

Project Title

HouseHunt : Finding Your Perfect Rental Home

Team Members:

Here List team members and their roles:

1. **P. DEVI SRAVANTHI (FRONTEND)**: Responsible for designing the user interface using React.js ensuring smooth communication. This role also handles bug fixing, feature integration, and overall system performance.
2. **S. MAHESH (BACKEND)**: Responsible for designing the user interface using Node.js and Express.js. This role focuses on ensuring a responsive, user-friendly design, as well as integrating the frontend with backend APIs.
3. **R. ANVESH (TESTING)**: Develops in ensuring the creation of secure, scalable RESTful APIs, as well as handling authentication, data processing, and business logic.
4. **S. PAVAN KUMAR (DATABASE)**: Manages the MongoDB database, focusing on schema design, data integrity, and database optimization to ensure efficient data storage and retrieval.

Introduction:

In today's fast-paced world, finding the perfect rental home can be a daunting task. The process of browsing through countless listings, filtering options, and contacting landlords can be overwhelming. That's where **HOUSEHUNT** comes in—a streamlined, user-friendly platform designed to simplify the rental home search process. Powered by the **MERN stack** (MongoDB, Express.js, React, and Node.js) brings together cutting-edge web technologies to provide an intuitive experience for both renters and property own

Project Overview

PURPOSE

HouseHunt is a comprehensive web platform designed to streamline the process of searching for and listing rental homes. Built using the **MERN stack** (MongoDB, Express.js, React, and Node.js), the platform offers an intuitive and user-friendly experience for both renters and property owners. The goal of the project is to create a seamless, efficient, and scalable platform that allows users to search, filter, and view rental listings, while also enabling landlords to list their properties with ease.

Features:

1.Registration and Login: Renters and landlords can create an account by providing their basic information (email, password) and logging in securely.

2.User Profiles: Renters can save their preferences, search history, and favourite properties.

3.Property Listing: Each listing includes comprehensive information, such as: Title, location, price, Number of bedrooms

4.Add/Edit/Delete Listings: Landlords can easily add new rental properties with all required details and images.

5.Responsive Design: Fully mobile-friendly interface for browsing on any device.

Technical Architecture:

The technical architecture of our HouseHunt follows a client-server model, with a REST API used for the initial client-server connection. The frontend serves as the client and incorporates for establishing real-time communication with the backend server. The backend utilizes socket.io and Express.js frameworks to handle server-side logic and facilitate real-time messaging, post uploading and more.

The frontend includes the user interface and presentation layer, as well as the socket.io-client for establishing a persistent to connection with the server. This enables real-time bidirectional communication, allowing for instant updates and seamless interaction between users.

Authentication is handled through the REST API, which securely verifies user credentials and provides access tokens or session cookies for subsequent requests. Once authenticated, the client establishes a connection with the backend to enable real-time features.

Real-time messaging is facilitated through the library, enabling instant exchange of messages between users. Users can engage in chats with there friends, sharing text, emojis, images, etc., in real-time.

Uploading posts is supported through the connection. Users can create and upload posts including text, images, videos, or a combination of media. The server receives and processes these uploads, ensuring they are associated with the correct user and made available for other users to view and interact with in real-time.

The backend utilizes Express.js to handle the REST API endpoints, routing requests to the appropriate controllers and services. User data, including profiles, posts, and other relevant

information, is stored and retrieved from a database such as MongoDB, ensuring efficient storage and retrieval of data.

Together, the frontend, backend, REST API, socket.io, Express.js, and database (e.g., MongoDB) form a comprehensive technical architecture for our social media app. This architecture enables real-time messaging, seamless post uploading, authentication, and secure data storage, providing users with a dynamic and interactive social media experience.

4.Setup Instructions

PRE-REQUISITES:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js:

- ✓ **Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

- ✓ **Express.js:**

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture. Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware.

Installation: Open your command prompt or terminal and run the following command:

```
npm install
```

- ✓ **MongoDB:**

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

- ✓ **React.js:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Installation:

- Open your command prompt or terminal of server and run the following command: **npm install**
- ✓ **HTML, CSS, and JavaScript:** Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.
- ✓ **Database Connectivity:** Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. To Connect the Database with Node JS go through the below provided link:
- ✓ **Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.
- ✓ **Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code.

