Full Stack Development with MERN

1. Introduction

- **Project Title:** House Hunt
- **Team Members:**

1. Setti Mahitha
2. Shaik saida
3. Sayyad mobeena
4. Sayani harini

Everyone is involved in all the phases of development.

2. Project Overview

- **Purpose:**

With its comprehensive and user-friendly platform, House Hunt seeks to transform the rental and home-buying industries by bringing together buyers, tenants, and homeowners. Simplifying and streamlining the process of locating and acquiring rental properties or homes for purchase is the main objective. House Hunt provides a smooth and effective experience for both consumers looking for their future home and property owners wishing to sell or fill vacancies by utilizing contemporary web technology.

- **Features:**
  1. **User Registration:** User registration in the HouseHunt application is a seamless process that allows property owners to create an account, upload pictures of their properties, and provide detailed descriptions.
  2. **User Confirmation:** User confirmation in the HouseHunt application ensures that property owners and renters are verified, enhancing trust and security. Upon registration, users receive a confirmation email to verify their account.
  3. **Profile Management:** Profile management in the HouseHunt application allows users to easily update and personalize their profiles. Property owners can edit their listings, upload new photos, and update availability. Renters can manage their booking history, save favorite properties, and adjust personal information.
  4. **Browsing and searching:** Browsing and searching in the HouseHunt application are designed for user convenience and efficiency. Users can explore a wide range of properties using advanced search filters, including location, price, amenities, and availability. The intuitive interface allows for easy navigation and quick access to property details, photos, and reviews.
  5. **Communication:** D Every home has a detailed view with pictures, details, and contact details for the owners. This facilitates the scheduling of house visits and other activities, which benefits both sides
  6. **Property Listing:** Property listing in the HouseHunt application enables property owners to showcase their rentals effectively. Owners can create detailed listings

with descriptions, photos, pricing, and availability. The intuitive interface guides them through the process, ensuring all necessary information is included.

3. Architecture

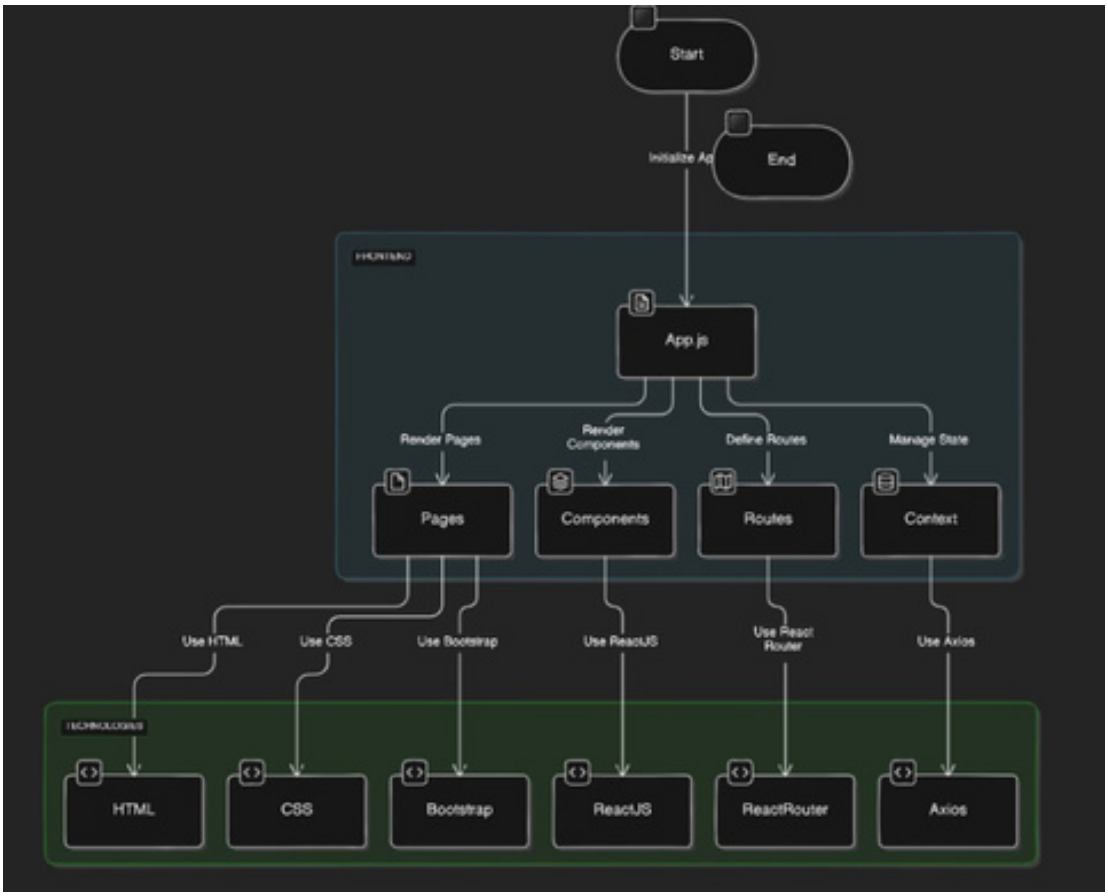
Frontend: For the House Hunt project, we have a modular, scalable, and maintainable frontend architecture. React js, html, React Router, css, Axios and bootstarp are used to create a dynamic and responsive user interface.

