HouseHunt: Property Rental & Management Platform

INTRODUCTION

The HouseHunt App is an innovative property rental platform designed to streamline the process of connecting renters with property owners. This system enables users to easily find, browse, and manage property inquiries and bookings, all within a user-friendly interface. By offering functionalities like property browsing, inquiry submission, and secure image uploading, the app caters to the needs of renters, owners, and administrators alike.

Renters can search for properties based on type, location, and availability, ensuring they find the right accommodation for their needs. Once a suitable property is selected, users can submit inquiries, manage their booking requests, and receive status updates. Owners benefit from a dedicated interface to manage their property listings, update availability, and communicate effectively with potential renters, while administrators oversee the app's smooth operation, ensuring compliance and resolving any disputes.

Built with a robust technical architecture, the HouseHunt App leverages a client-server model, using front-end frameworks like Bootstrap for an engaging user experience, and a backend powered by Express.js and MongoDB to handle secure data transactions. This system offers a seamless, efficient, and secure property rental experience, meeting the growing demand for accessible and well-organized housing services.

KEY FEATURES

RENTER REGISTRATION & PROFILE CREATION:

- **SECURE SIGN-UP** using email and password authentication.
- **PROFILE CREATION** that securely stores personal information for property inquiries.

PROPERTY BROWSING & FILTERING:

- **ALLOWS USERS TO SEARCH AND FILTER** properties based on type, address, and real-time availability.
- **LIVE AVAILABILITY UPDATES** ensure renters select only available properties, minimizing conflicts.

PROPERTY INQUIRY & BOOKING MANAGEMENT:

- **USER-FRIENDLY INQUIRY INTERFACE** where renters submit their details for

properties of interest.
- **AUTOMATED STATUS UPDATES** notify renters about the status of their booking requests (e.g., pending, approved, rejected).

OWNER'S DASHBOARD:

- **OWNERS CAN MANAGE PROPERTY LISTINGS**, view booking inquiries, and update booking statuses (e.g., pending, approved).
- **SECURE ACCESS TO RENTER DETAILS** for managing inquiries and communications.

ADMIN CONTROLS & APPROVAL:

- **ADMINS APPROVE OWNER REGISTRATIONS**, ensuring that only verified property owners are listed.
- **PLATFORM OVERSIGHT**, including user management, policy enforcement, and dispute resolution for a smooth user experience.

DESCRIPTION

The HouseHunt App is a user-centric platform designed to make property rental and management easy and efficient. The app connects renters and property owners through a streamlined digital interface, allowing users to search, filter, and inquire about properties based on type, location, and real-time availability.

For renters, the app offers secure registration, profile creation, and inquiry submission, with automated notifications and reminders to ensure timely updates on their requests. Owners benefit from a dedicated dashboard where they can manage their property listings, confirm inquiries, view renter details, and provide status updates. An admin interface allows for owner registration approvals, system monitoring, and compliance management, ensuring a smooth, reliable experience.

Built using Bootstrap for a modern frontend, the app also uses Axios for seamless backend communication, with Express.js and MongoDB handling server logic and data storage. Security libraries like bcrypt ensure secure handling of user data.

With features that enhance accessibility, communication, and efficiency, the HouseHunt App supports the growing demand for accessible housing options, providing renters with convenient, reliable access to properties while helping owners manage their listings effectively.

SCENARIO-BASED CASE STUDY

1. USER REGISTRATION:

John, a renter in search of a new apartment, downloads and opens the HouseHunt App. He starts by registering as a customer (Renter), providing his email address and creating a password to ensure a secure login. Once the registration process is complete, John is welcomed to the app with the option to log in.

2. BROWSING PROPERTIES:

After logging in, John is directed to a dashboard showcasing a list of available properties. The app offers various filters for him to search for properties based on criteria such as property type, address, and availability. John filters the list to find apartments in his desired area.

3. INQUIRING ABOUT A PROPERTY:

John selects a suitable apartment and clicks the "Get Info" button. An inquiry form appears, prompting John to provide his contact details. He can also add a message for the owner. Once the form is completed, John submits the inquiry request. He receives an immediate confirmation message indicating that his request is under review.

4. INQUIRY CONFIRMATION:

Sarah, the property owner, upon reviewing John's inquiry and her property's availability, confirms the booking request. The status of John's inquiry changes to "Approved," and John receives a notification with the property details and owner contact information via the app.

5. INQUIRY MANAGEMENT:

As the potential move-in date nears, John can access his booking history through the app's dashboard. Here, he can manage upcoming inquiries and view their updated statuses. If needed, he can contact the owner or the support team for assistance.

6. ADMIN APPROVAL (BACKGROUND PROCESS):

In the background, the app's admin is reviewing new owner registrations. Sarah, as a legitimate property owner, is approved and added to the platform. The admin ensures that only verified owners are listed, and the platform remains compliant with rental regulations and policies.

7. PLATFORM GOVERNANCE:

The admin monitors the platform's overall operation, addressing any issues, disputes,

or system improvements. Ensuring the app's compliance with privacy regulations and the terms of service is also a key responsibility, ensuring a smooth and secure experience for all users.

8. OWNER'S PROPERTY MANAGEMENT:

Sarah logs into her dashboard and reviews her listed properties. She sees John's inquiry and confirms the property's availability. Throughout the day, Sarah manages other properties, updates their statuses (e.g., rented, available), and ensures all inquiries are attended to efficiently.

9. RENTER-OWNER COMMUNICATION:

At the agreed time, John and Sarah connect to discuss the property details or arrange a viewing. Sarah provides further information about the property, and John's questions are addressed.