To run the existing HouseHunt App project downloaded from google drive:

Install Dependencies:

- 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 run dev
```

- The video conference app will be accessible at <http://localhost:3000>

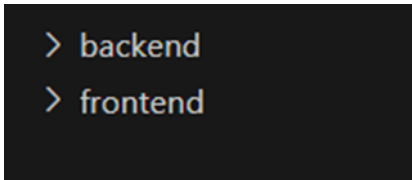
✓ **Access the App:**

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the login page, indicating that the installation and setup were successful.

You have successfully installed and set up the HouseHunt app on your local machine. You can now proceed with further customization, development, and testing as needed.

Project structure:

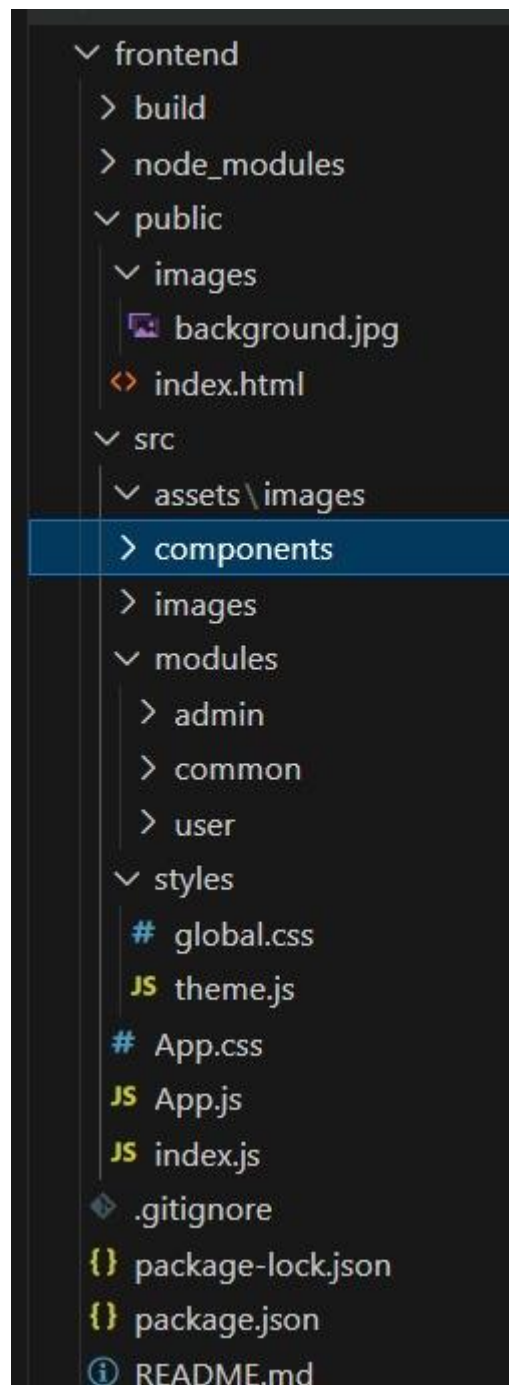
- ✓ Inside the HouseHunt directory, we have the following folders



