HouseHunt Project Documentation

1. Introduction

Project Title: HouseHunt: Finding Your Perfect Rental Home

Team ID: LTVIP2025TMID52706
Team Size: 4
Team Leader: Namala Sreenivasulu (Role: Frontend - React.js)
Team Member: Mohammad Mahaboob Basha (Role: Database - MongoDB)
Team Member: Mekala Venkateswarlu (Role: Backend - Node.js, Express.js)
Team Member: Shaik Arshad Basha (Role: Backend - Bcrypt.js, CORS)

The House Hunt application is designed to solve practical problems in the house rental domain. It's a tenant looking for their next home, a landlord is seeking ease of Listing or an admin ensuring secure and trustworthy engagement. This platform provides a comprehensive, scalable, and user-friendly experience. The design promotes seamless User journey, emphasizing security, speed and simplicity.

2. Project Overview

Objective: To develop a full-stack web application that facilitates the search, listing, and booking of rental properties. The system caters to three distinct user roles: Renters (Tenants), Owners, and Admin, each with specific functionalities and responsibilities.

Core Technologies (MERN Stack with Enhancements):

- **Frontend:** React.js, Bootstrap, Material UI, Ant Design, Axios, Moment.js, MDB React UI Kit
- **Backend:** Node.js, Express.js, MongoDB, Mongoose, JWT (JSON Web Tokens), BCrypt.js, CORS, Dotenv, Moment.js, Multer, Nodemon
- **Database:** MongoDB (NoSQL)

3. Technology Stack

Frontend: - React.js with React Router and Context API for state management - Axios for async HTTP requests - UI Libraries: Material UI, Ant Design, Bootstrap, MDB React.

Backend: - Express.js (RESTAPI layer), Node.js (runtime) - bcrypt.js for hashing passwords, JWT for secure tokens - Multer for handling image file uploads

Database: - MongoDB with Mongoose for ODM functionality – Schema validation and population of related fields Additional Tools: - Postman for API testing – Nodemon for backend hot-reloading – Git/ GitHub for version control

The application utilizes a NoSQL MongoDB database, structured into three primary collections.

Database Design (MongoDB Collections)

3.1. users Collection

Stores information about all registered users (Renters, Owners, Admins).

- `_id`: (ObjectId) Automatically generated unique identifier by MongoDB.
- `name`: (String) Full name of the user.
- `email`: (String) User's email address (unique, used for login).
- `password`: (String) Hashed password for security.
- `type`: (String) Defines the user's role (e.g., "Renter", "Owner", "Admin").

3.2. properties Collection

Contains details about each rental property listed by owners.

- `_id`: (ObjectId) Automatically generated unique identifier.
- `userID`: (ObjectId) Foreign key referencing the `_id` of the `users` collection (the owner of the property).
- `propType`: (String) Type of property (e.g., "Apartment", "House", "Condo").

- `propAdType`: (String) Type of advertisement (e.g., "Rent" - given the project focus).
- `isAvailable`: (Boolean) Indicates if the property is currently available for booking.
- `propAddress`: (String) Full address of the property.
- `ownerContact`: (String) Contact information for the property owner.
- `propAmt`: (Number) Rental amount/price of the property.
- `propImages`: (Array of Strings) An array of URLs or paths to property images.
- `addInfo`: (String) Additional descriptive information about the property.

3.3. bookings Collection

Records all booking requests made by renters.

- `_id`: (ObjectId) Automatically generated unique identifier.
- `propertyId`: (ObjectId) Foreign key referencing the `_id` of the `properties` collection.
- `userId`: (ObjectId) Foreign key referencing the `_id` of the `users` collection (the renter who made the booking).
- `ownerId`: (ObjectId) Foreign key referencing the `_id` of the `users` collection (the owner of the booked property).
- `username`: (String) Name of the user who made the booking (for quick reference).
- `status`: (String) Current status of the booking (e.g., "pending", "approved", "rejected").

4. Architecture

The application is built using the MERN stack, following a clear separation of concerns. Frontend communicates with the backend using RESTful APIs through Axios. The backend handles routing, business logic, and data validation. MongoDB serves as the database for users, properties, and bookings. The system supports scalability through modular code practices and Mongo DB’s ability to horizontally scale.

5. Setup Instructions

Prerequisites: - Node.js and npm installed - MongoDB local instance or Atlas URI - Git

Steps:-

1.Clone repo: `git clone <repo-url>`

2. Install frontend: `cd frontend && npm install`

3. Install backend: `cd ../backend && npm install`

4. Configure `.env` with PORT, JWT_SECRET, and MONGO_URI

5.Run backend: `npm start`

6. Run frontend: `cd frontend && npm start` App is live at`http://localhost:3000`

6. Folder Structure

Client: - /src/modules/: Role-specific modules (admin, owner, renter) - /src/components/: Reusable

components like Navbar, Footer, Cards - /assets/: Static files (e.g., images, icons) Server: -

/models/: Schemas for users, properties, bookings - /routes/: API endpoints for auth, properties,

booking - /middleware/: Authorization and error middleware - /config/: Database connection config

7. Running the Application

Commands: -

Frontend: `cd frontend && npm start`

Backend: `cd backend && npm start`

8. API Documentation

Auth Routes: - POST /api/auth/register - POST /api/auth/login Property Routes: - GET

/api/properties - POST /api/properties - PUT /api/properties/:id - DELETE /api/properties/:id

Booking Routes: - POST /api/bookings - GET /api/bookings/user/:id - PUT /api/bookings/:id/status

Admin Routes: - GET /api/admin/users - PUT /api/admin/users/:id/approve

9. Authentication

Implemented using JWT. Tokens are issued on login and verified in protected routes using middleware.

- Passwords hashed with bcrypt.js - Token stored in localStorage - Middleware checks token validity

and user role

10. User Interface

Each role has a unique interface: - Renter: Browse/filter listings, view details, submit bookings -

Owner: Dashboard to manage properties, view bookings - Admin: Control panel to approve/reject

owners, monitor system activity UX/UI guidelines were followed to ensure clean and responsive

layout using grid systems and cards.

11. Testing

Test Coverage: - Manual API testing via Postman - Cross-browser UI testing (Chrome, Edge, Firefox)

- Login/Booking validations tested with both valid and invalid input Plans: - Unit tests for API

endpoints with Jest - UI flow tests with Cypress

12. Known Issues

- No pagination on property list - File size/image type validation needed - Booking rejection not

implemented - Admin can't edit/delete users yet

13. Future Enhancements

- Integrate Stripe/Razorpay for payments - Add renter-owner messaging system - Admin analytics

dashboard - Google Maps API integration for location selection - Create mobile version with React

Native - Lease agreement upload & digital signature system

14. Contribution Guidelines (Optional)

- Follow ESLint and Prettier config rules - Use feature branches for pull requests - Submit clear

documentation for added components Development Principles: - DRY (Don’t Repeat Yourself) - Reusable

component design - Clean commit history and messages