10. POST-BOOKING FOLLOW-UP:

After the booking is finalized (offline), Sarah updates the property's status within the app (e.g., isAvailable: false). John receives a final status update on his booking request via the app.

TECHNICAL ARCHITECTURE

The HouseHunt App features a modern technical architecture based on a client-server model. The frontend utilizes React.js with Bootstrap for a responsive user interface, with Axios handling seamless API communication. The backend is powered by Express.js, offering robust server-side logic, while MongoDB provides scalable data storage for user profiles, property listings, and booking inquiries. Authentication is secured using JWT for session management and bcrypt for password hashing. The admin interface oversees owner registration, platform governance, and ensures compliance, with Role-based Access Control (RBAC) managing access levels. Scalability is supported by MongoDB, and performance optimization is achieved with load balancing and caching techniques.

FRONTEND TECHNOLOGIES:

- **React.js:** A JavaScript library for building interactive and reusable user interfaces.
- **Bootstrap:** Provides a responsive and modern UI that adapts to various devices, ensuring a user-friendly experience.
- **Axios:** A promise-based HTTP client for making requests to the backend, ensuring smooth data communication between the frontend and server.

BACKEND FRAMEWORK:

- **Express.js:** A lightweight Node.js framework used to handle server-side logic, API routing, and HTTP request/response management, making the backend scalable and easy to maintain.

DATABASE AND AUTHENTICATION:

- **MongoDB:** A NoSQL database used for flexible and scalable storage of user data, property listings, and booking records. It supports fast querying and large data volumes.
- **JWT (JSON Web Tokens):** Used for secure, stateless authentication, allowing users to remain logged in without requiring session storage on the server.
- **Bcrypt:** A library for hashing passwords, ensuring that sensitive data is securely stored in the database.

ADMIN PANEL & GOVERNANCE:

- **Admin Interface:** Provides functionality for platform admins to approve owner registrations, manage platform settings, and oversee day-to-day operations.
- **Role-based Access Control (RBAC):** Ensures different users (Renters, Owners, Admins) have appropriate access levels to the system's features and data, maintaining privacy and security.

SCALABILITY AND PERFORMANCE:

- **MongoDB:** Scales horizontally, supporting increased data storage and high user traffic as the platform grows.
- **Load Balancing:** (Conceptual) Ensures traffic is evenly distributed across servers to optimize performance, especially during high traffic periods.
- **Caching:** (Conceptual) Reduces database load by storing frequently requested data temporarily, speeding up response times and improving user experience.

TIME MANAGEMENT AND SCHEDULING:

- While not as central as in a doctor app, booking requests in HouseHunt involve dates, and the system manages the creation and tracking of these requests with timestamps.

SECURITY FEATURES:

- **HTTPS:** (Conceptual) The platform uses SSL/TLS encryption to secure data transmission between the client and server.
- **Data Encryption:** Sensitive user information (passwords) is encrypted at rest using bcrypt. Data in transit is secured via HTTPS.

NOTIFICATIONS AND REMINDERS:

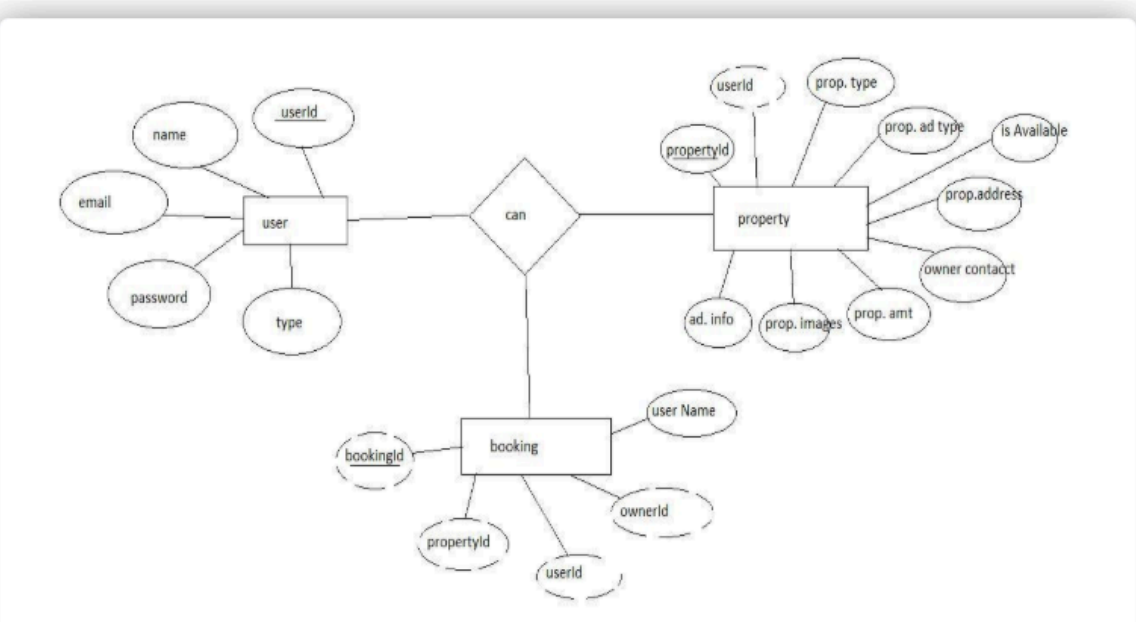
- **In-app Status Updates:** Notifications for booking request confirmations, rejections, and other status updates are provided within the app.