```
> backend  
> frontend
```

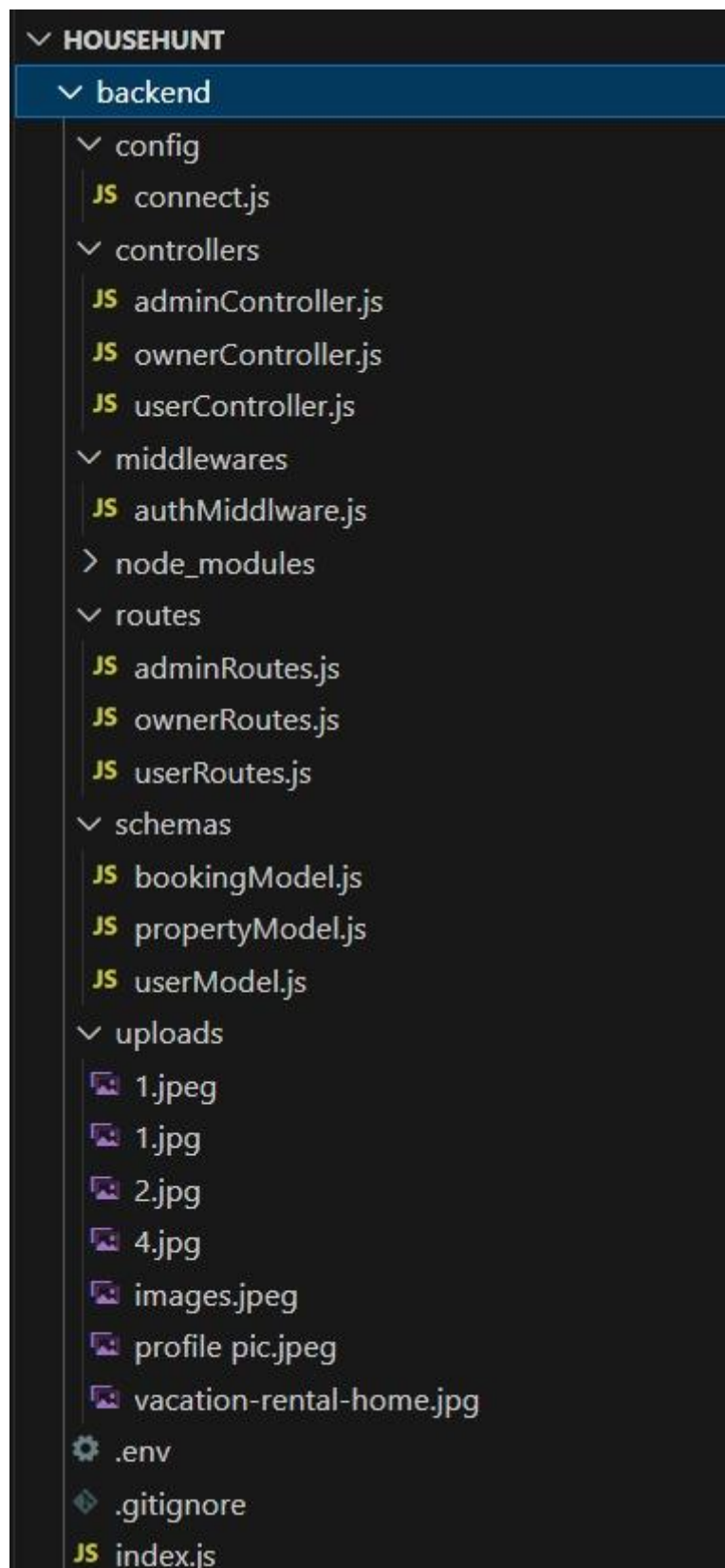
✓ **Frontend directory:**

The below directory structure represents the directories and files in the client folder (front end) where, react Js is used.



✓ Backend directory:

The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with socket.io.



Project Flow:

Project demo:

Before starting to work on this project, let's see the demo.

Demo link : <https://drive.google.com/file/d/1WLQlgdINVjAJh8HpA-ClJp41JSON934F/view>

Milestone 1: Project setup and configuration.

✓ Folder setup:

Now, firstly create the folders for frontend and backend to write the respective code and install the essential libraries.

- Frontend folders.
- Backend folders

✓ Installation of required tools:

Open the frontend folder to install necessary tools. For frontend (client), we use host:

```
C:\Users\ajaya\OneDrive\Desktop\HouseHunt>cd frontend
```

```
C:\Users\ajaya\OneDrive\Desktop\HouseHunt\frontend> npm start  
Compiled successfully!
```

You can now view **frontend** in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.43.38:3000

Note that the development build is not optimized.
To create a production build, use `npm run build`.