Overview:

1. React.js: Manages the dynamic user interface and handles state management for an interactive user experience.
2. HTML: Structures the content and forms the backbone of the application's web pages.
3. React Router: Handles routing.
4. CSS: Styles the elements to create a visually appealing and consistent look across the application.
5. Axios: For HTTP requests.
6. Bootstrap: Provides responsive design and pre-built components to ensure the application is mobile-friendly and easy to navigate.React-Leaflet: Interactive maps.

Key Components :

1. App.js: Main application layout and routing.
2. components: Reusable UI components.
3. pages: Major views like Home, Profile, and List.
4. routes: Routing configurations.
5. context: Global state management (authentication, notifications).
6. lib: Utility functions, loaders, and API request



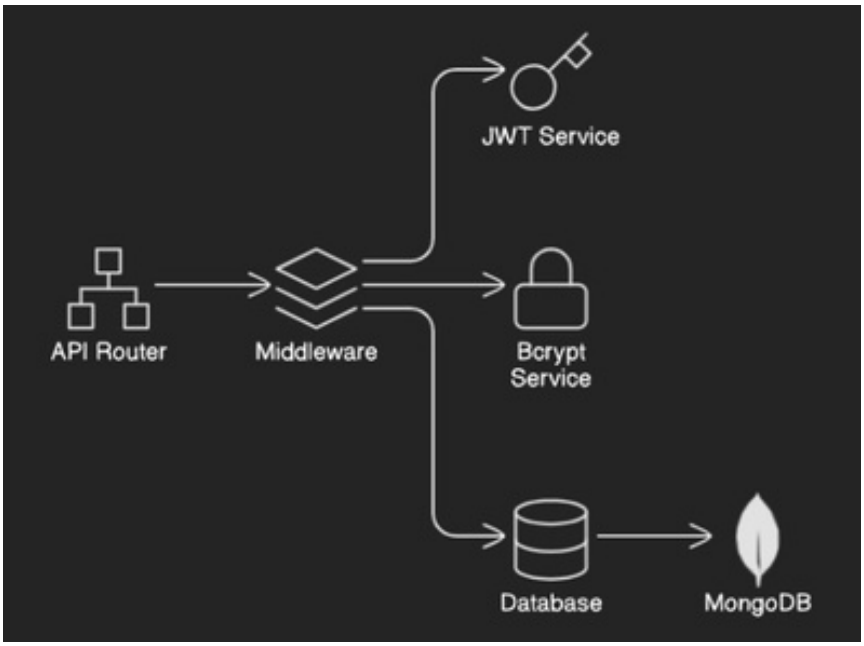
Backend: Our backend architecture for the Rent Ease project is designed to be modular, scalable, and maintainable. It uses Node.js, Express.js, and Prisma to interact with a MongoDB database.

Overview :

1. Server-side JavaScript activities can be done using Node.js.
2. The web framework Express.js is used to create RESTful APIs.
3. Application data can be stored in a NoSQL database called MongoDB.
4. Prisma: an ORM that ensures type-safe database operations while working with MongoDB.
5. JWT: Token-based authentication that is safe.
6. Bcrypt: For hashing passwords.