ER DIAGRAM

The Entity-Relationship (ER) diagram for the HouseHunt app represents three key entities: **Users**, **Properties**, and **Bookings**, with their respective attributes and relationships.

- The **Users** collection holds basic user information, including _id, name, email, password, type (to differentiate between Renters, Owners, and Admins), isApproved (for Owners), and phone (optional).
- The **Properties** collection stores information specific to property listings, such as their _id, owner (acting as a foreign key referencing the Users collection), propertyType, adAdType, address, description, images (array of URLs), rentAmount, and isAvailable. The owner field links each property to its corresponding owner's user account.
- The **Bookings** collection stores details about booking inquiries, including the _id, property (foreign key referencing the Properties collection), renter (foreign key referencing the Users collection), owner (foreign key referencing the Users collection), renterDetails (snapshot of renter info), and status (e.g., pending, approved, rejected). This collection maintains the relationship between renters, properties, and owners for each booking inquiry.

The relationships are as follows: a **User** can be linked to one or more **Properties** (one-to-many, where the User is an Owner), a **User** (Renter) can make multiple **Bookings** (one-to-many), and a **Property** can have multiple **Bookings** (one-to-many). The foreign keys owner in the Properties collection and property, renter, and owner in the Bookings collection establish these connections, enabling the app to manage the interactions between renters and owners effectively.



PRE REQUISITES

NODE.JS AND NPM:

- **Node.js:** A JavaScript runtime that allows you to run JavaScript code on the server-side.
- **npm (Node Package Manager):** Required to install libraries and manage dependencies.
- Download Node.js: Node.js Download

EXPRESS.JS:

- **Express.js:** A web application framework for Node.js that helps you build APIs and web applications with features like routing and middleware.
- Install Express: npm install express

MONGODB:

- **MongoDB:** A NoSQL database that stores data in a JSON-like format, suitable for storing data like user profiles, property details, and booking inquiries.
- Set up a MongoDB database (local or MongoDB Atlas).
- MongoDB Atlas: MongoDB Atlas

REACT.JS:

- **React.js:** A popular JavaScript library for building interactive and reusable user interfaces.

- Install React.js: npx create-react-app my-app (then navigate to my-app folder)

FRONT-END FRAMEWORKS AND LIBRARIES:

- **Bootstrap:** Utilized for a responsive and modern UI.
- **Axios:** A promise-based HTTP client for making requests to the backend.

DATABASE CONNECTIVITY (MONGOOSE):

- **Mongoose:** An Object-Document Mapping (ODM) library, to connect your Node.js backend to MongoDB for managing CRUD operations.
- Install Mongoose: npm install mongoose

SETUP AND INSTALLATION INSTRUCTIONS

CLONE THE PROJECT REPOSITORY:

- Download the project files or clone the repository using Git.
  git clone <your-househunt-repository-url>

INSTALL DEPENDENCIES:

- Navigate to the frontend and backend directories and install all required dependencies for both parts of the application.
  - **Frontend:**
    - Navigate to the frontend directory: cd frontend
    - Run: npm install
  - **Backend:**
    - Navigate to the backend directory: cd backend
    - Run: npm install

CONFIGURE MONGODB:

- Ensure you have a MongoDB Atlas cluster set up.
- In the backend directory, open the .env file and configure the connection URL for MongoDB:
  MONGO_URI=mongodb+srv://<your-atlas-username>:<your-atlas-password>@cluster0.your_cluster_id.mongodb.net/househunt?retryWrites=true&w=majority
- **Important:** Whitelist your IP address in MongoDB Atlas Network Access.

SET UP ENVIRONMENT VARIABLES:

- In the backend directory, create a .env file (if it doesn't exist) to store environment-specific variables:
  PORT=5000
  JWT_SECRET=your_strong_random_jwt_secret
  NODE_ENV=development

- Replace your_strong_random_jwt_secret with a securely generated key.

START THE DEVELOPMENT SERVERS:

- **Frontend:**
  - Open a terminal, navigate to the frontend directory.
  - Run: npm start
  - The application will be available at http://localhost:3000 in your browser.
- **Backend:**
  - Open a **separate** terminal, navigate to the backend directory.
  - Run: npm start
  - The backend API server will be running at http://localhost:5000.

ACCESS THE APPLICATION:

- After running both servers, access the HouseHunt Webpage in your browser at http://localhost:3000 for the frontend interface.

PROJECT STRUCTURE

The project is structured to include both Frontend and Backend components, each responsible for distinct tasks in the development of the HouseHunt Webpage.

FRONTEND PART:

The frontend is responsible for creating and rendering the user interface that renters, owners, and admins interact with. It consists of the following files and folders:

- **REACT COMPONENTS:** Each component is designed for user interactions such as displaying properties, submitting inquiries, and viewing booking statuses.
- **ROUTING:** Handles navigation between pages like renter dashboard, property detail, booking history, etc.
- **STATE MANAGEMENT:** Keeps track of the logged-in user's session, property details, and booking statuses.
- **STYLING:** Uses CSS and UI libraries (e.g., Bootstrap) to style the components.

BACKEND PART:

The backend handles the server-side operations, including user authentication, data handling, and API responses. It contains the following files and folders:

- **API ENDPOINTS:** Defines the routes for handling renter, owner, and admin functionalities, such as property listing, booking inquiries, and status updates.
- **DATABASE MODELS:** Defines schemas for Users, Properties, and Bookings using MongoDB and Mongoose.

- **AUTHENTICATION & AUTHORIZATION:** Manages login, registration, and access control for different user roles.
- **UPLOAD SYSTEM:** Handles file uploads for property images.
- **ADMIN FUNCTIONS:** Admin-related routes for managing users, properties, and owner approvals.

