Full Stack Development with MERN

1.Introduction 3-4

- Project Title
- Team members

2.Project Overview 5-6

- Purpose
- Features

3.Architecture 7-9

- Frontend
- Backend
- Database

4.Setup Instructions 10-12

- Prerequisites
- Installation

5.Folder Structure 13-15

- Client
- Server

6.Running the Application 16-17

7.API Documentation 18-22

8.Authentication 23-25

9.User Interface 26-27

10.Testing 28-29

- Describe the testing strategy and tools used

11.Known Issues 30-31

- Document any known bugs that users to showcase the application

12.Feature Enhancements 32-33

Introduction

- ❖ Project Title: HouseHunt - Finding Your Perfect Rental Home
- ❖ Team Members: 1) Hema Lavanya Balam
  2) Turram Yamini Geetha
  3) Kovvuri Sri Sai Divya
- ❖ Roles:

➢ Lavanya Balam served as the Frontend Developer, responsible for designing and developing the user interface using React.js. She focused on creating responsive and user-friendly pages for a seamless user experience.

➢ Yamini Geetha worked as the Backend Developer, managing the server-side development using Node.js and Express.js. He handled API creation, user authentication, and integration with the database.

➢ Sai Divya took the role of UI/UX Designer and Documentation Lead. She worked on designing wireframes, improving the overall design flow, and prepared detailed project documentation along with performing testing tasks.

In our team of three members, we collaborated closely throughout the development of the HouseHunt project. We began by dividing the core responsibilities based on our individual strengths and interests. One member focused on designing and building the frontend using React.js to ensure a smooth and user-friendly interface. Another member took charge of backend development using Node.js and Express.js, handling API creation and database connectivity with MongoDB. The third member coordinated both sides and worked on integrating the frontend and backend seamlessly. We frequently discussed progress, fixed bugs together, and tested features in teams. All members contributed to the documentation and final presentation. Important decisions, such as UI layout, data structure, and deployment planning, were made through group discussions. We used GitHub to share code and track changes efficiently. Overall, the project was a team effort with equal contributions and strong coordination.

Project Overview

- Purpose: The purpose of the HouseHunt project is to provide a user-friendly, efficient, and centralized platform for individuals seeking rental properties. It aims to simplify the traditional process of house hunting by offering a digital solution where users can search, filter, and view rental listings based on their preferences such as location, price, property type, and amenities.

  The primary goal is to bridge the gap between property owners and potential tenants by offering a smooth browsing experience, secure communication, and reliable property information—all in one place.

  - Features: The HouseHunt application includes the following key features and functionalities:
  - Property Search: Users can search for rental properties based on location, budget, property type, and number of bedrooms.
  - Detailed Listings: Each property includes images, rent details, description, location map, and owner/agent contact information.
  - Filter & Sort Options: Advanced filters to refine search results by rent range, furnished/unfurnished status, and more.
  - User Authentication: Secure login and registration system for both tenants and property owners.
  - Add Property Listings: Property owners can upload and manage their property listings with photos and details.
  - Responsive Design: Fully responsive interface compatible with desktops, tablets, and mobile devices.

- Booking or Contact Feature (optional): Option for users to contact owners or schedule a visit.
- Secure Data Handling: User data is stored securely with proper backend validation.

Architecture

- The HouseHunt application follows a full-stack web architecture comprising a React-based frontend, a Node.js and Express.js-powered backend, and MongoDB as the database. The architecture is designed to ensure seamless interaction between users and data, efficient request handling, and a responsive UI experience.
- Frontend Architecture (React.js):The frontend of HouseHunt is developed using React.js, a component-based JavaScript library for building dynamic user interfaces. It follows a modular and reusable component structure, enabling scalability and maintainability.

Key Aspects:

1. Component-Based Structure: Pages such as Home, Search, Property Details, Login/Register are built as separate components.
2. Routing: Implemented using React Router to enable smooth navigation between views without page reloads.
3. State Management: Utilizes React's useState and useEffect hooks for managing component states and API data.
4. API Integration: Communicates with the backend via axios for asynchronous data fetching (e.g., login, get property listings).
5. Responsive UI: Styled using CSS or TailwindCSS to ensure a mobile-friendly, user-friendly experience.