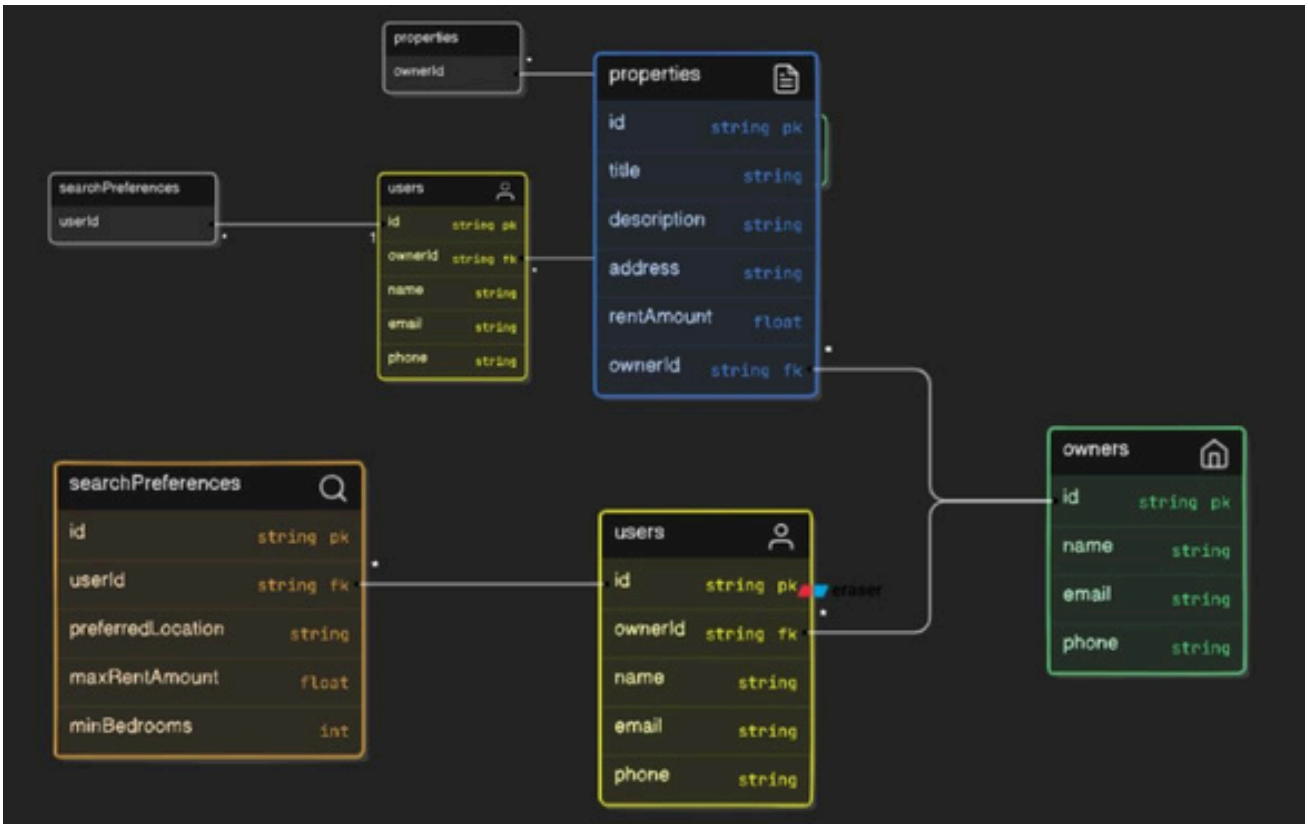
Key Components :

1. API Routes: Define endpoints and link to controllers.
2. Controllers: Handle request processing and response generation.
3. Middlewares: Custom middleware functions for authentication, error handling, and validation. 4. Database Interaction: Prisma for type-safe and efficient database operations.
4. JWT: Token-based authentication that is safe.
5. Bcrypt: For hashing passwords.



Database: MongoDB and Prisma ORM are used in the House Hunt project's database architecture to provide effective and type-safe data management. The following significant collections are part of the architecture:

1. Users Collection: Maintains relationships and authentication while preserving user data.
2. Posts Collection: Provides information on properties that users have listed, such as the details of the properties and the user connections.
3. PostDetails Collection: Contains further details on the property, like utilities, descriptions, and its closeness to amenities.
4. SavedPosts Collection: Keeps track of user-saved posts and provides links to the saved listings.