APPLICATION FLOW

This project has three main types of users: Renter, Owner, and Admin. Each has specific roles and responsibilities that are defined by the API endpoints.

RENTER/ORDINARY USER:

- **CREATE AN ACCOUNT & LOGIN:** Renters can register and log in using their email and password.
- **VIEW PROPERTIES:** After logging in, renters will see a list of available properties in their dashboard.
- **SUBMIT INQUIRY:** Renters can select a property and submit an inquiry by filling out a form with their contact details.
- **VIEW BOOKING STATUS:** Renters can track the status of their inquiries (pending, approved, rejected) and receive notifications when the status changes.
- **CANCEL INQUIRY:** (Conceptual) Renters can cancel their inquiries from their booking history page.

ADMIN:

- **MONITOR ALL OPERATIONS:** The admin oversees the platform, including the management of users, properties, and bookings.
- **APPROVE OWNER APPLICATIONS:** Admins can review and approve owner applications, making them available to list properties in the app.
- **MANAGE POLICIES:** Admin enforces platform policies, terms of service, and privacy regulations.
- **USER MANAGEMENT:** Admins can manage the profiles of renters and owners, monitor their actions, and maintain a secure environment.

OWNER:

- **ACCOUNT APPROVAL:** Owners must receive approval from the admin before they can list properties on the platform.
- **MANAGE PROPERTIES:** Once approved, owners can manage their listed properties, including adding, updating, and deleting listings.
- **MANAGE BOOKINGS:** Owners can manage the booking inquiries they receive from renters, including confirming or rejecting them.

- **PROPERTY NOTIFICATIONS:** Owners are notified when new inquiries are submitted or when renters update their inquiry details.

FINAL CONFIGURATION & RUNNING THE APP

Run Both Servers:

- The React frontend and Node.js backend servers should run simultaneously for the app to function correctly.
- You can open two terminal windows (one for frontend, one for backend) to run both servers together.
- **Frontend:** npm start (in frontend directory)
- **Backend:** npm start (in backend directory)
- Ensure the proxy setting in frontend/package.json is configured to http://localhost:5000.

VERIFYING THE APP:

- **Check Frontend:** Open your browser and go to http://localhost:3000. The React.js application should load with the property listings, inquiry forms, and status updates.
- **Check Backend:** Open Postman or any API testing tool to verify if the backend endpoints are working correctly, such as user login, owner registration, property creation, and booking inquiry submission.

ADDITIONAL SETUP

Version Control:

- Initialize a Git repository in the root of your project: git init
- Add and commit your project files: git add . and git commit -m "Initial commit"

Deployment (Optional):

- Consider using cloud platforms like Heroku, AWS, or Vercel for frontend hosting and backend API deployment.

FOLDER SETUP

The project is structured to include both Frontend and Backend components, each responsible for distinct tasks in the development of the HouseHunt Webpage.

PROJECT ROOT STRUCTURE:

```
project-root/
├── frontend/
└── backend/
```

BACKEND SETUP:

```
backend/
├── config/
├── controllers/
├── models/
├── routes/
├── middlewares/
├── uploads/
├── index.js  (or server.js as per some conventions)
└── .env
```

FRONTEND SETUP:

```
frontend/
├── public/
├── src/
│   ├── components/
│   ├── modules/ (containing admin, common, user sub-folders)
│   ├── context/
│   ├── images/
│   └── App.js
├── .env (optional, usually for build-time env vars)
└── package.json
```