- **Backend Architecture (Node.js + Express.js):** The backend is built using Node.js with the Express.js framework, designed to handle HTTP requests, manage routes, and connect to the database.

  Key Features:

  1. RESTful APIs: Created for all core functionalities such as user authentication, property creation, and property search.
  2. Routing: Structured into route files (/auth, /properties) with proper middleware for validation and authentication.
  3. Middleware: Uses express.json(), custom error handlers, and middleware for validation (e.g., express-validator, JWT verification).
  4. Authentication: Implements JWT (JSON Web Tokens) for secure login and role-based access (user vs. admin).
  5. CORS & Security: Configured CORS and environment variables for secure backend communication.

- **Database Design (MongoDB):** MongoDB is used as the NoSQL database for storing user and property data in a flexible, JSON-like document format.

  Main Collections:

  1. Users

```
{
  "_id": ObjectId,
  "name": "Lavanya",
  "email": "lavanya@email.com",
  "password": "hashed_password",
  "role": "user" // or "owner"
}
```

2. Properties

```
{
  "_id": ObjectId,
  "title": "2 BHK Flat in Hyderabad",
  "description": "Spacious flat near IT hub",
  "location": "Hitech City",
  "rent": 12000,
  "bedrooms": 2,
  "furnished": true,
  "ownerId": ObjectId (refers to Users),
  "images": ["img1.jpg", "img2.jpg"]
}
```

Database Interactions:

1. Mongoose ODM is used to define schemas, interact with MongoDB, and handle relationships (e.g., user to properties).
2. Supports CRUD operations: Create new listings, Read/search listings, Update property info, Delete properties.
3. Validation: Input validation and schema-level constraints ensure data consistency.

Setup Instructions

➢ This section outlines the steps required to set up the HouseHunt: Finding Your Perfect Rental Home project on a local development environment.

- Prerequisites:

Before beginning the installation, ensure the following software and tools are installed:

✓ Node.js – JavaScript runtime environment for running backend and frontend

👉 https://nodejs.org/

✓ MongoDB – NoSQL database for storing property and user data

👉 https://www.mongodb.com/try/download/community

✓ Git – Version control system to clone the project repository

👉 https://git-scm.com/

✓ Visual Studio Code (or any code editor) – For editing and running the project

👉 https://code.visualstudio.com/

✓ Postman – For testing backend APIs

✓ MongoDB Compass – GUI tool to view and manage MongoDB data

- Installation:

Follow the steps below to clone and run the HouseHunt project on your local system:

Step 1: Clone the Repository

1. Open your terminal or command prompt.

2. Run the following command

```
git clone
https://github.com/your-username/househunt.git
cd househunt
```

Step 2: Backend Setup

1. Navigate to the backend folder:

```
cd backend
```

2. Install dependencies:

```
npm  install
```

3. Create a file named .env in the backend directory and add the following environment variables:

```
PORT=5000
MONGO_URI=mongodb://localhost:27017/househunt
JWT_SECRET=your_jwt_secret_key
```

4. Start the backend server:

```
npm  start
```

Step 3: Frontend Setup

1. Open a new terminal window.

2. Navigate to the frontend folder:

```
cd frontend
```

3.Install dependencies:

```
npm install
```

4. Start the React frontend server:

```
npm  start
```

➢ Accessing the Application

Once both frontend and backend servers are running:

Frontend: http://localhost:3000

Backend API: http://localhost:5000/api

Folder Structure

The HouseHunt project follows a clear and modular folder structure to separate concerns between the client-side (frontend) and server-side (backend) codebases. This ensures scalability, maintainability, and clarity during development.

Client (Frontend - React.js)

The frontend is developed using React.js, structured around components and pages for a responsive user interface.

