1. INTRODUCTION

- **PROJECT TITLE:** HouseHunt: Finding Your Perfect Rental Home

- **TEAM MEMBERS:**

  Team ID: LTVIP2025TMID52222

  Team Size: 4

  Team Leader: L Bhanusree

  Team member: Irfa Fathima

  Team member: Syeda Misbah Mehnaaz

  Team member: K Nikhitha

2. PROJECT OVERVIEW

- **PURPOSE:**

The primary purpose of the HouseHunt project is to streamline and digitize the house rental process by offering a centralized, user-friendly platform for renters, property owners, and administrators. By leveraging modern web technologies (MERN stack), this application enhances the efficiency, transparency, and convenience of finding and managing rental homes.

- **FEATURES:**

1. **Simplify the Rental Search Process** ○ Provide renters with a centralized database of verified rental listings.
   - Allow users to filter and search for properties based on specific needs (e.g., budget, location, amenities).
2. **Enable Direct Communication Between Renters and Owners** ○ Facilitate secure, in-app messaging for inquiries and lease negotiations.
   - Build trust and reduce the need for third-party agents.
3. **Empower Property Owners with Management Tools** ○ Allow property owners to easily add, update, and manage property listings.
   - Let owners handle booking requests and communicate directly with potential tenants.
4. **Ensure Platform Integrity with Admin Oversight** ○ Admin dashboard for approving owners, monitoring platform activity, and enforcing policies.
   - Maintain a secure and trustworthy platform for all users.
5. **Provide End-to-End Rental Journey** ○ From discovery and inquiry to booking confirmation and lease agreement—everything happens within the app.
   - Support seamless move-in by providing clear rental steps and status updates.

6. **Support Scalability and Real-Time Data Handling** ○ Use MongoDB and Express for efficient data handling and scalability.
    - ○ Leverage responsive frontend (React + Bootstrap + Material UI) for optimal user experience.
7. **Build a Transparent and Secure Rental Ecosystem** ○ Keep users informed at every stage of the rental journey. ○ Provide secure login and authentication for all user roles (Renter, Owner, and Admin).

3. ARCHITECTURE

- **FRONTEND:**

The frontend of HouseHunt is built using React.js**,** following a component-based architecture. It uses React Router for navigation, Axios for API communication, and combines Bootstrap and Material-UI for responsive and modern UI design. Key features include role-based dashboards, reusable components (like property cards, forms, and filters), and dynamic rendering based on user roles (Renter, Owner, Admin). The app ensures a smooth user experience with structured routing, real-time data updates, and form validation

- **BACKEND:**

The backend of HouseHunt is built using Node.js and Express.js, forming a Restful API that handles all server-side logic. It connects to a MongoDB database using Mongoose to manage data such as users, properties, and bookings. The system supports user authentication and role-based access control using JWT **(**JSON Web Tokens**)** and bcrypt for secure password storage. The architecture is modular, with separate files for routes, controllers, models, and middleware, ensuring clean code structure, scalability, and maintainability. It enables smooth communication between the frontend and backend, ensuring real-time updates and secure data transactions.

- **DATABASE:**

MongoDB stores data in collections (like tables) and documents (like records) using JSON-like objects. Below are the three main schemas: Users, Properties, and Bookings.

Users:

1. _id: (MongoDB creates by unique default)
2. name
3. email
4. password
5. type **Property**:

1. userID: (can be act as foreign key)
2. _id: (MongoDB creates by unique default)
3. prop.Type
4. prop.AdType
5. isAvailable

6. prop.Address
7. owner contact
8. prop.Amt
9. prop.images
10. add.Info **Booking:**

1. _id: (MongoDB creates by unique default)
2. propertyId
3. userId
4. ownerId
5. username