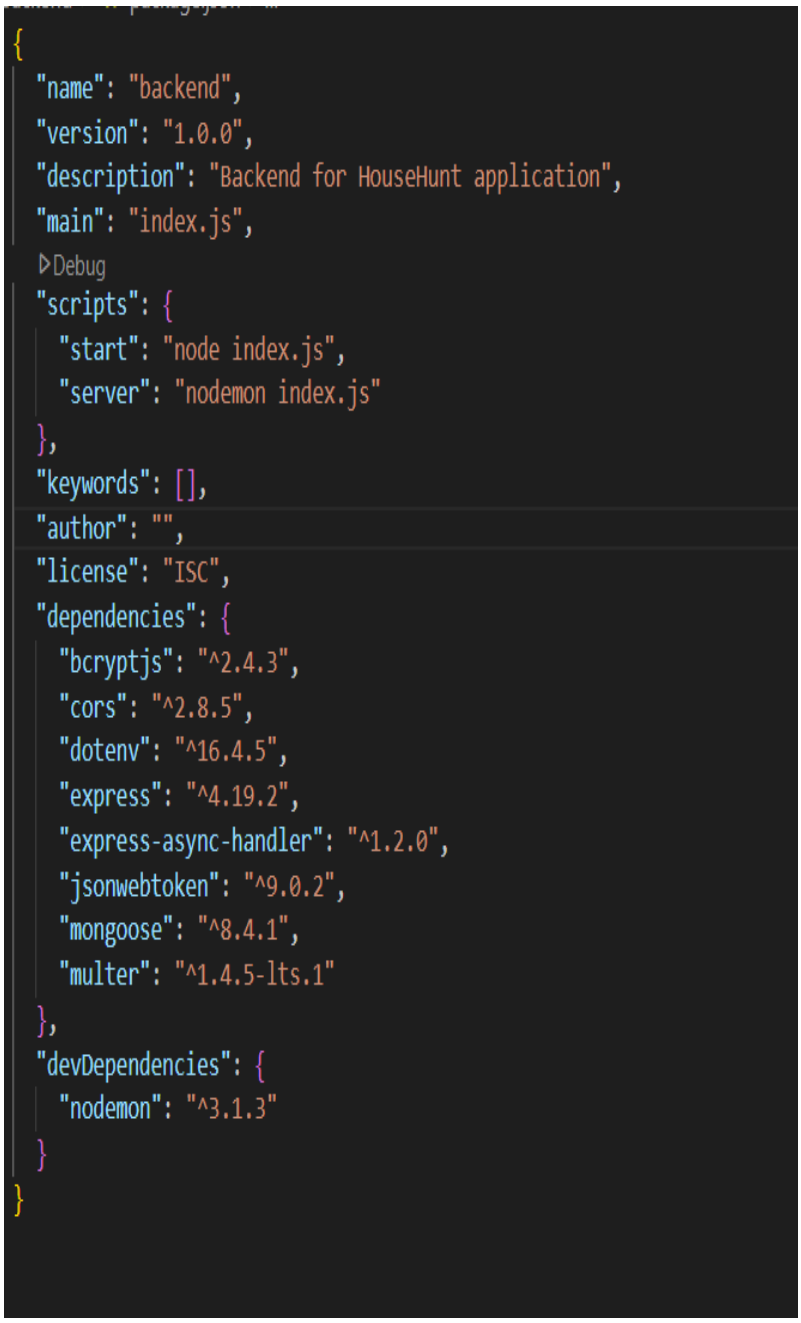
MILESTONE 1: PROJECT SETUP AND CONFIGURATION

Setting up a structured environment is crucial for any application. By organizing the project into separate folders for the frontend and backend, we ensure that the codebase is manageable and scalable.

PROJECT FOLDERS:

- **Frontend Folder:** Contains all code related to the user interface, written in JavaScript using React.js and Bootstrap.
- **Backend Folder:** Manages the server, API routes, and database interactions, typically handled through Node.js and Express.js.

```json
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "proxy": "http://localhost:5000",
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.14.3",
    "@mui/joy": "^5.0.0-beta.2",
    "@mui/material": "^5.14.5",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "antd": "^5.8.3",
    "axios": "^1.4.0",
    "bootstrap": "^5.3.7",
    "react": "^18.2.0",
    "react-bootstrap": "^2.10.10",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.15.0",
    "react-scripts": "5.0.1"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```