```
webpack compiled successfully
```


Open the backend folder to install necessary tools. For backend (server), we use:

Milestone 2: Backend development

- **Set Up Project Structure:**

- Create a new directory for your project and set up a package.json file using npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.

- **Create Express.js Server:**

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

```
const express = require("express");
const authMiddleware = require("../middlewares/authMiddleware");
const multer = require("multer");

const {
  registerController,
  loginController,
  forgotPasswordController,
  authController,
  (alias) const getAllBookingsController: (req: any, res: any) => Promise<any>
  import getAllBookingsController
  getAllBookingsController,
  updateProfileController,
  deleteAccountController,
  updateBookingStatusController,
} = require("../controllers/userController");

const router = express.Router();

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "./uploads/");
  },
  filename: function (req, file, cb) {
    cb(null, file.originalname);
  },
});

const upload = multer({ storage: storage });

router.post("/register", registerController);
router.post("/login", loginController);
router.post("/forgotpassword", forgotPasswordController);
router.get('/getAllProperties', getAllPropertiesController)
```

○

- **Implement Data Models:**

- Define Mongoose schemas for the different data entities like posts, chats, users, etc.,
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operation

```
const jwt = require("jsonwebtoken");

module.exports = async (req, res, next) => {
  try {
    const authorizationHeader = req.headers["authorization"];
    if (!authorizationHeader) {
      return res
        .status(401)
        .send({ message: "Authorization header missing", success: false });
    }

    const token = req.headers["authorization"].split(" ")[1];
    jwt.verify(token, process.env.JWT_KEY, (err, decode) => {
      if (err) {
        return res
          .status(200)
          .send({ message: "Token is not valid", success: false });
      } else {
        req.body.userId = decode.id;
        next();
      }
    });
  } catch (error) {
    console.error(error); // Handle or log the error appropriately
    res.status(500).send({ message: "Internal server error", success: false });
  }
};
```

```
C:\Users\ajaya\OneDrive\Desktop\HouseHunt\backend>npm start
```

```
> backend@1.0.0 start
```

```
> nodemon index
```

```
[nodemon] 3.0.1
```

```
[nodemon] to restart at any time, enter `rs`
```

```
[nodemon] watching path(s): *.*
```

```
[nodemon] watching extensions: js,mjs,cjs,json
```

```
[nodemon] starting `node index.js`
```

```
Server is running on port 8001
```

```
Connected to MongoDB
```

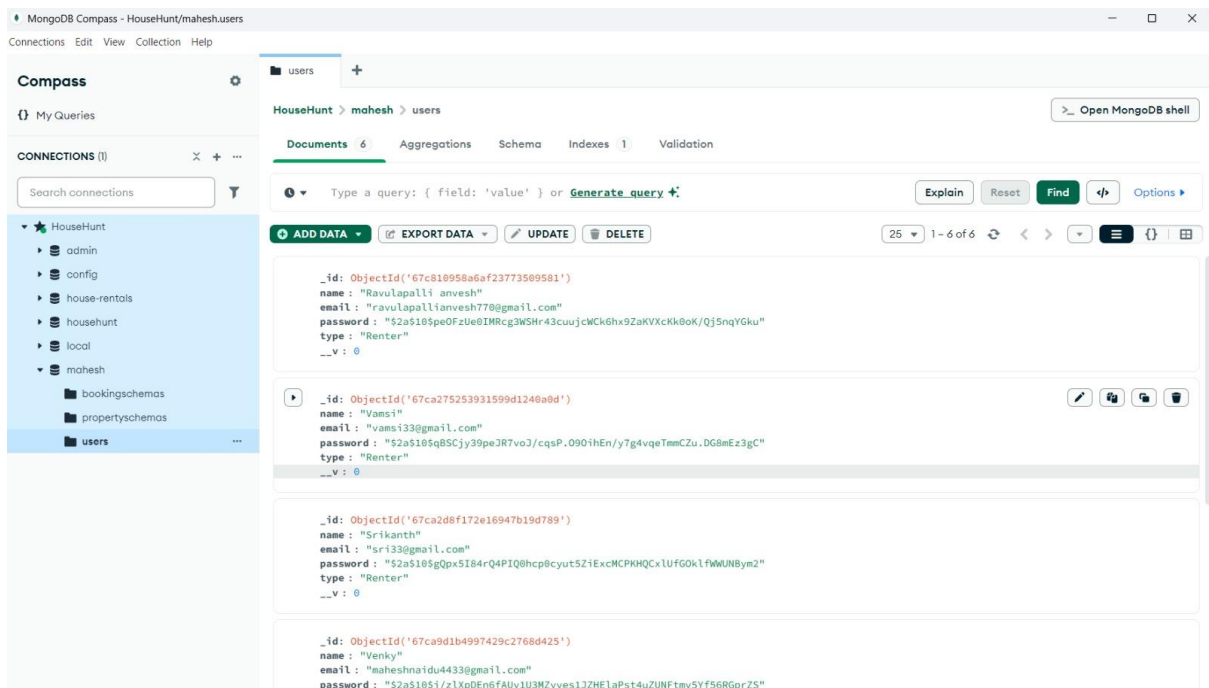
○

Milestone 3: Database development:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas.
- Create a database and define the necessary collections for users, posts, chats, etc.,

The code for the database connection to the server is

The code for the schemas in the database is provided below