4. Setup Instructions

- **Prerequisites:**

1. Node.js
2. MongoDB
3. Git
4. React-app

- **Installation:**
  1. Clone the repository:

```
https://github.com/Harish-08726/HouseHunt-Finding-your-perfect-rental-home
```

1. Install dependencies for client , server and socket

```
cd frontend

npm install


cd backend  npm install


node index.js
```

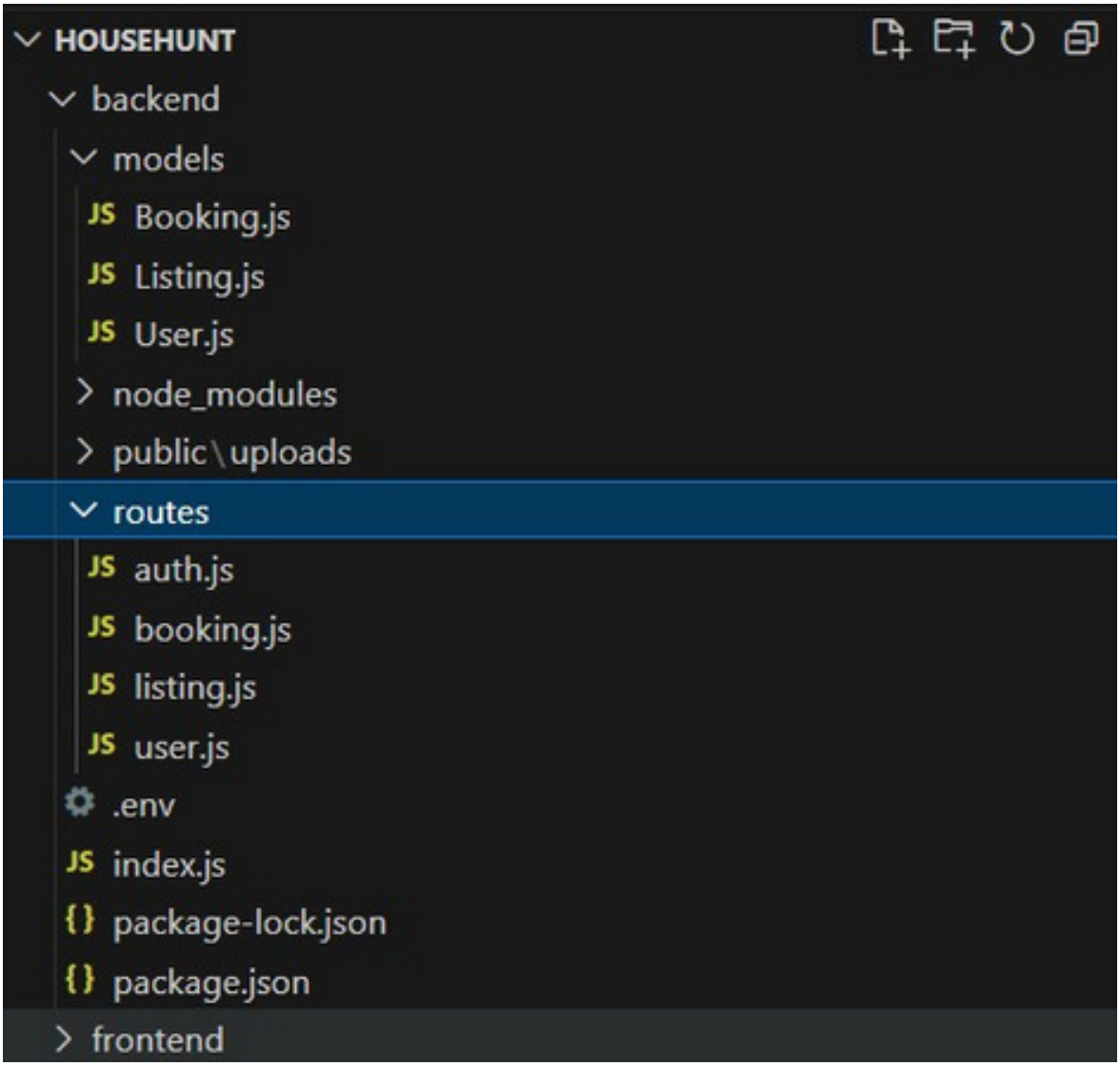
3. Set up environment variables:

Create a .env file in the server directory (api) with following environment variables

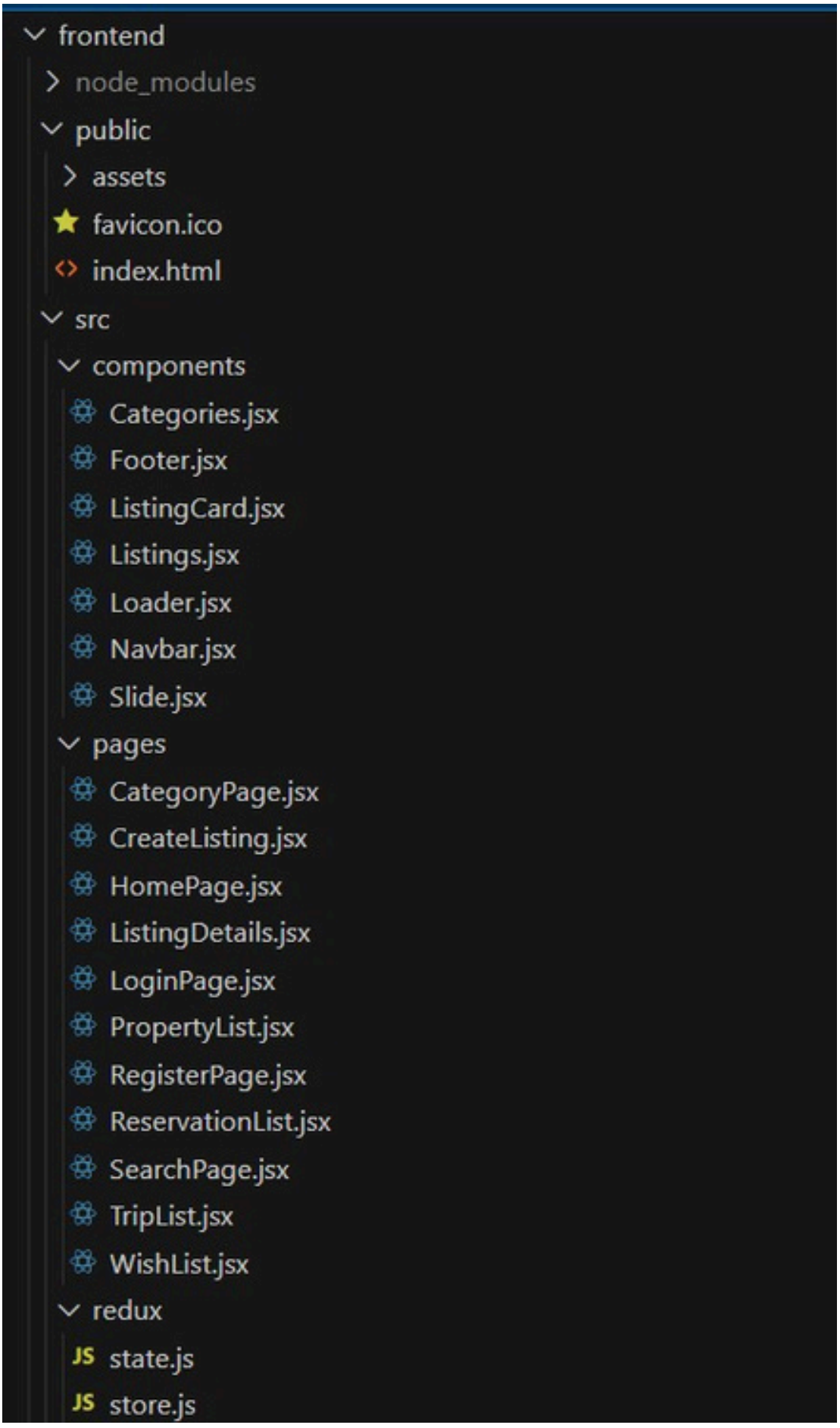
```
MONGO_DB=''mongodb://127.0.0.1:27017/'

JWT_KEY =  '12345'
```