```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "Backend for HouseHunt application",
  "main": "index.js",
  ▷Debug
  "scripts": {
    "start": "node index.js",
    "server": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "express-async-handler": "^1.2.0",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.4.1",
    "multer": "^1.4.5-lts.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.3"
  }
}
```

LIBRARY AND TOOL INSTALLATION:

- **Backend Libraries:** Node.js, MongoDB, Bcrypt, Express.js, Dotenv, Mongoose, Multer, JSON Web Token.
- **Frontend Libraries:** React.js, Bootstrap, Axios.

MILESTONE 2: BACKEND DEVELOPMENT

The backend forms the core logic and data management for the project. A

well-structured backend ensures efficient data handling, security, and scalability.

EXPRESS.JS SERVER SETUP:

- **Express Server:** Acts as a hub for all requests and responses, routing them to appropriate endpoints.
- **Middleware Configuration:** Middleware like express.json() and cors enable cross-origin communication and request body parsing.

API ROUTE DEFINITION:

- **Route Organization:** Routes are organized by functionality (e.g., users, owners, admin, properties) for readability and maintainability.
- Express Route Handlers: Route handlers manage the flow of data between client and server, such as fetching, creating, updating, and deleting records.

DATA MODELS (SCHEMAS) WITH MONGOOSE:

- **User Schema:** Defines structure for user data and includes fields for personal information and role-based access (e.g., owner, renter). Using a schema ensures consistent data storage
- **CRUD Operations**: CRUD functionalities (Create, Read, Update, Delete) provide a standard interface for handling data operations. They simplify data manipulation in a structured, predictable way

USER AUTHENTICATION:

- **JWT Authentication**: JSON Web Tokens securely handle session management, ensuring that only verified users access protected routes. JWT tokens are embedded in request headers, verifying each request without storing session data on the server.

ADMIN AND TRANSACTION HANDLING:

- **Admin Privileges:** Administrators oversee user registrations, approve owner accounts, and manage platform settings.
- **Transaction Management:** This functionality allows renters to submit inquiries and owners to manage their properties and bookings in real-time.

ERROR HANDLING:

- **Middleware for Error Handling:** Catches issues and sends meaningful error responses with HTTP status codes.

MILESTONE 3: DATABASE DEVELOPMENT

Using MongoDB as the database provides a flexible, schema-less structure, perfect

for handling different types of user, property, and booking data.

SCHEMAS FOR DATABASE COLLECTIONS:

- **User Schema:** Defines fields for user information (name, email, password, type, isApproved).
- **Property Schema:** Defines fields for property listings (address, type, rent, images, owner).
- **Booking Schema:** Defines fields for booking inquiries (property, renter, owner, status).

DATABASE COLLECTIONS IN MONGODB:

- MongoDB collections, such as users, properties, and bookings, provide a structured, NoSQL approach to data management.

MILESTONE 4: FRONTEND DEVELOPMENT

Frontend development focuses on creating an interactive, intuitive user experience through a React-based user interface.

REACT APPLICATION SETUP:

- **Folder Structure and Libraries:** The React app structure is organized into components, modules (pages), and context.
- **UI Component Libraries:** Bootstrap provides pre-built components for rapid UI development and consistent design.

UI COMPONENTS FOR REUSABILITY:

- **Reusable Components:** UI elements like forms, dashboards, property cards, and buttons are designed as reusable components.
- **Styling and Layout:** Styling and layout components maintain a cohesive look and feel.