```

_id: ObjectId('67c810958a6af23773509581')
name: "Ravulapalli anvesh"
email: "ravulapallianvesh770@gmail.com"
password: "$2a$10$peOFzUe0IMRcg3WSHr43cuujcWck6hx9ZaKVXcKk0oK/Qj5nqYGku"
type: "Renter"
__v: 0

```

Milestone 4: Frontend development & Integration

1. Home page of Our App

. Having login and registration icons

- **Create login and registration pages**

1. Firstly go to registration sign and register for our app.
2. And then go for login .
3. Then search for your perfect rental Home.

- **Add login details**

1. Allow the user to create a account.
2. Upload the media file to firebase storage or any other cloud platform and store the data and file link in the MongoDB.
3. Retrieve the posts and display to all the users.

- **Add sight properties**

1. Add a profile page for every individual user.
2. Add properties for search by user.

- **Add Property details**

- 1.House for Rent/Sale.
2. Location.
- 3.Type either it was flat/House.
- 4.Price.
5. Contact.
6. Status.

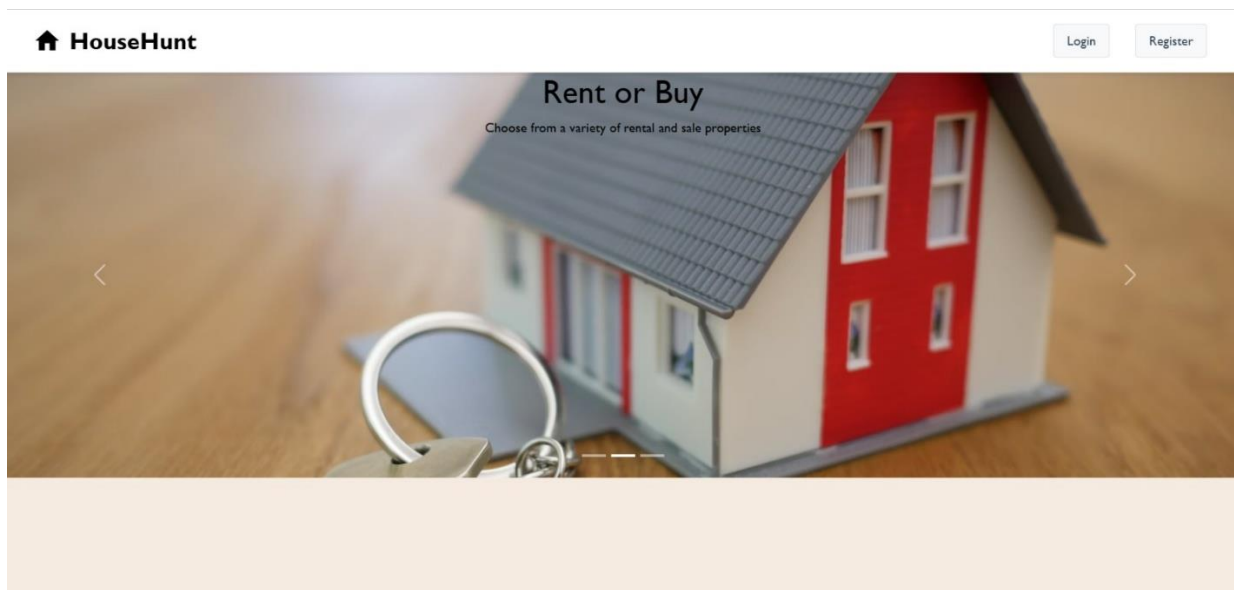
- **Accept/Reject the Booking**

1. Booking accept if there was allowing bookings.
2. Booking reject when all bookings were filled.

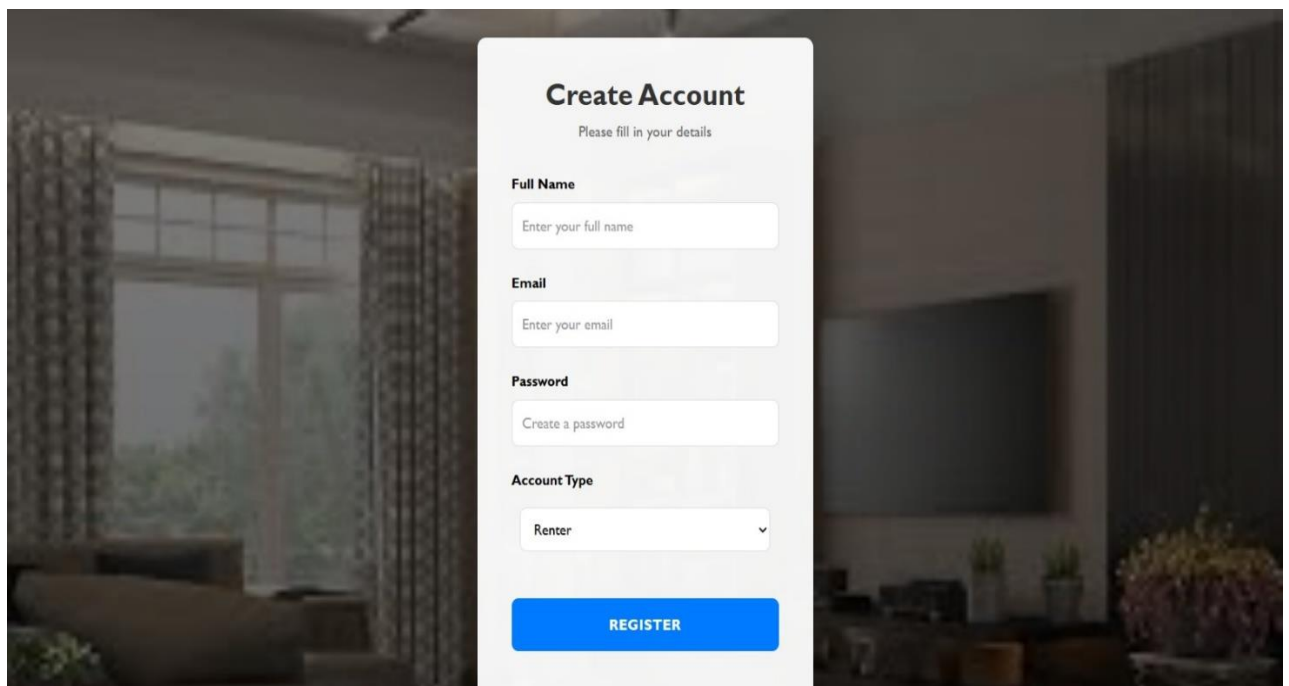
Milestone 5: Project Implementation:

On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The user interface of the application looks a bit like the one's provided below.

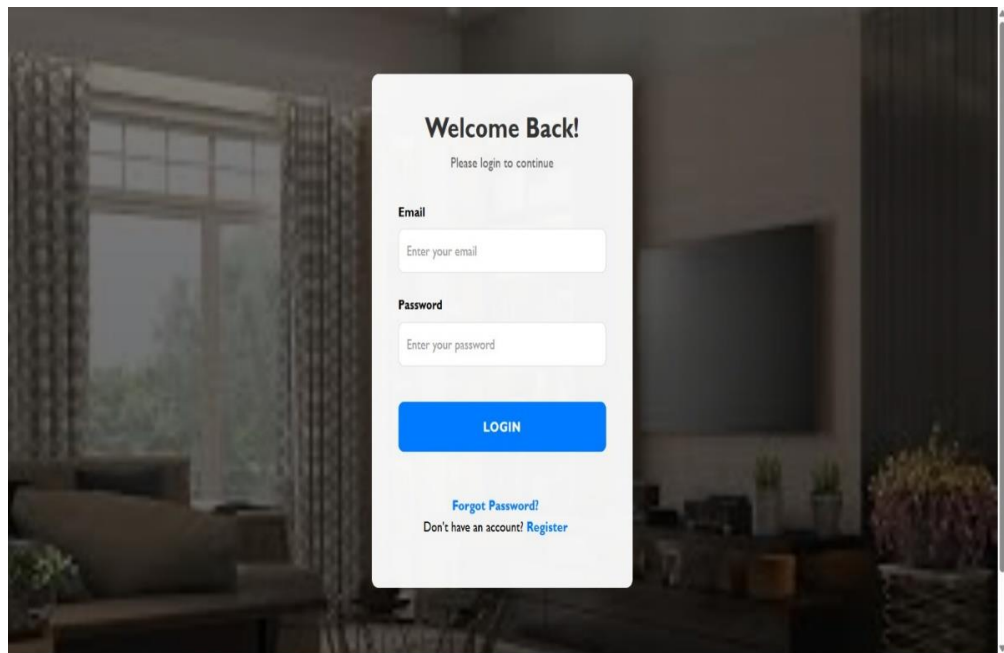
- Home page



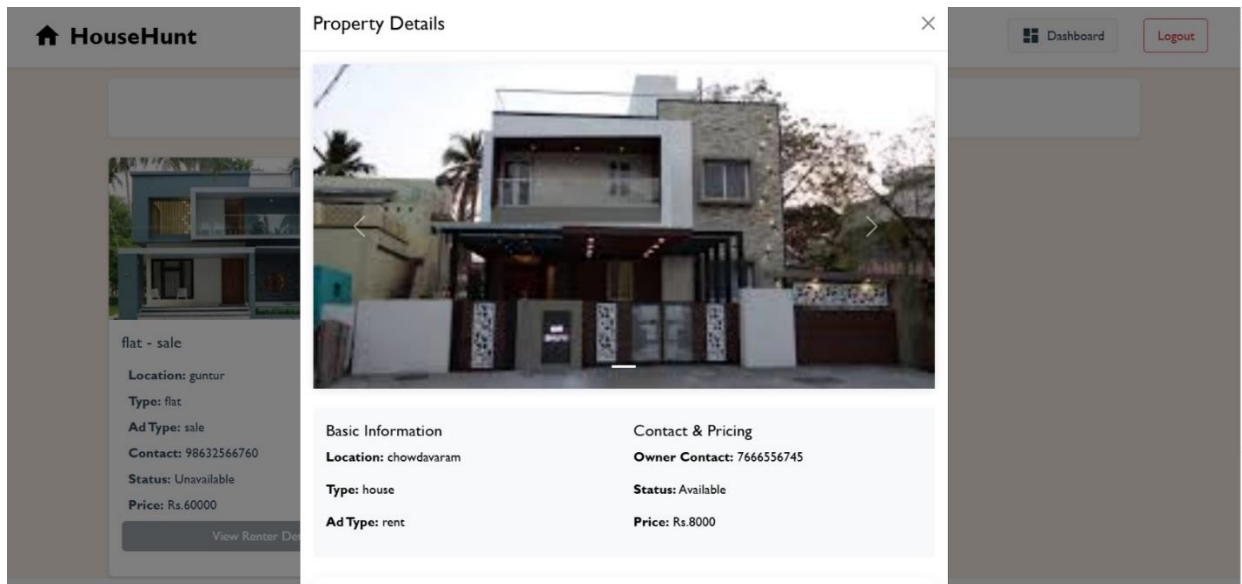