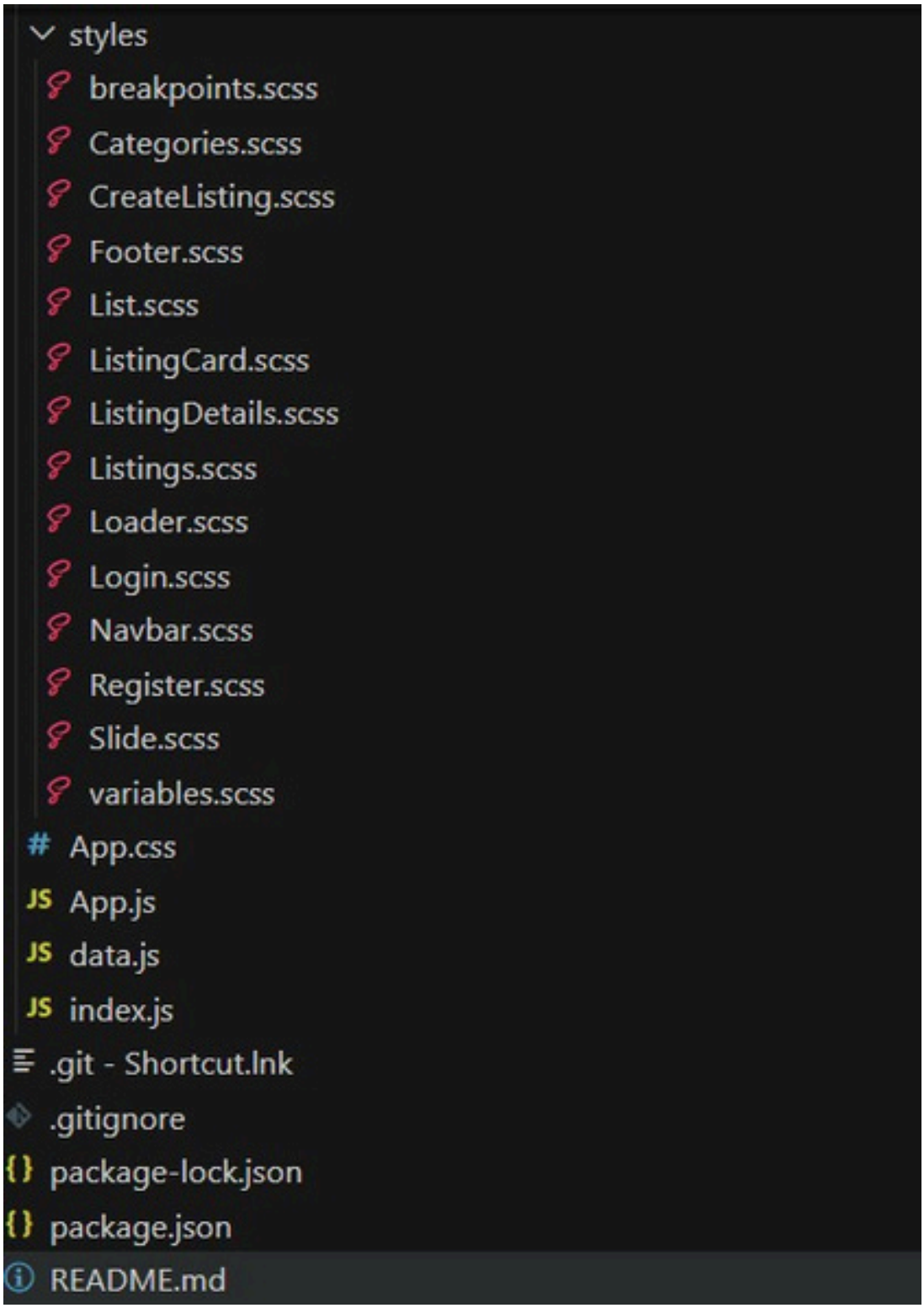
5. Folder Structure

- **backend**



- **Front end:**


frontend
node_modules
public
assets
favicon.ico
index.html
src
components
Categories.jsx
Footer.jsx
ListingCard.jsx
Listings.jsx
Loader.jsx
Navbar.jsx
Slide.jsx
pages
CategoryPage.jsx
CreateListing.jsx
HomePage.jsx
ListingDetails.jsx
LoginPage.jsx
PropertyList.jsx
RegisterPage.jsx
ReservationList.jsx
SearchPage.jsx
TripList.jsx
WishList.jsx
redux
state.js
store.js