4. SETUP INSTRUCTIONS

- **PREREQUISITES:**

1. **Node.js and npm –** JavaScript runtime and package manager for running backend code.
2. **Express.js** – Lightweight Node.js framework for handling APIs, routing, and middleware.
3. **MongoDB** – NoSQL database used to store user, property, and booking data.
4. **Mongoose** – ODM library for defining schemas and interacting with MongoDB in Node.js.
5. **React.js** – Frontend JavaScript library for building dynamic and reusable UI components.
6. **Ant Design** (Antd) – React-based UI component library for building elegant interfaces.
7. **Moment.js** – Library for parsing, formatting, and manipulating date/time in JavaScript.
8. **Bootstrap** – CSS framework for responsive and mobile-friendly UI layouts.
9. **Material UI** – React UI framework for designing modern and accessible components.
10. **HTML, CSS, JavaScript** – Core web technologies for structuring, styling, and scripting the frontend.
11. **Database Connectivity** – Use Mongoose to connect Node.js backend with MongoDB for CRUD operations

- **INSTALLATION:**

• Navigate into the cloned repository directory:
cd househunt
• Install the required dependencies by running the following commands:
cd frontend npm install cd ../backend npm install
Start the Development Server:
• To start the development server, execute the following command: npm start
• The house rent app will be accessible at http://localhost:3000

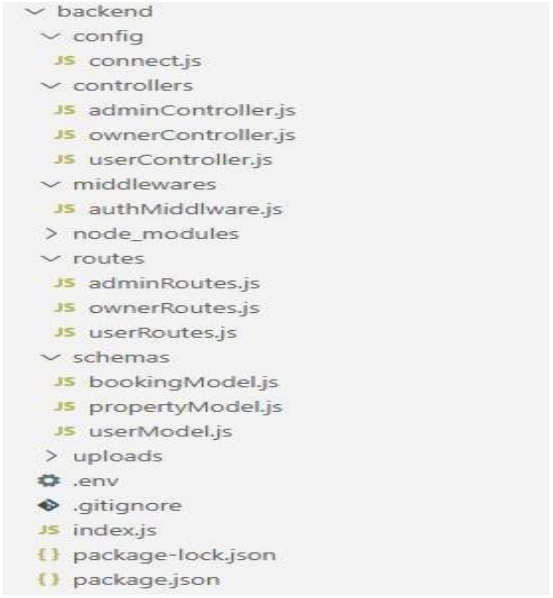
5. FOLDER STRUCTURE

- **CLIENT:**

The React frontend follows a modular structure for maintainability and scalability. It includes components for reusable UI parts, pages for route-level views, services for API calls, contexts for shared state like authentication, and assets for images and static files. The main routing logic is handled in App.js, and global styling is managed in the styles folder. This structure ensures a clean separation of concerns and efficient development workflow.

- **SERVER:**

The backend is organized into separate folders for routes (API endpoints), controllers (business logic), models (database schemas), middleware (auth and error handling), and config (database and environment settings). The main server file initializes the app and connects to MongoDB. This modular setup ensures clean code structure and easy maintenance.



6. RUNNING THE APPLICATION

- **FRONTEND:**

  ○ **npm install:** Installs all frontend dependencies. ○ **npm start:** Runs the React development server. ○ **npm run build:** Builds the app for production.

- **BACKEND:**

  ○ **npm install:** Installs all backend dependencies. ○ **npm start:** Starts the Node.js server. ○ **nodemon server.js:** Starts the server with auto-restart on file changes.

7. API DOCUMENTATION

1. User Authentication

POST `/register`
- Registers a new user.

Request Parameters (Body):

- name
- email
- password
- type (renter or owner)

Example Response:

```
json { message: "User already
exists", success: false
}
```

POST `/login` - User login returns JWT token.

Request Parameters (Body):

- email
- password

Example Response:

```
json {
 message: "Login success successfully",
success: true,   token,   user: user,
 }
```

2. Property Management

GET / `getAllProperties`

- Fetch all or filtered properties.

Query Parameters:

- Id
- Property type
- Property Ad type
- Property Address
- Owner Contact
- Property Amount

Example Response:

```
Json { success:
true, data:
allProperties
 }
```

POST / `postproperty`

- Add new property (Owner only).