- Creating account



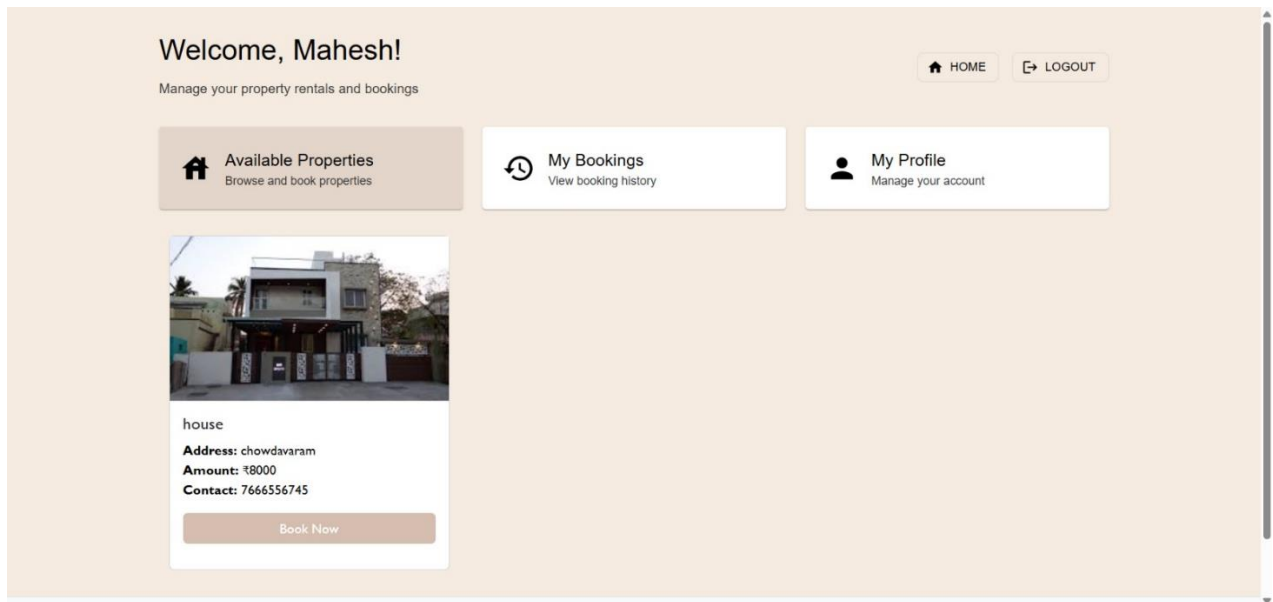
. login page:



. Properties list



. Booking sight



Testing:

1. Unit Testing:

- **Description:** Focuses on testing individual components or functions in isolation to ensure they work as intended.
- **Example:** Testing a React component to verify that it renders correctly based on props.

2.Integration Testing:

- **Description:** Tests how different modules or components work together, ensuring that integrated parts of the application function correctly.
- **Example:** Testing an API endpoint to verify that it correctly interacts with the database and returns the expected response.

Known Issues:

1. **Session Management:** Users may experience session timeouts or loss of state when navigating between pages or after refreshing.
2. **Performance Lag:** The application can slow down with a large number of posts or user interactions, particularly due to unoptimized database queries.
3. **Error Handling:** Generic error messages for failed API calls can confuse users; more specific messages are needed.

Future Enhancements:

- **Enhanced User Profiles:** Add more customizable profile options, including themes, cover photos, and bio sections.
- **User Analytics Dashboard:** Provide users with insights into their post engagement and follower statistics

Finally, for any further assistance, use the links below:

Demo link: <https://drive.google.com/file/d/1WLQlgdINVjAJh8HpA-ClJp41JSoN934F/view>