**Full Stack Development with MERN**

**1. Introduction**

**• Project Title:** Househunt-Finding Your Perfect Rental home.

**• Team Members:** Bandlamudi Ravi Kiran(Team Leader)

Annem Manasa(Team member)

Ande Yagnesh(Team member)

Arakasula Blessy Namratha(Team member)

**2. Project Overview**

**• Purpose:** The purpose of HouseHunt is to create a user-friendly, secure online platform . that connects renters with property owners, making it easier .

**• Features:**  User registration and secure login for renters and owners

 Verified property listings with photos and details

 Advanced search and filter options

 Booking requests and approval workflows

 In-app messaging between renters and owners

 Admin approval and monitoring

 Digital lease agreements

 Reviews and ratings system

 Responsive design for all devices

 Owner dashboard to manage listings and bookings

 Property availability calendar for scheduling rentals

 Favorites and wishlist for saving preferred properties

**3. Architecture**

**• Frontend:** The frontend is built using **React.js**, which provides a component-based.

1. **Components:** Separate components for registration, login, property listings, booking, messaging, and admin dashboard.
2. **Routing:** React Router handles navigation between pages without full page reloads.
3. **State Management:** Local state in components and Context API or Redux (optional) for shared state (e.g., user authentication status).
4. **API Communication:** Axios library fetches and posts data to the backend REST APIs.
5. **UI Libraries:** Bootstrap and Ant Design are used for responsive layouts and pre-built UI components

**• Backend:** The backend uses **Node.js** with the **Express.js** framework to build Restful APIs.

1. **Server Setup:** Express handles incoming HTTP requests and routes them to appropriate controllers.
2. **Authentication:** JSON Web Tokens (JWT) are used to secure endpoints and manage sessions.
3. **Middleware:** Custom middleware for error handling, authentication checks, and file uploads (Multer).

**• Database:** Data is stored in **MongoDB**, using **Mongoose** as an Object Data Modeling.

 **Schemas:**

1. **User Schema:** Stores user information (role, email, hashed password, verification status).
2. **Property Schema:** Contains property details (title, description, photos, price, owner reference).
3. **Booking Schema:** Tracks booking requests, statuses, renter and property references.
4. **Review Schema:** Holds user reviews and ratings for properties.

 **Relationships:**

1. Each property references the owner’s User ID.
2. Each booking references both the renter and the property.

 **CRUD Operations:**

1. Create: Adding new users, properties, bookings.
2. Read: Retrieving listings and user info.
3. Update: Editing property details, approving bookings.
4. Delete: Removing properties or bookings as needed.

**5. Setup Instructions**

**Client:**  Built with React components for each page (registration, booking, dashboards)

 React Router handles navigation

 Axios connects to backend APIs

 Bootstrap and Ant Design for styling

 Context API manages authentication state

**Server:**  Express routes define API endpoints

 Controllers handle business logic

 Mongoose models manage MongoDB data

 Middleware for authentication, error handling, and file uploads

 bcrypt.js and JWT secure user accounts

**6. Running the Application**

* + **Frontend:** npm start in the client directory.
  + **Backend:** npm start in the server directory.

**7. API Documentation**

* **Backend API Endpoints:**

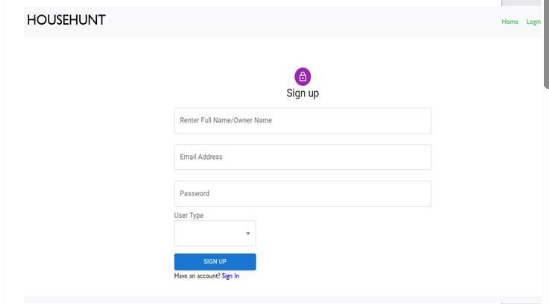
1. **POST /api/auth/register**  
   Register a new renter or owner.
2. **POST /api/auth/login**  
   Login and receive JWT token.
3. **GET /api/properties**  
   List all properties (supports filters).
4. **GET /api/properties/:id**  
   Get details of a specific property.
5. **POST /api/properties** (Owner)  
   Create a new property listing.
6. **PUT /api/properties/:id** (Owner)  
   Update a property listing.
7. **DELETE /api/properties/:id** (Owner)  
   Delete a property listing.
8. **POST /api/bookings** (Renter)  
   Create a booking request.
9. **PUT /api/bookings/:id/approve** (Owner)  
   Approve a booking request.
10. **GET /api/bookings/me**  
    List user’s bookings.

**8. Authentication**

**Authentication and Authorization:**

1. Users log in with email and password (hashed with bcrypt.js).
2. A **JWT token** is generated after login.
3. The token is sent in the **Authorization header** with API requests.
4. Middleware checks the token to verify identity and role (renter, owner, admin).
5. Access to routes is granted based on user role.

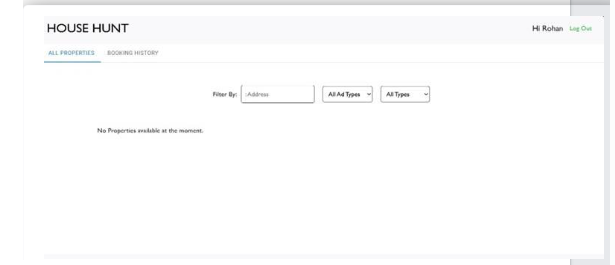
**9. User Interface**

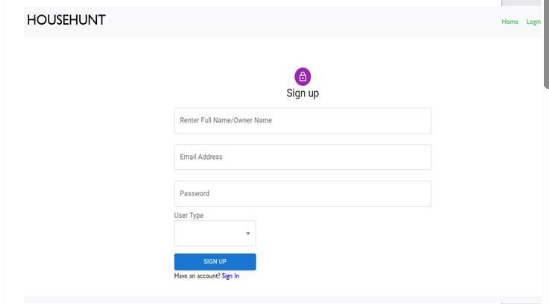


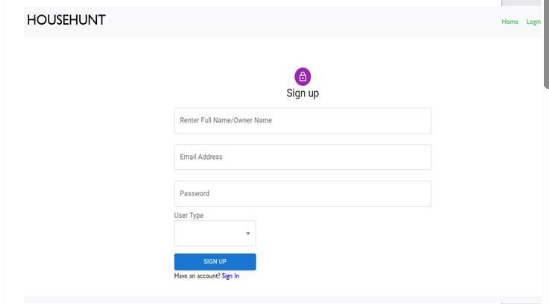
**10. Testing**

1. **Unit Testing:**  
   Test individual backend functions (e.g., user registration, property creation) to ensure each module works as expected.
2. **Integration Testing:**  
   Verify that frontend and backend work together correctly (e.g., registering a user and retrieving properties).
3. **End-to-End Testing:**  
   Simulate real user workflows like registration, login, searching properties, booking, and messaging.

**11. Screenshots or Demo**







**12. Known Issues**

1. Sometimes email verification emails may go to spam folders.
2. Image uploads can fail if the file size exceeds the allowed limit.
3. On slow networks, property images may load with a delay.

**13. Future Enhancements**

1. Online payment integration for rent deposits and transactions
2. SMS notifications for booking updates and messages
3. Advanced analytics dashboard for owners and admins
4. Tenant background verification and credit checks
5. Multi-language support for a broader audience
6. Push notifications in mobile browsers
7. AI-based property recommendations for renters
8. Calendar view for booking availability