Request Parameters (Body):

- Id
- Property type
- Property Ad type
- Property Address
- Owner Contact
- Property Amount

Example Response:

```
json
{
  "message": "Property added successfully",
  "propertyId": "64aabc1234"
}
```

DELETE `/deleteproperty/:propertyid`

- Delete a property (Owner only).

Example Response:

```
json
{
  "message": "Property deleted"
}
```

3. Admin Endpoints

GET `/getallusers`

- Get pending owner approvals (Admin only).

PUT `/handlestatus`

- Approve owner account (Admin only).

Example Response:

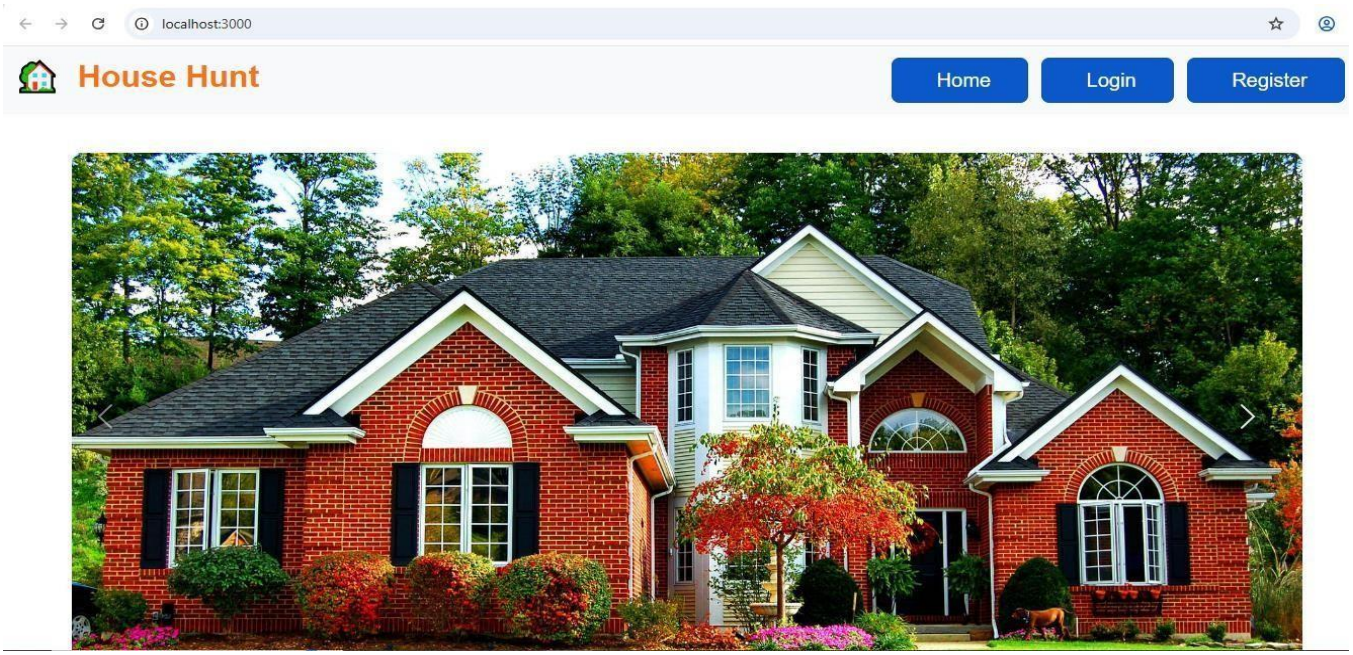
```
json
{
  "message": "Owner approved successfully"
}
```

7. AUTHENTICATION

The HouseHunt app uses **JWT (JSON Web Tokens)** for authentication and authorization. When a user logs in, the server generates a JWT containing the user's identity and role, which is sent back to the client.

- **Authentication**: The token proves the user's identity on each request and is stored client-side (usually in localStorage or cookies).
- **Authorization**: The backend verifies the token on every protected route and grants access based on the user's role (e.g., renter, owner, or admin).
- **Token Expiry**: Tokens have expiration times to enhance security, requiring users to reauthenticate after expiry.