FRONTEND LOGIC IMPLEMENTATION:

- **API Integration:** Axios is used to make API calls to the backend.
- **Data Binding and State Management:** React's state management binds data to the UI, automatically updating it as the user interacts with the app.

MILESTONE 5: PROJECT IMPLEMENTATION AND TESTING

After completing development, running a final set of tests is crucial for identifying any bugs or issues.

VERIFY FUNCTIONALITY:

- Running the entire application ensures that each part (frontend, backend, database) works cohesively.
- Testing various user flows (e.g., owner registration/approval, property listing, renter inquiry, booking status update) helps confirm that all processes are functioning as intended.

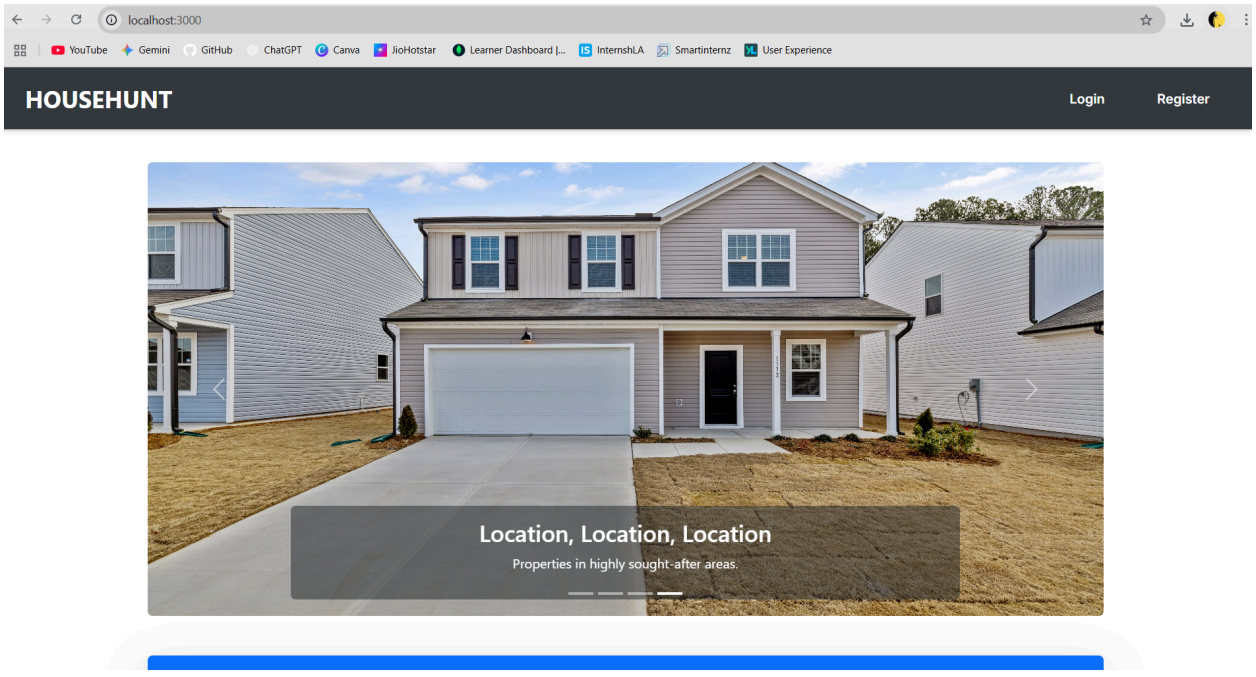
USER INTERFACE ELEMENTS:

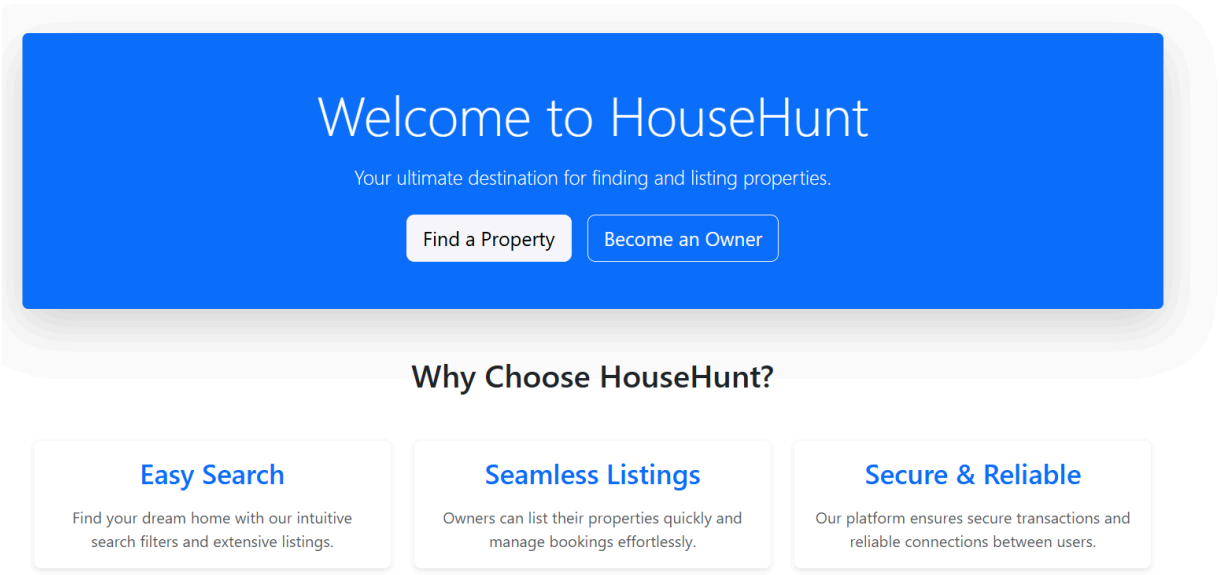
- Testing the UI includes verifying the look and feel of each page—landing, login, registration, and dashboards for different user types.
- Ensuring responsive design and usability across devices and screen sizes is also essential.

FINAL DEPLOYMENT:

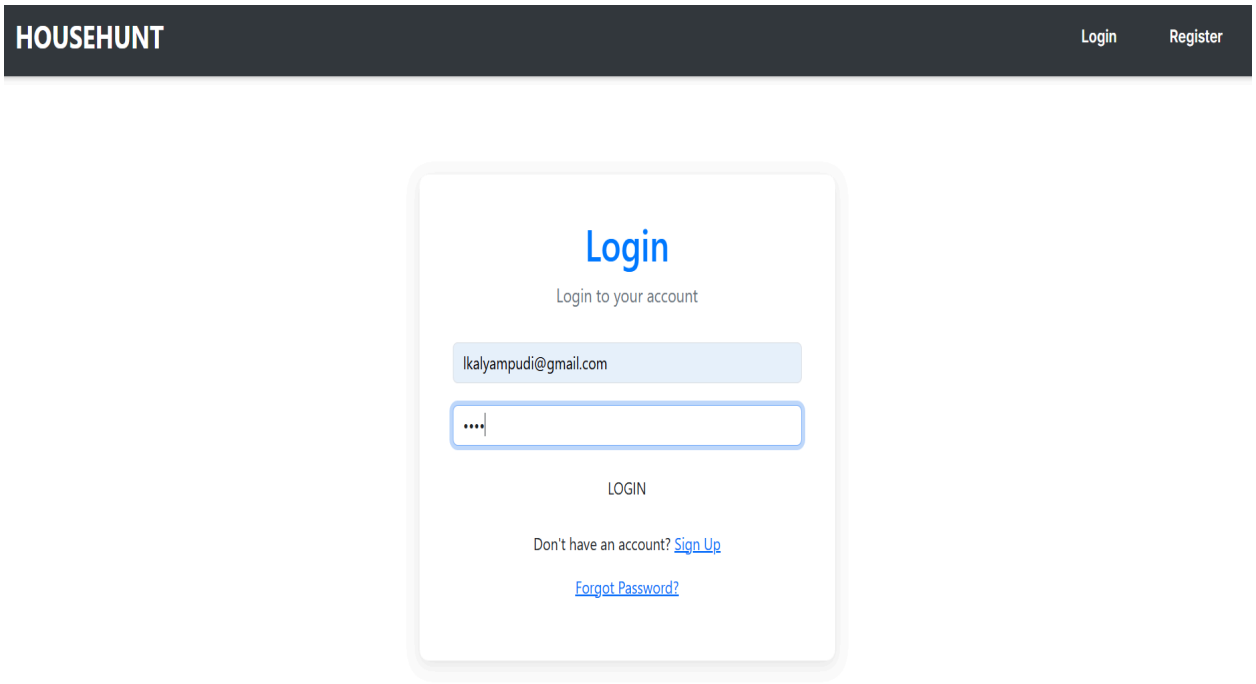
- Once testing is complete, the application can be deployed to a production server.

LANDING PAGE:

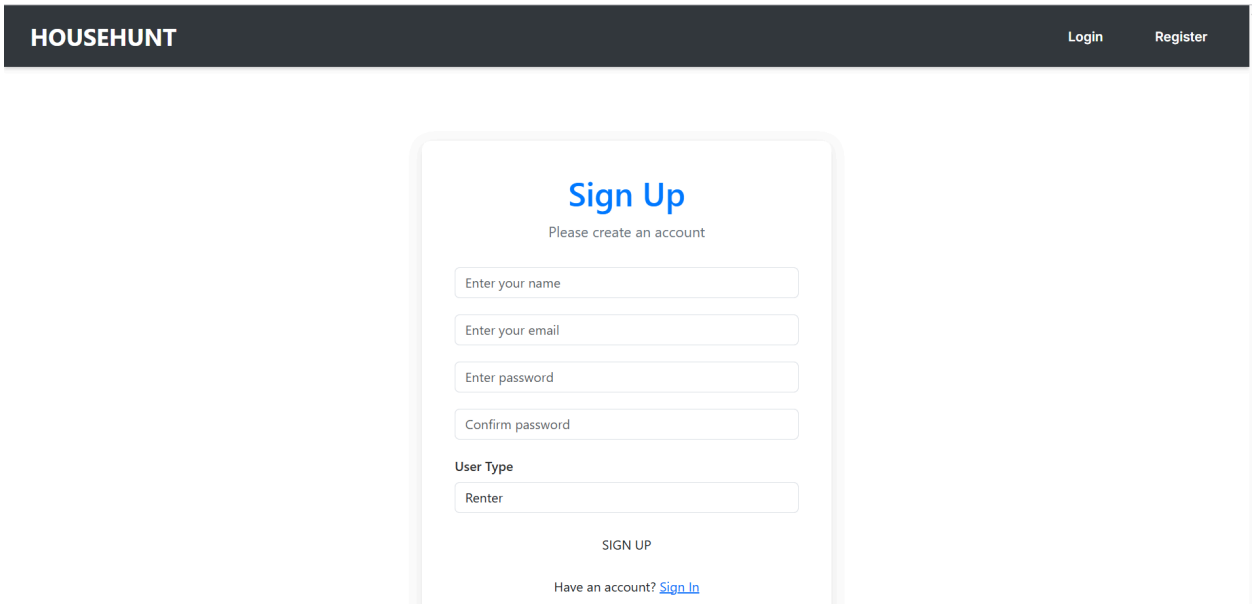




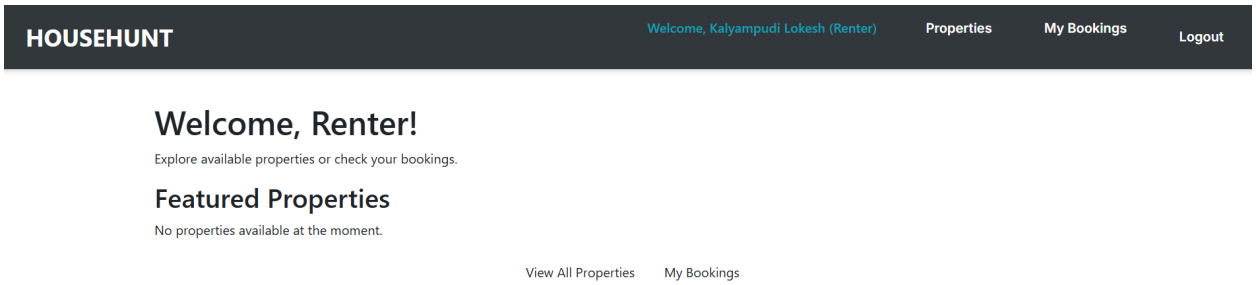
LOGIN PAGE :



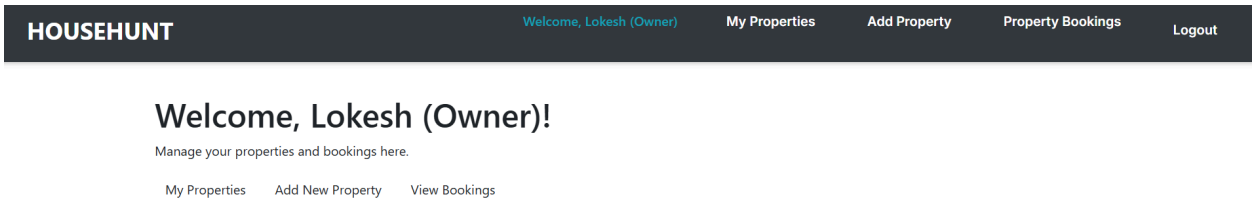
REGISTRATION PAGE:



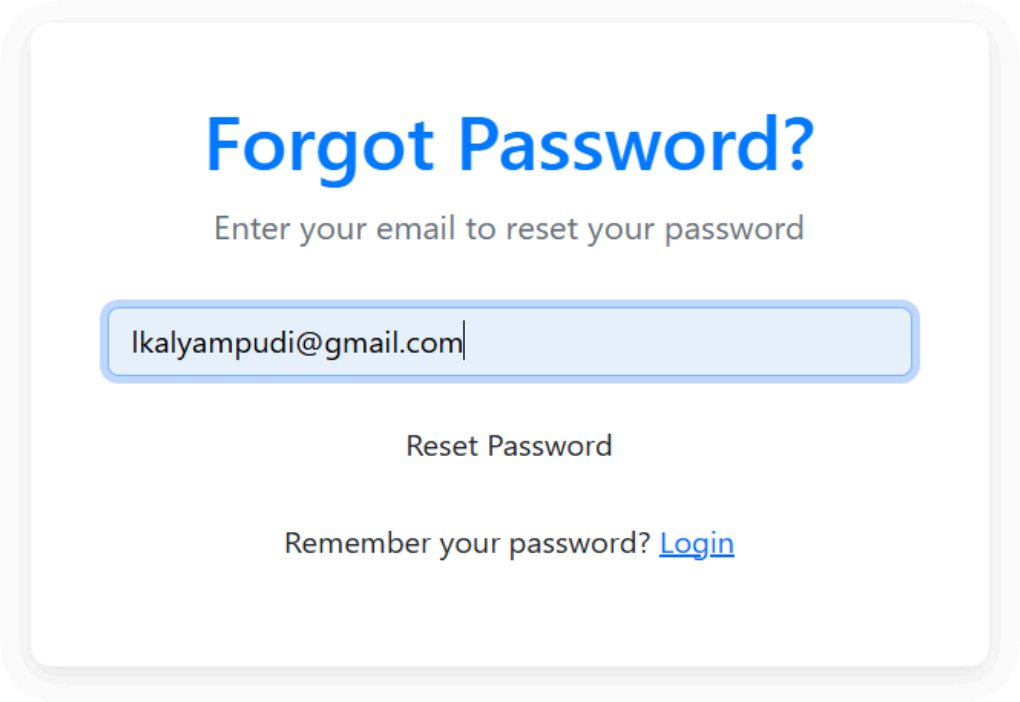
RENTER HOME:



OWNER HOME:



FORGET PASSWORD:



CONCLUSION:

This project outlines the development of a comprehensive property rental and management system with user roles for renters, owners, and admin. Each milestone—setup, backend, database, frontend, and final implementation—forms a structured foundation for a scalable application. The integrated functionalities allow for seamless property management, user authentication, and role-specific