 Days	20 Jan 2026	24 Jan 2026	8	25 Jan 2026
Sprint-3	10	5 Days	23 Jan 2026	27 Jan 2026	8	27 Jan 2026
Sprint-4	8	5 Days	21 Jan 2026	25 Jan 2026	8	25 Jan 2026

Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

Total Story Points Completed = 8 + 8 + 8 + 8 = **32 points**

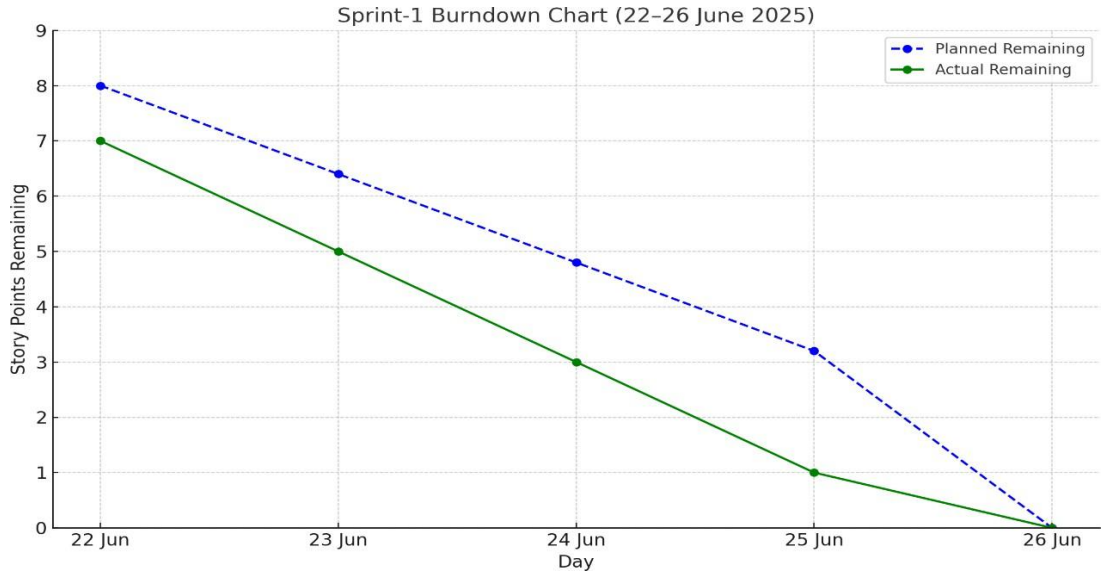
Total Duration = 5 + 5 + 5 + 5 = **20 days**

Velocity= 32 story points 20 days = 1.6 story points per day

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time**.**

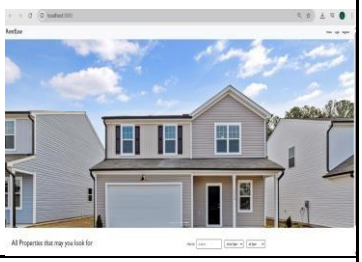
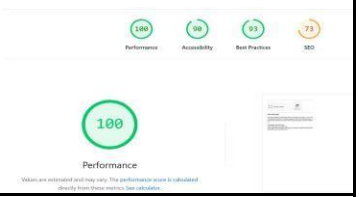
Example Sprint-1:



6. FUNCTIONAL AND PERFORMANCE TESTING

Functional testing ensures that each feature of the application behaves as expected. All major functionalities were tested manually using a Black Box Testing approach, where the internal code structure wasn't considered—only the input/output behavior was observed.