8. USER INTERFACE



9. TESTING

- **Unit Testing -** Validates individual backend functions, controllers, and utilities in isolation to ensure each module works as expected.

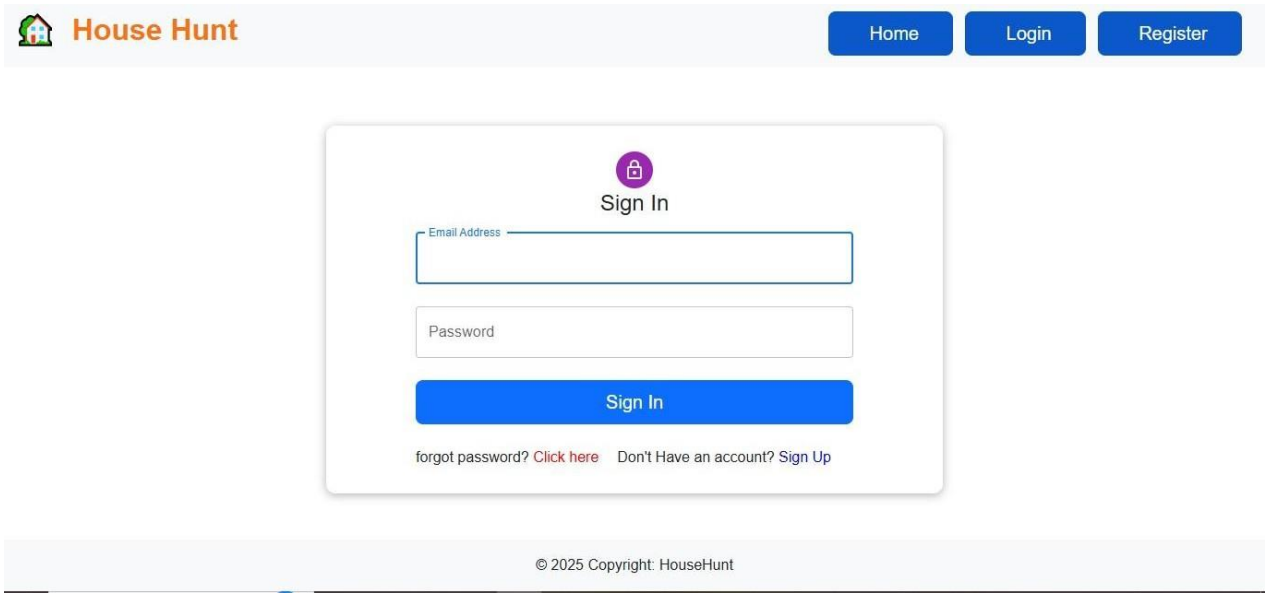
- **Integration Testing -** Checks the interaction between different backend modules like authentication, database operations, and API responses.
- **Manual UI Testing -** Ensures frontend components render and behave correctly across various user actions and screen sizes.

- **Security Testing (Basic) -** Ensures protected routes, authentication, and role-based access controls are functioning securely to prevent unauthorized access.

