**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 18-07-2024 |
| Team ID | SWTID1720167264c |
| Project Name | Cab Booking App |
| Maximum Marks | 10 |

**Project Title:** Cab Booking App  
**Date:** 18-07-2024  
**Prepared by:** NANDAM SAKETHRAM

**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the [Your Project Title] project.

**Technologies Used**

* **Backend Framework:** Node.js with Express.js
* **Database:** MongoDB

**Project Structure**  
Provide a screenshot of the backend project structure with explanations for key directories and files.



**Key Directories and Files:**

1. **controllers**
   * **adminControllers.js**: Contains controller function for handling admin login operation.
   * **bookingControllers.js**: Contains controller functions for handling booking-related operations, including creating, updating, and deleting bookings.
   * **cabControllers.js**: Contains controller functions for handling cab-related operations, such as adding, updating, and removing cabs.
   * **userControllers.js**: Contains controller functions for handling user-related operations, such as registration, login, and profile management.
2. **images**
   * **swift1.jpg**: An example image file that is used for testing purposes.
3. **middleware**
   * **seedAdmin.js**: Middleware script for seeding admin details into the database. This is used to initialize the database with default admin credentials (AdminID and Password) .
   * **upload.js**: Middleware for handling file uploads, using Multer to manage image or document uploads.
4. **models**
   * **admin.js**: Mongoose schema and model for the admin collection in the database, defining the structure of admin documents.
   * **booking.js**: Mongoose schema and model for the booking collection in the database, defining the structure of booking documents.
   * **cab.js**: Mongoose schema and model for the cab collection in the database, defining the structure of cab documents.
   * **user.js**: Mongoose schema and model for the user collection in the database, defining the structure of user documents.
5. **routers**
   * **adminRoutes.js**: Defines the routes for admin-related API endpoints, mapping HTTP requests to the appropriate controller functions.
   * **bookingRoutes.js**: Defines the routes for booking-related API endpoints, mapping HTTP requests to the appropriate controller functions.
   * **cabRoutes.js**: Defines the routes for cab-related API endpoints, mapping HTTP requests to the appropriate controller functions.
   * **userRoutes.js**: Defines the routes for user-related API endpoints, mapping HTTP requests to the appropriate controller functions.
6. **Root Files**
   * **.env**: Environment variables file, storing configuration settings such as port number, database connection strings and secret keys.
   * **app.js**: Main application file that sets up the Express server, middleware, and routes.
   * **package-lock.json**: Automatically generated file that locks the versions of installed npm packages.
   * **package.json**: Contains metadata about the project, including dependencies, scripts, and other configuration settings.

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

**User Authentication**

• **POST** /login - Handles user authentication.  
• **POST** /register - Registers a new user with the provided details.

**Admin Authentication**

• **POST** /adminlogin - Handles admin authentication.

**User Management**

**For Users**

• **GET** /getUser/ - Retrieves details of a specific user.  
 • **PUT** /userEdit/ - Updates the name, email, or password of a specific user.

**Cab Management**

**For Users**

• **GET** /cabs - Retrieves all cab details.  
 • **POST** /cabs - Adds a new cab with the provided form data.

**For Admins**

• **DELETE** /deleteCar/ - Deletes a specific cab by its ID.

**Booking Management**

**For Users**

• **GET** /getrides - Retrieves all ride details.  
 • **POST** /rides - Adds a new ride with the provided data.  
 • **GET** /getrides/ - Retrieves details of a specific ride.  
 • **DELETE** /deleteride/ - Deletes a specific ride by its ID.

**For Admins**

• **GET** /getAllUsers - Retrieves all user details.  
 • **DELETE** /deleteUser/ - Deletes a specific user by its ID.

**Integration with Frontend**

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

* **User Authentication**: Implemented basic user and admin authentication within the MERN stack. Upon successful login, the frontend stores the user state and authentication status to manage session persistence and secure access to protected routes.
* **Data Fetching**: Direct data fetching is employed where frontend components make API calls to retrieve and display necessary data. This includes fetching cab details, user bookings, and administrative data directly from the backend endpoints, ensuring seamless interaction and real-time data updates.

**Error Handling and Validation**

* **Error Handling**: Error handling is managed within the controllers using try-catch blocks in asynchronous methods. This approach ensures that any issues during API calls are caught and handled gracefully, providing meaningful error messages to the frontend and maintaining application stability.
* **Validation**: Input validation is handled within the application logic without using external libraries. Basic validation ensures that data integrity is maintained, and users are guided to provide correct inputs.

**Security Considerations**

* **Authentication**: Basic authentication for both users and admins has been implemented. To secure routes that require authentication, a state property such as ‘isAuthorized’ is used in React. This ensures that users or admins cannot access protected pages without proper authentication, enhancing overall security.
* **Data Encryption**: While no specific encryption mechanism is used for passwords or other sensitive data, the application handles authentication states securely. Future enhancements could include integrating encryption libraries for enhanced security of sensitive information both at rest and in transit.