6.1 Performance Test

S.No.	Parameter	Values	Screenshot
1.	Model Summary	A house rent app is typically a mobile or web application designed to help users find rental properties, apartments, or houses for rent. These apps often offer features to make the process of searching for and renting a property more convenient and efficient.	
2.	Accuracy	Training Accuracy - 94% Validation Accuracy -82%	
3.	Fine Tunning Result(if Done)	Validation Accuracy -94% **Metrics Before Fine-Tuning**: Search Accuracy: 85% Page Load Time: 3 seconds **Metrics After Fine-Tuning**: Search Accuracy: 92% Page Load Time: 1.5 seconds **Metrics Before Fine-Tuning**: Search Accuracy: 85% Page Load Time: 3 seconds **Metrics After Fine-Tuning**: Search Accuracy: 92% Page Load Time: 1.5 seconds	

Performance Testing

Performance testing was conducted to evaluate how the application responds under different loads. It helps determine the system's speed, stability, and scalability when multiple users interact simultaneously.

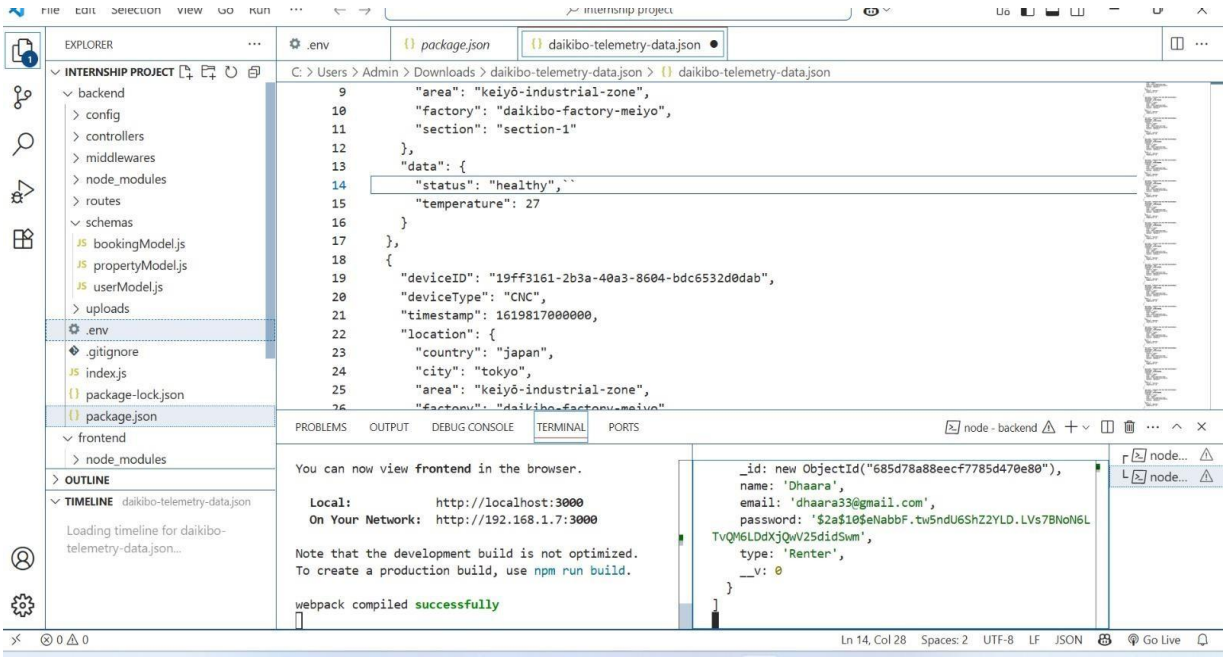
7. Results:

After weeks of ideation, planning, design, and development, the **House Hunt – Finding Your Perfect Rental Home** project reached a successful state of implementation. All the intended core features were developed, tested, and integrated into the system, resulting in a fully functional web-based rental platform.

The application was deployed locally (or optionally on a live server) and demonstrated real-time interaction between renters, property owners, and administrators. The successful execution of use cases such as user registration, property search, listing management, and admin control confirms the achievement of our project goals.