6. Running the Application • Commands to start the frontend and backend servers locally.

○ **Frontend:** npm run dev in the client directory. ○ **Backend:** nodemon –env-file .env app.js in the api directory. ○ **Socket:** nodemon app.js in the socket directory.

7. API Documentation

1. User Authentication • **POST /api/auth/register**

○ Description: Registers a new user. ○ Parameters:

▪ username (string, required)

▪ email (string, required)

▪ password (string, required) ○ Example Response:

{

"message": "User registered successfully" }

- **POST /api/auth/login**

  - Description: Authenticates a user and returns a token.
  - Parameters:

▪ email (string, required)

▪ password (string, required) ○ Example Response:

{

"token": "JWT_TOKEN_HERE"

}

- **POST /api/auth/logout**

  - Description: Logs out the user.

Example Response:

{

"message": "User logged out successfully"

}

1. User Management • **GET /api/users**

○ Description: Retrieves all users.

Example Response:

```
[
 {
  "id": "USER_ID",
  "username": "USERNAME",
  "email": "EMAIL"
 }
]
```


- **PUT /api/users/**

  - Description: Updates user information by ID.
  - Parameters:

▪ username (string, optional) ▪ email (string, optional) ○ Example Response:


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


- **DELETE /api/users/**

  - Description: Deletes a user by ID.

Example Response:


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


- **POST /api/users/save**

  - Description: Saves a post for the user. ○ Parameters: postId (string, required)

Example Response:


```
{
 "message": "Post saved successfully"
}
```


- **GET /api/users/profilePosts**

- Description: Retrieves posts saved by the user.

Example Response:

```
[
 {
   "id": "POST_ID",
   "title": "Post Title",      "description": "Post Description"
 } ]
```

3. Post Management • **GET /api/posts**

○ Description: Retrieves all posts.

Example Response:

```
[
 {
                                                    "id": "POST_ID",
                                                   "title": "Post Title",
   "description": "Post Description"
 }
]
```

- **GET /api/posts/**

- Description: Retrieves a post by ID.

Example Response:

```
{
                                                    "id": "POST_ID",
  "title": "Post Title",    "description": "Post Description"
}
```

- **POST /api/posts**

- Description: Creates a new post. ○ Parameters

▪ title (string, required)

▪ description (string, required) Example Response:

```
{
  "message": "Post created successfully"
}
```

- **PUT /api/posts/**

  - Description: Updates a post by ID. ○ Parameters:

▪ title (string, optional)

▪ description (string, optional) Example Response:

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

- **DELETE /api/posts/**

  - Description: Deletes a post by ID.

Example Response:

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

8. Authentication

In our House Hunt project, authentication and authorization are handled through a secure token-based system using JSON Web Tokens (JWT). Below are the key aspects of how authentication and authorization are managed:

1.Authentication Flow

1. **User Registration:**
   - Endpoint: POST /api/auth/register
   - By entering their username, email address, and password, users can register.
   - For security purposes, the password is hashed before being entered into the database.

1. **User Login:**
   - Endpoint: POST /api/auth/login ○ Users authenticate by providing their email and password.
   - If the credentials are correct, a JWT token is generated and sent back to the client.

2. **User Logout:**

   ○ Endpoint: POST /api/auth/logout ○ Users can log out, which invalidates the token on the client side.

2.Token Handling

- **JWT Generation:** ○ A secret key is used to produce a JWT after a successful login. ○ The user's ID and expiration details are contained in the token.

Token Storage: ○ The token is kept locally, usually in cookies or local storage on the client side. ○ The token is contained in the request headers for API requests that need to be authenticated.

Token Verification:

- Middleware: verifyToken verifies the token's legitimacy. The request is approved if the token is valid; if not, an error message is returned and the request is denied.
- Middleware is used by each protected route on the backend to confirm the JWT.