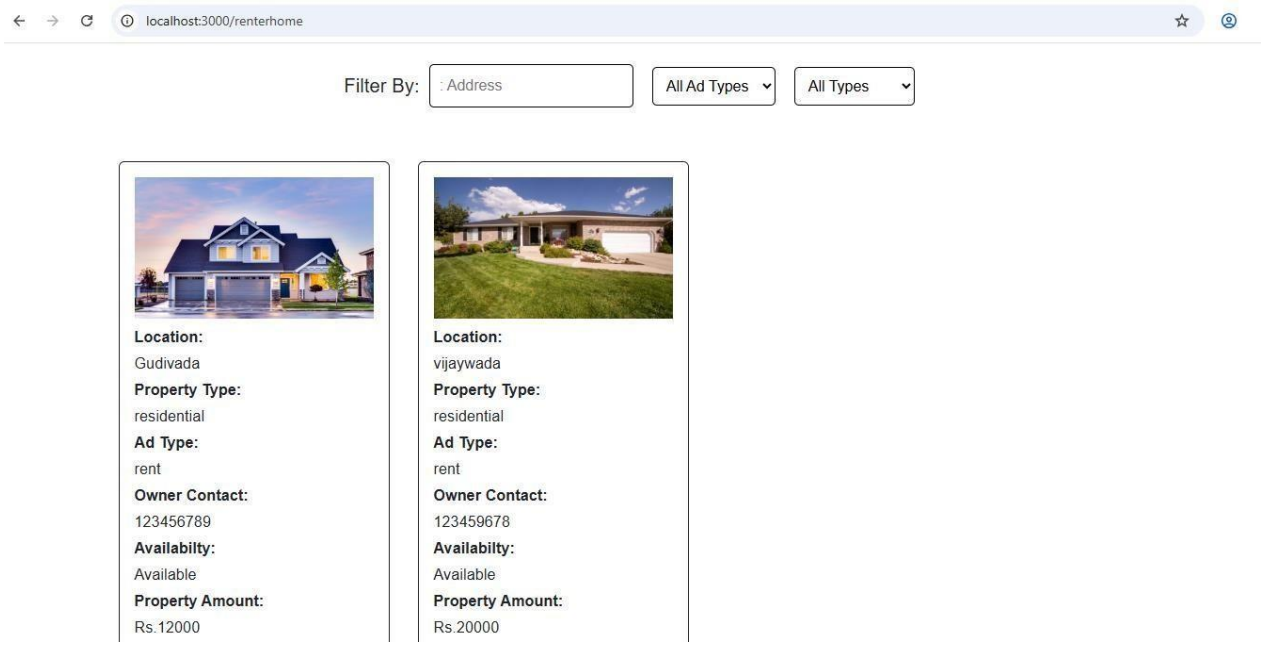
- **Postman Tests** - Used to verify API functionality, request/response behavior, and endpoint reliability through manual or automated test scripts.

10. SCREENSHOTS OR DEMO

Login Page:



Renter Page:



11. KNOWN ISSUES

- **Property Images Upload Delay -** Image uploads can be slow or fail intermittently due to lack of optimized file handling or cloud storage integration (e.g., Cloudinary/S3 not yet configured).
- **Missing Email Verification -** New users can register without verifying their email, which may lead to fake accounts or spam inquiries.
- **No Booking Conflict Check -** The system currently doesn't check for overlapping bookings, allowing double bookings for the same property.
- **Role-Based Access Not Fully Enforced -** Some admin or owner-specific routes are accessible without strict role validation, posing a security risk.

- **Mobile Responsiveness Issues -** Certain UI components (like modals and filters) break or overlap on smaller screens.
- **Booking Status Not Updated in Real-Time -** Booking status changes (like "confirmed") may not immediately reflect on the renter's dashboard without a manual refresh**.**

12. FUTURE ENHANCEMENTS

- **In-App Payments Integration** - Enable secure rent payments and deposits via Stripe, PayPal, or Razor pay.
- **Real-Time Chat System** - Implement Web Socket-based messaging for instant communication between renters and owners.
- **Ratings & Reviews** - Allow renters to review properties and rate owners, building trust and transparency.
- **Map-Based Property Search** - Integrate Google Maps or Map box to let users visually search for properties by location.
- **Push Notifications** - Send alerts for booking status, new listings, messages, or rent due reminders.
- **Document Upload & E-Signature** - Support digital lease agreements with identity verification and e-signature functionality.
- **Multi-Language & Localization Support** - Make the app accessible to users in different regions and languages.
- **AI-Powered Property Recommendations** - Use machine learning to suggest properties based on user preferences and search history.
- **Tenant Background Checks** - Allow owners to request renter background or credit history reports via third-party integrations.
- **Admin Dashboard Enhancements** - Add analytics, user activity logs, and report generation tools for better platform governance.