📁 frontend/ – Main folder for the client application

```
frontend/
|
├── public/          # Static files like index.html
|
├── src/             # Main source folder
|   ├── assets/          # Images, icons, or other static assets
|   ├── components/       # Reusable UI components (Navbar,
Footer, Property Card)
|   ├── pages/           # Pages like Home, Login, Register, Property
Details
|   ├── services/         # Axios calls or API services
|   ├── App.js           # Main component with routing
|    ├── index.js          # React entry point
|
├── .env                  # Environment variables
├── package. json          # Project metadata and dependencies
```

Highlights:

- components/ contains modular UI blocks used across pages.
- pages/ represents each major screen (Home, Listings, Add Property).
- services/ manages all API calls in one place using Axios.
- Routing is handled using react-router-dom in App.js.

server (Backend - Node.js + Express.js)

The backend is built using Node.js and Express.js, structured in a RESTful pattern.

📁 backend/ – Main folder for the backend application

```
backend/
|
 ├── config/          # Database connection and environment
setup
|   └── db.js
|
 ├── controllers/          # Logic for handling requests (e.g.,
auth, property)
|   └── propertyController.js
|
 ├── middleware/         # JWT auth, error handling, request
validation
```

```
│   └── authMiddleware.js
│
├── models/              # Mongoose schemas (User, Property)
│   └── Property.js
│
├── routes/              # API route definitions
│   └── propertyRoutes.js
│
├── .env                # Environment configuration
├── server.js             # Entry point for the backend server
├── package. json         # Backend dependencies and scripts
```

Highlights:

- ✓ models/ contains MongoDB schemas using Mongoose.
- ✓ routes/ maps HTTP endpoints to controller functions.
- ✓ controllers/ handle the business logic.
- ✓ middleware/ includes custom logic like JWT authentication.

Running the Application

To run the Househunt application locally, you'll need to execute commands in both the frontend and backend directories.

Here are the commands to start both servers:

1. Start the Frontend Server:

Navigate to the client directory:

Bash

```
cd client
```

Start the frontend application:

Bash

```
npm start
```

This command will typically open the frontend application in your default web browser (e.g., http://localhost:3000).

2. Start the Backend Server:

Navigate to the server directory:

(You'll likely need to open a new terminal window or tab for this, or navigate back to your project's root and then into the server directory)

Bash

```
cd server
```

Start the backend application:

Bash

```
npm start
```

This command will start the backend server, which will likely run on a different port (e.g., http://localhost:5000 or http://localhost:8080), and handle API requests from the frontend.

Important Notes:

Prerequisites: Ensure you have Node.js and npm (Node Package Manager) installed on your system.

Dependencies: Before running npm start for the first time in each directory, you might need to install the project dependencies by running npm install in both the client and server directories.

Port Conflicts: If you encounter issues with ports already being in use, you might need to change the default ports in your application's configuration files (e.g., .env files).

Environment Variables: Check if there are any required environment variables (e.g., for database connections, API keys) that need to be set up in .env files in either the client or server directories.

API Documentation

- The backend of HouseHunt exposes a set of RESTful APIs that allow clients (frontend or third-party) to interact with the system, including user authentication and property management.

✅ POST /api/auth/register

Description: Register a new user (tenant or property owner)

Request Body:

```
{
  "name": "Lavanya",
 "email": "lavanya@email.com",
 "password": "secure Password"
}
```

Response:

```
{
 "message": "User registered successfully",
 "user Id": "64bde123456789..”
```

✅ POST /api/auth/login

Description: Authenticate user and return JWT token

Request Body:

```
{
  "email": "lavanya@email.com",
  "password": "secure Password"
}
```

Response:

```
{
  "token": "jwt_token_string"
}
```

➢ Property Routes

All property routes require JWT token in headers:

Authorization: Bearer <token>

✅ POST /api/properties/

Description: Add a new property listing (owner only)

Request Body:

```
{
  "title": "2 BHK Flat in Hyderabad",
  "description": "Spacious flat with amenities",
  "location": "Madhapur",
  "rent": 15000,
  "bedrooms": 2,
  "furnished": true,
  "images": ["image1.jpg", "image2.jpg"]
}
```

