Full Stack Development with MERN

API Development and Integration Report

Date	15-07-2024
Team ID	SWTID1720064122
Project Name	House Hunt App using Mern
Maximum Marks	6 Marks

Project Title: House Hunt App using Mern
Date: 10-07-2024
Prepared by: Aitha Pranith
Subbannagari Lahari
Kuragayala Hariraj

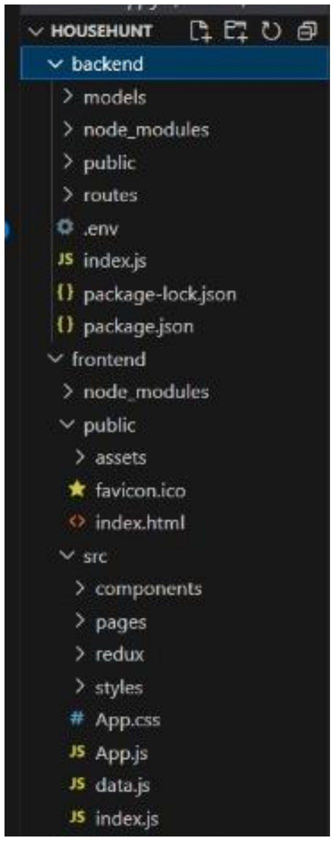
Objective

The objective of this report is to document the API development progress and key aspects of the Frontend and backend services implementation for the House Hunt project. House Hunt aims to provide a seamless platform for users to find, list, and book rental properties, leveraging modern web technologies to ensure a smooth and secure experience.

Technologies Used

- Frontend: React, Redux, Material UI
- Backend: Node.js with Express.js
- Database: MongoDB
- Authentication: JWT (JSON Web Tokens)
- Styling: SCSS

Project Structure :



```
HOUSEHUNT
  backend
    models
    node_modules
    public
    routes
    .env
    index.js
    package-lock.json
    package.json
  frontend
    node_modules
    public
      assets
      favicon.ico
      index.html
    src
      components
      pages
      redux
      styles
      App.css
      App.js
      data.js
      index.js
```

Key Directories and Files

Client Directory Structure

/src/controllers

- This directory contains functions to manage data flow and logic within the client application.
- apiController.js: Manages API requests and responses from the client to the server.
- authController.js: Handles user authentication logic and tokens.
- listingController.js: Controls interactions with property listings.
- userController.js: Manages user-related actions and data.

/src/models

- This directory contains functions to manage data flow and logic within the client application.
- ListingModel.js: Defines the structure and methods for property listing data.
- UserModel.js: Represents the user data model with methods for user actions.

/src/routes

- Defines the routes or navigation paths within the client-side application.
- index.js: Configures client-side routing using React Router, defining routes for different pages.
- PrivateRoute.js: Implements a higher-order component (HOC) to protect private routes requiring authentication.

/src/middlewares

- Custom middleware functions for handling various aspects of request processing within the client-side application.
- errorHandler.js: Middleware to handle errors gracefully within the client application.
- authMiddleware.js: Middleware for managing authentication state and tokens.

/src/config

- Configuration files and constants used throughout the client-side application.
- apiConfig.js: Configuration settings for API endpoints and constants used across the client application.
- themeConfig.js: Handles global theme settings and styling configurations using Material UI themes.

Server Directory Structure:

1. /controllers

- Contains functions to handle requests and responses.
- authController.js: Handles authentication-related requests.
- bookingController.js: Manages booking-related requests.
- listingController.js: Manages property listing-related requests.
- userController.js: Manages user-related requests.

2. /models

- Includes Mongoose schemas and models for MongoDB collections.
- Booking.js: Schema for booking details.
- Listing.js: Schema for property listings.
- User.js: Schema for user information.

3. /routes

- Defines the API endpoints and links them to controller functions.
- auth.js: Routes for user authentication (register, login).
- booking.js: Routes for managing bookings.

- listing.js: Routes for managing property listings.
- user.js: Routes for managing user data.

