Full Stack Development with MERN

API Development and Integration Report

Date	12-July-2024
Team ID	SWTID1720069738
Project Name	House Hunt
Maximum Marks	6 Marks

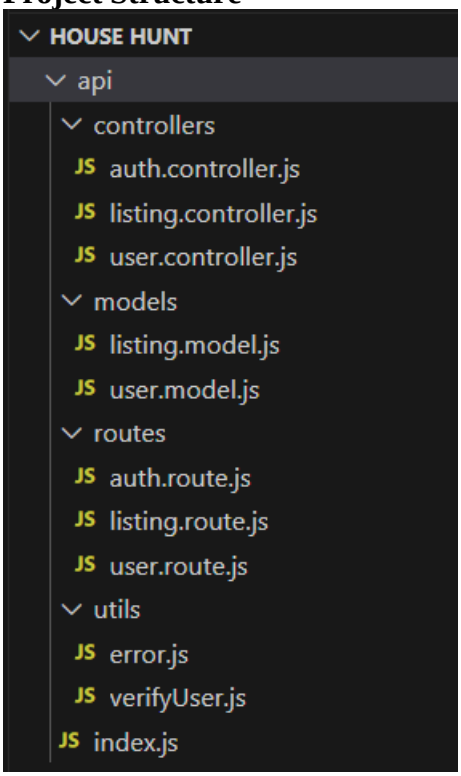
Project Title: House Hunt
Date: 12-July-2024
Prepared by: K.Rama & L.Chaitanya

Objective
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the “House Hunt” project.

Technologies Used

- **Backend Framework:** Node.js with Express.js
- **Database:** MongoDB , Mongoose
- **Authentication:** bcryptjs, JWT, cookie-parser

Project Structure



Key Directories and Files

1./controllers

Contains functions to handle requests and responses.

- **auth.controller.js**: Handles authentication-related functions such as signup, signin, and signout.
- **listing.controller.js**: Manages CRUD operations for listings including creating, deleting, updating, and fetching listings.
- **user.controller.js**: Manages user-related functions such as updating user details, deleting users, and fetching user-specific listings.

2./models

Includes Mongoose schemas and models for MongoDB collections.

- **listing.model.js**:Defines the Mongoose schema for the Listing model, including fields like name, description, address, prices, and user reference.

- **user.model.js**:Defines the Mongoose schema for the User model, including fields like username, email, password, and avatar.

3./routes

Defines the API endpoints and links them to controller functions.

- **auth.route.js**;API endpoints for authentication-related actions (signup, signin, Google authentication, signout) and links them to their respective controller functions.

- **listing.route.js**;API endpoints for listing-related actions (create, delete, update, fetch) and links them to their respective controller functions.

- **user.route.js**; API endpoints for user-related actions (test, update, delete, fetch listings) and links them to their respective controller functions.

4. /middlewares

Custom middleware functions for request processing.

- **verifyUser.js**:Custom middleware function to verify JWT tokens for user authentication and authorization.

5./config

Configuration files for database connections, environment variables, etc.

- **index.js**:Sets up the Express server, connects to MongoDB, and configures middleware such as `cookieParser` and static file serving.

API Endpoints
A summary of the main API endpoints and their purposes:

User Authentication

- **POST /api/user/signup** - Registers a new user.
- **POST /api/user/signin** - Authenticates a user and returns a token.

- **POST/api/auth/signout-**Sign out the authenticated user.

User Management

- **GET /api/user/-** Retrieves user information by ID.
- **PUT /api/user/**- Updates user information by ID.
- **DELETE /api/user/-** Deletes a user by ID.
- **GET /api/user/listings/-** Retrieves listings associated with a user by user ID.
- **GET /api/user/test/-** A test route to check if the API is working.

Integration with Frontend
The backend communicates with the frontend via RESTful APIs. Key points of integration include:

- **User Authentication:** Tokens are passed between frontend and backend to handle authentication.

1.**Backend Setup (`/api/auth/signin` and `/api/auth/signup`)**:

- Endpoints for user authentication are assumed to exist based on the frontend references to `SignIn`, `SignUp`, and the expected routes.

2.**Frontend Integration (`SignIn.jsx` and `SignUp.jsx`)**:

- The `SignIn.jsx` and `SignUp.jsx` components handle user input and make POST requests to `/api/auth/signin` and `/api/auth/signup`, respectively.
- Upon successful authentication (sign-in), the JWT token returned by the backend is stored in `localStorage`.

- **Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction.

1.**Backend Endpoints (`/api/listing/get/:listingId` and `/api/listing/update/:listingId`)**:

- These endpoints are referenced in the frontend code (`UpdateListing.jsx`, `Listing.jsx`) for fetching and updating listing details.

2.**Frontend Integration (`UpdateListing.jsx` and `Listing.jsx`)**:

- `UpdateListing.jsx` fetches existing listing details and updates them using API calls to `/api/listing/get/:listingId` and `/api/listing/update/:listingId`.
- `Listing.jsx` fetches listing details based on the `listingId` parameter from the URL (`/listing/:listingId`).

Error Handling and Validation
Describe the error handling strategy and validation mechanisms:

Error Handling:

- **Error Handling Strategy**: The backend utilizes middleware for centralized error handling. This middleware intercepts errors and sends appropriate error responses back to clients. For example, in the `error-handler.js` middleware, errors are caught, logged, and formatted before being sent as JSON responses to clients.

Validation:

Input validation using libraries like Joi or express-validator.

Validation Mechanisms:

- **User Authentication**: Handles authentication using tokens passed between frontend and backend. Tokens are validated on the backend to ensure authenticity and authorization.
- **Input Validation**: Utilizes `express-validator` middleware (`userValidator.js` and `listingValidator.js`) to validate incoming request data against defined schemas. This includes validating user registration and login inputs, as well as listing creation and update inputs. Validation rules ensure that data conforms to expected formats and constraints, enhancing security and data integrity.

Security Considerations
Outline the security measures implemented:

Authentication:

Secure token-based authentication.

Token-based Authentication: Implemented using JSON Web Tokens (JWT) for secure user authentication. Tokens are generated upon successful login and used to authenticate subsequent

API requests. The backend verifies and decodes tokens to ensure the validity and authenticity of requests.

Data Encryption:

Encrypt sensitive data at rest and in transit.

- **Encryption at Rest**: Ensures sensitive data (e.g., user passwords, session tokens) is stored in an encrypted format within the database (`userModel.js`).

- **Encryption in Transit**: Utilizes HTTPS protocol to encrypt data transmitted between the frontend and backend servers. This secures data during API calls and prevents interception by unauthorized parties.