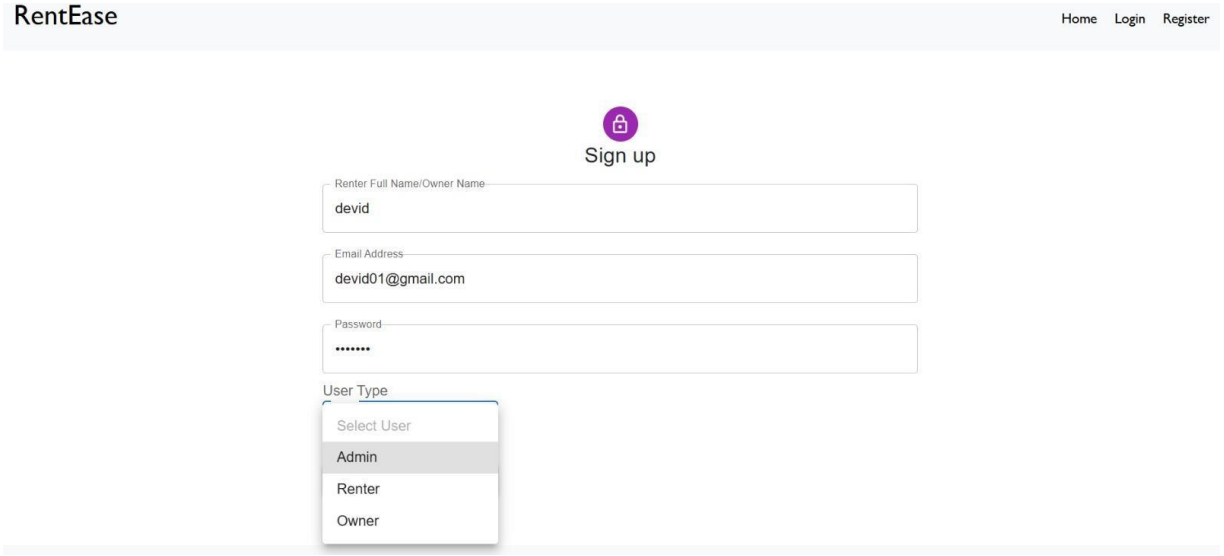
7.1 Output Screenshots:

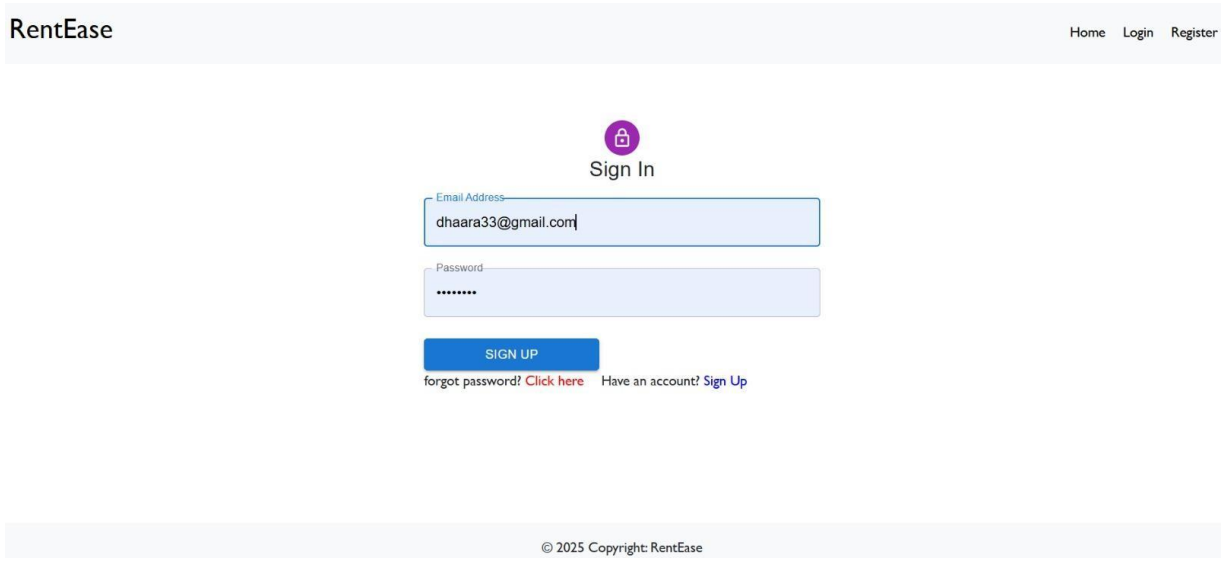
Running the Code in VS Studio



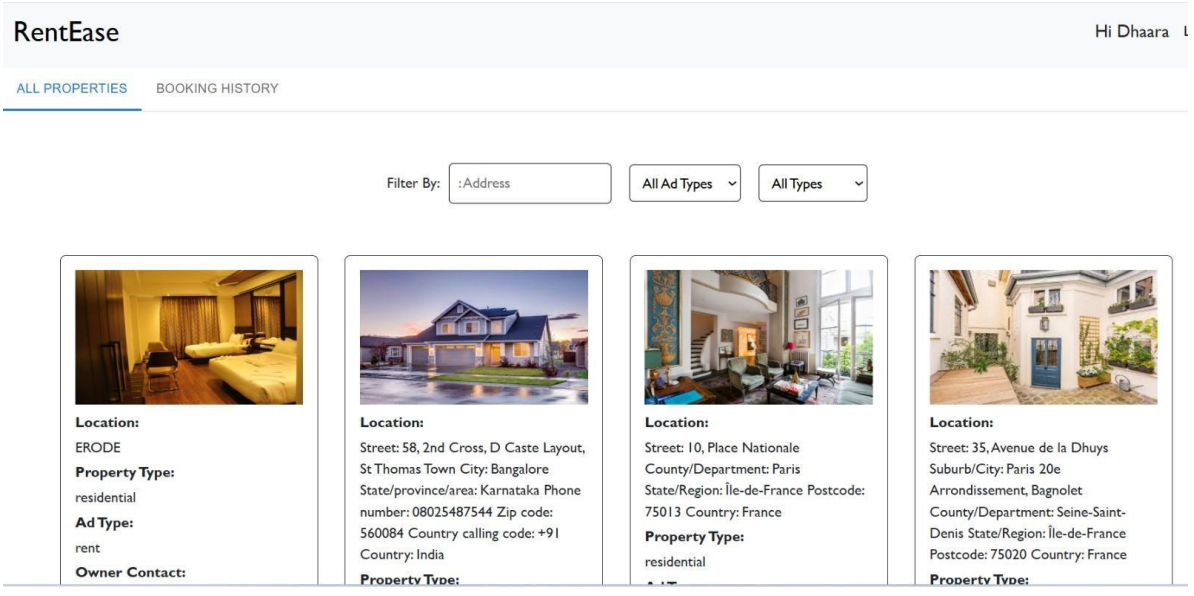
Output:

User Registration and Login:

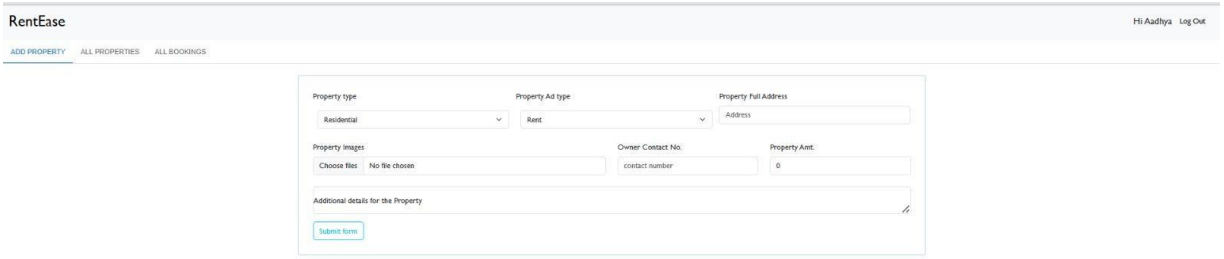


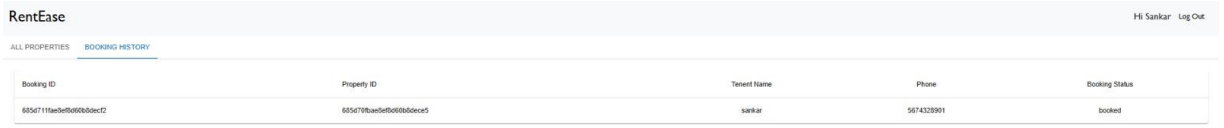


Property Listing Page

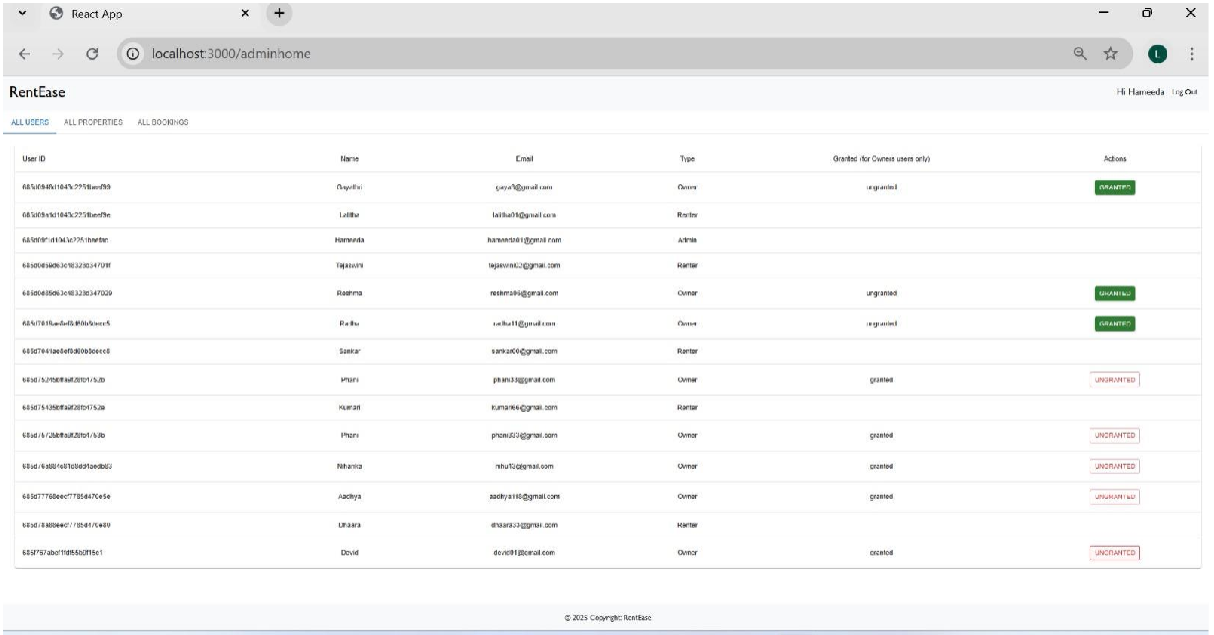


Booking Workflow





Admin Dashboard:



8. ADVANTAGES & DISADVANTAGES

Advantages

1. **User-Friendly Interface**: The React-powered frontend provides a smooth and intuitive user experience, even for first-time users.
2. **Centralized Platform**: Eliminates the need for third-party brokers by directly connecting renters and property owners.
3. **Real-Time Interaction**: Enables instant booking requests and quick approvals, improving efficiency in property rental processes.
4. **Role-Based Access**: Clear separation between renter, owner, and admin roles enhances security and platform integrity.
5. **Scalability**: Built with the MERN stack, the system is highly scalable and can handle growing data and user loads.
6. **Mobile Responsive**: The application adapts well across devices, offering accessibility from desktop, tablet, and mobile screens.
7. **Admin Monitoring**: Admins have control over listings and users, maintaining a safe and moderated platform environment.