3.Authorization

- **Role-Based Access Control (RBAC):**
  - The token is kept locally, usually in cookies or local storage on the client side. The token is contained in the request headers for API requests that need to be authenticated. Based on user roles, some actions are limited (e.g., owner, buyer).
  - Middleware can be used to verify user roles prior to granting access to particular routes.

Protected Routes:

- The verifyToken middleware is used by authentication-required routes to guarantee that only authorized users can access them.

4.Security Measures

- **Password Hashing:**

○ To ensure that plain text passwords are never kept in the database, user passwords are hashed using bcrypt before being entered.

- **Token Expiration:**

○ To minimize the possibility of token misuse, JWT tokens have an expiration time set to limit their validity.

- **Secure Token Storage:**

○ Tokens are kept safe from cross-site scripting (XSS) assaults in secure, http-only cookies.

- **HTTPS:**
  - To secure sensitive data in transit, make sure that HTTPS is used for encryption in all clientserver transactions.

9. User Interface:

10. Testing

Testing Strategy and Tools Used:

Our testing strategy for the House Hunt app focused on validating functionality, performance, and usability across various components. We combined manual testing by our team with automated testing using Postman. This approach ensured that both frontend and backend functionalities were thoroughly tested, achieving seamless integration and an optimal user experience.

- **Manual Testing:** Our team conducted thorough manual testing across different browsers and device sizes to ensure the House Hunt app's responsiveness and visual consistency. Key aspects tested included user registration, login/logout, house search functionality, listing details, and user interactions such as saving favorites and contacting property owners. This ensured a seamless and consistent user experience across various platforms.
- **Automated Testing with Postman:** We utilized Postman for API testing to validate backend functionalities such as CRUD operations for house listings, user authentication, and data validation. This approach helped us ensure API endpoints returned expected responses and handled edge cases effectively.

11. Screenshots or Demo

Demo Link :

https://drive.google.com/file/d/12Aw4qvwxcaaUsRTKaVOBAUiWMWwUVYEg/view?usp=drivesdk

12. Known Issues

Authentication and Authorization Issues:

- **Login Failures:** Users may experience issues logging in due to incorrect credentials or server errors.
- **Session Expiry:** Users might be logged out unexpectedly, leading to a poor user experience.

User Interface and Experience Bugs:

- **UI/UX Glitches:** Elements might not display correctly on various devices or screen sizes.
- **Loading Issues:** Users may experience long loading times or failed loads when accessing property details.

Performance and Scalability Issues:

- **Slow Performance:** The app may experience performance degradation under heavy load or with large datasets.
- **Scalability Limits:** There could be limitations in scaling the backend services to handle increased traffic.

API Integration Issues:

- **API Rate Limits:** The app might hit rate limits on external APIs, causing failures in fetching data.
- **Endpoint Errors:** There could be issues with API endpoints returning errors or not responding as expected.

13. Future Enhancements

- **Enhanced Property Search:**
- **AI-Powered Search:** Implement AI algorithms to suggest properties based on user preferences and search history.
- **Augmented Reality (AR) Tours:** Allow users to take virtual tours of properties using AR technology.
- **Advanced Filtering and Sorting Options:**
- **Map-Based Search:** Integrate a map view to allow users to search for properties based on geographic location.
- **Customizable Filters:** Enable users to create and save custom filters for more personalized searches.
- **Improved User Experience:**
- **User Reviews and Ratings:** Allow renters to leave reviews and ratings for properties and landlords.
- **Wishlist and Alerts:** Let users save favorite properties to a wishlist and set up alerts for new listings that match their criteria.
- **Security Enhancements:**
- **Identity Verification:** Implement advanced identity verification methods to ensure the authenticity of users.
- **Secure Messaging:** Enhance the in-app messaging system with end-to-end encryption for better privacy and security.
- **Localization and Global Expansion:**
- **Multi-Language Support:** Add support for multiple languages to cater to a global audience.
- **Region-Specific Features:** Customize features to suit the needs of different regions, such as local currency support and regional property laws.
- **Mobile App Enhancements:**
- **Offline Access:** Enable offline access to saved searches and listings for users in areas with limited internet connectivity.
- **Push Notifications:** Implement push notifications for new listings, price drops, and other relevant updates.