4. /middlewares

- Custom middleware functions for request processing.
- authMiddleware.js: Middleware for verifying JWT tokens.
- errorHandler.js: Middleware for handling errors.

5. /config

- Configuration files for database connections, environment variables, etc.
- db.js: Database connection configuration.
- keys.js: Environment variables and keys configuration.

6. /node_modules

- Contains all the npm packages required for the server-side of the application.

7. /public/uploads

- Directory for storing uploaded files like property images and user profile pictures.
- 1.jpg, 2.jpg, 3.jpeg, ...: Sample image files.

8. .env

- Configuration file for database connection URL, frontend server and JWT secret key.

9. index.js

- Entry point for the server-side application, sets up Express server and connects to MongoDB.

10. package-lock.json

- Automatically generated file that describes the exact tree of dependencies used in the project.

11. package.json

- Lists project dependencies and scripts for the server-side application.

API Endpoints

A summary of the main API endpoints and their purposes:

User Authentication

- **POST /api/user/register** - Registers a new user.
- **POST /api/user/login** - Authenticates a user and returns a token.

User Management

- **GET /api/user/:id** - Retrieves user information by ID.
- **PUT /api/user/:id** - Updates user information by ID.

Property Listings

- **GET /api/properties** - Retrieves all property listings.
- **POST /api/properties** - Creates a new property listing.
- **GET /api/properties/:id** - Retrieves details of a specific property listing.

Wish Lists

- **GET /api/wishlist** - Retrieves all items in a user's wish list.
- **POST /api/wishlist** - Adds a new item to a user's wish list.
- **DELETE /api/wishlist/:id** - Removes an item from a user's wish list.

Bookings

- **GET /api/bookings** - Retrieves all bookings.
- **POST /api/bookings** - Creates a new booking.
- **GET /api/bookings/:id** - Retrieves details of a specific booking.

Integration with Frontend

Backend-Frontend Communication

The backend of HouseHunt communicates seamlessly with the frontend via RESTful APIs. This integration ensures efficient data exchange for user interactions and content management.

- **User Authentication:** User authentication is managed using JWT (JSON Web Tokens) to securely pass authentication tokens between the frontend and backend. This ensures that users can securely log in, access personalized content, and perform authenticated actions like creating listings or managing bookings.

- **Data Fetching:** Frontend components in HouseHunt utilize RESTful API calls to fetch and manage data dynamically. For instance, property listings, user profiles, and booking details are fetched from the backend and displayed in real-time on the frontend, providing users with up-to-date information.

Error Handling and Validation
The error handling strategy and validation mechanisms:

- **Error Handling:** HouseHunt implements robust error handling mechanisms through centralized middleware. This approach ensures that all errors, whether related to database queries, API requests, or server-side operations, are efficiently captured and managed. Custom error messages and status codes provide clear feedback to users and developers alike, enhancing the application's reliability.

- **Validation:** Validating input is essential to guaranteeing the security and integrity of data. This project uses the express-validator library to implement input validation. With the help of this library, middleware functions that automatically check incoming request data against certain rules can be defined.

  For instance, the backend verifies input fields including email format, password strength, and mandatory fields before processing a request to create or change user data (POST /api/user/register, PUT /api/user/:id). Invalid data cannot enter the system because suitable error answers are delivered back to the client in the case that any validation rule fails.

Security Considerations
Security measures implemented:

- **Authentication:** HouseHunt prioritizes secure token-based authentication to safeguard user credentials and session management. JWT tokens are securely generated, transmitted over HTTPS, and validated on each authenticated request. This approach mitigates risks associated with session hijacking and unauthorized access to user accounts.

- **Data Encryption:** Sensitive data within HouseHunt, both at rest and in transit, is encrypted using industry-standard protocols and practices. This includes encrypting user passwords stored in the database using hashing algorithms like bcrypt. Additionally, all data transmitted between the frontend and backend is encrypted using HTTPS to prevent eavesdropping and data tampering.