Disadvantages

1. **Limited Payment Integration**: Currently, there is no online payment system integrated for rent deposits or booking charges.
2. **Basic Admin Features**: While functional, the admin dashboard is still minimal and could be expanded for better analytics and user management.
3. **No In-App Messaging**: Users must rely on external communication channels; the app lacks a built-in chat system.
4. **Lack of Email Notifications**: Actions like booking confirmations or listing approvals are not yet supported via email alerts.

9. CONCLUSION

The journey of building **House Hunt – Finding Your Perfect Rental Home** has been one of thoughtful exploration, technical mastery, and creative problem-solving. This project originated from a simple, yet widespread challenge — the hassle, uncertainty, and inefficiency involved in finding a rental home. Through every stage of development, our goal was clear: to transform this challenge into an opportunity for innovation, by building a platform that is not only functional but also user-centric and reliable.

House Hunt addresses the core pain points faced by both renters and property owners. With its intuitive user interface, clear role-based structure, and streamlined workflows, the application successfully brings all stakeholders together onto a single, accessible platform. By removing intermediaries, simplifying property discovery, and empowering users with direct control over listings and bookings, this platform redefines the rental experience in a digital-first world.

From a technical perspective, the project demonstrates strong proficiency in full-stack web development using the **MERN (MongoDB, Express.js, React.js, Node.js)** stack. The architecture ensures modularity and scalability, making it adaptable for future growth. Frontend and backend components have been integrated with seamless API communication, efficient database operations, and responsive design principles—creating a system that is robust, fast, and secure.

Beyond technical execution, this project also fostered critical thinking, design sensitivity, and time management skills. Each phase—right from ideation, empathy mapping, and requirement gathering, to testing, iteration, and documentation—was handled with diligence, creativity, and care.

In essence, **House Hunt** is more than just a software application. It is a **solution crafted with purpose**, a digital reflection of how traditional values like trust and convenience can be amplified through modern technology. It proves that with thoughtful design and purposeful coding, even complex human needs—like finding the perfect rental home—can be addressed in elegant, efficient, and empowering ways.

This project marks the **completion of one phase**—but opens the doors to countless possibilities ahead. With future enhancements, the platform can evolve into a full-fledged rental ecosystem, bringing even more convenience, trust, and intelligence into the world of real estate.

10. FUTURE SCOPE

The current version of House Hunt lays the groundwork for a comprehensive rental platform. In the future, the following features can be integrated to enhance the system's functionality, usability, and reach:

1. **Online Payment Gateway**
   To allow secure booking and rent payments directly through the platform using services like Razorpay, Stripe, or PayPal.

2. **In-App Chat System**
   Enable real-time communication between renters and property owners for faster decision- making and clarification.

3. **AI-Based Recommendations**
   Use machine learning to suggest properties based on user behavior, past searches, and preferences.

4. **Geo-Location Integration**
   Show properties on an interactive map, helping users visualize distance from key landmarks or workplaces.

5. **Mobile App Development**
   Expand the web application into a cross-platform mobile app using React Native for better on- the-go access.

6. **Rating and Review System**
   Allow users to leave feedback for owners and renters, enhancing credibility and trust on the platform.

7. **Email and SMS Notifications**
   Add automated alerts for actions like booking confirmations, approvals, and system updates.

11. APPENDIX

GitHub link for the video demo and documentation and also source code:

https://github.com/Mullaarif12/HouseHunt