Response:

```
{
  "message": "Property added successfully",
  "property Id": "64c0ef12345..."
}
```

✅ GET /api/properties/

Description: Get all property listings (public)

Query Parameters (optional):

- ✓ location: Filter by location
- ✓ minRent: Minimum rent
- ✓ maxRent: Maximum rent
- ✓ bedrooms: Number of bedrooms

Example:GET/api/properties?location=Hyderabad&minRent=10000

Response:

```
[
 {
   "_id": "64c0ef12345...",
  "title": "2 BHK Flat in Hyderabad",
  "rent": 15000,
  "location": "Madhapur",
  "furnished": true,
  "bedrooms": 2
 }
]
```

✅ GET /api/properties/:id

Description: Get details of a specific property by ID

Example: GET /api/properties/64c0ef12345...

Response:

```
{
  "_id": "64c0ef12345...",
  "title": "2 BHK Flat in Hyderabad",
  "description": "Spacious flat",
  "location": "Madhapur",
  "rent": 15000,
  "bedrooms": 2,
  "furnished": true,
  "images": ["image1.jpg", "image2.jpg"],
  "owner Id": "64beabc..”
}
```

✅ PUT /api/properties/:id

Description: Update a property listing (owner only)

Request Body (partial or full):

```
{
  "rent": 16000,
  "furnished": false
}
```

Response:

```
{
  "message": "Property updated successfully"
}
```

✅ DELETE /api/properties/:id

Description: Delete a property listing (owner only)

Response:

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

✅ Headers for Protected Routes

All protected routes must include the Authorization header:

```
Authorization: Bearer <your_token_here>
```

➢ Testing

You can test all routes using Postman or any API client.

Authentication

- Authentication and Authorization

  Authentication and authorization in the HouseHunt project are handled using JWT (JSON Web Tokens) to ensure secure access to protected routes and resources. This mechanism ensures that only registered and authenticated users can perform actions such as adding, editing, or deleting property listings.

- Authentication Flow

  1. User Registration (/api/auth/register):

➢ New users (property owners or tenants) can register by providing their name, email, and password.

➢ Passwords are securely hashed using bcrypt before storing them in the database.

2. User Login (/api/auth/login):

➢ Users log in with their email and password.

➢ On successful login, the server verifies the credentials and issues a JWT token.

➢ The token is sent to the client in the response and must be stored (e.g., in localStorage or cookies).

- Token-Based Authentication (JWT)

➢ Upon login, the backend generates a JWT token signed with a secret key (stored securely in .env).

➢ The token includes:

➢ User ID

- ➢ Role (e.g., user, owner)
- ➢ Expiration time

Example JWT Payload:

```
{
  "user Id": "64be123...",
  "role": "owner",
  "iat": 1712340000,
  "exp": 1712376000
}
```

- Authorization
- For protected routes (like posting or updating a property), the JWT token must be sent in the Authorization header:

  - ✓ Authorization: Bearer <your_token_here>
  - ✓ A custom authentication middleware is used:
  - ✓ Verifies the JWT token
  - ✓ Decodes user info and attaches it to the request
  - ✓ Blocks unauthorized access with appropriate error message.

- Security Methods Used

bcrypt – For password hashing and salting.

jsonwebtoken – For token creation and verification.

Middleware – To protect routes based on token presence and role.

Environment Variables – Secrets like JWT keys are stored in .env.

User Interface

User Interface – GIF Previews

➢ Below are short GIF demonstrations highlighting the main features and functionality of the HouseHunt user interface. These GIFs help visualize how users interact with the application in real-time.

1. Home Page Navigation

Description: Demonstrates the initial landing page with featured properties and navigation bar.
GIF Preview:
📁 gifs/home_navigation.gif

2. Property Search and Filters

Description: Shows how users can search for rental homes using filters like location, rent, bedrooms, and property type.
GIF Preview:
📁 gifs/search_filters.gif

3. Viewing Property Details

Description: Clicking a property opens a detailed view with images, description, and contact info.
GIF Preview:
📁 gifs/property_details.gif

4.Login and Registration

Description: Demonstrates the process of signing in or registering as a new user.

GIF Preview:

📁 gifs/login_register.gif

5. Add New Property (Owner Panel)

- Description: Property owner fills a form to add a new rental listing.
- GIF Preview:
- 📁 gifs/add_property.gif

Mobile Responsive View (Optional)

- Description: Shows how the application adapts to mobile screens.
- GIF Preview:
- 📁 gifs/mobile_responsive.gif

Testing

➢ The HouseHunt project underwent a structured testing process to ensure that all features work correctly and reliably across different use cases. The testing strategy included both manual and automated methods to validate the frontend and backend components of the application.

- Testing Strategy

The testing process was divided into three main phases:

1. Functional Testing:

Verified that all application features perform as expected. Covered user registration, login, property search, property posting, and API responses.

2. UI/UX Testing:

Ensured that the user interface is responsive and accessible across different screen sizes (desktop, tablet, mobile). Validated layout consistency and navigation flow.

3. Integration Testing:

Checked the seamless interaction between frontend and backend.
Ensured that data flows correctly between the UI, API, and data bases.

- Tools Used

Tool Purpose
Postman
To test backend REST APIs (CRUD operations, authentication)
Google Chrome Dev Tools

To check UI responsiveness, console errors, and network requests

MongoDB Compass

To visually verify data being stored and updated in the database

Visual Studio Code

For code debugging and running the development servers

React Developer Tools

For inspecting React component structure and state

Known Issues

- Despite thorough development and testing, the HouseHunt project currently has a few known limitations and bugs that users and developers should be aware of. These issues will be addressed in future updates or releases.

1. No Image Upload to Cloud

Issue: Images added during property posting are stored locally or as placeholder paths.
Impact: Limits real-time image viewing unless configured manually.
Planned Fix: Integrate image upload using a cloud storage service (e.g., Cloudinary or Firebase).

2. No Pagination on Listings

Issue: All properties are fetched and displayed on a single page.
Impact: Could slow down performance as the number of listings increases.
Planned Fix: Implement pagination or infinite scrolling for better performance.

3. Basic Role Management

Issue: Current role handling is minimal (e.g., user vs. owner) and lacks admin control.
Impact: No moderation or approval for inappropriate listings.
Planned Fix: Add admin panel for managing users and properties.

4. Mobile Optimization Incomplete

Issue: Some components may appear misaligned on smaller screen sizes.

Impact: Affects usability on mobile devices.
Planned Fix: Improve responsive layout with media queries and testing.

5. No Email Verification on Registration

Issue: New users are registered immediately without email confirmation.
Impact: May allow spam or fake registrations.
Planned Fix: Integrate email verification flow using a service like Node mailer.

Future Enhancements

- The HouseHunt project is designed with scalability in mind. While the current version covers essential features for rental property listings, there are several potential improvements and advanced features that can be implemented to enhance functionality, performance, and user experience.

1. Image Upload to Cloud Storage

Feature: Integrate cloud-based image uploads using services like Cloudinary, Firebase Storage, or Amazon S3.
Benefit: Enables users to upload and manage property images in real-time with secure and scalable storage.

2. Admin Dashboard

Feature: Add a role-based admin panel for managing users, approving listings, and removing inappropriate content.

Benefit: Improves platform control, security, and content moderation.

3. Chat System Between Owners and Tenants

Feature: Implement real-time messaging or inquiry system between users and property owners.
Benefit: Enhances communication and reduces delays in booking or property inquiries.

4. Mobile App Version

Feature: Develop a mobile application using React Native or Flutter.
Benefit: Expands reach and provides a better experience for users on smartphones and tablets.

5. Map Integration

Feature: Integrate Google Maps API to display property locations visually.
Benefit: Helps users identify property proximity to schools, workplaces, and public transport.