Full Stack Development with MERN

API Development and Integration Report

Date	20-JULY-2024
Team ID	PNT2022TMID1720168902
Project Name	FOOD MINE- FOOD ORDERING
Maximum Marks	6 Marks

Project Title: FOOD MINE
Date: 16-JULY-2024
Prepared by: VUNNAM INDIRA AND RAGHU RAM.CH

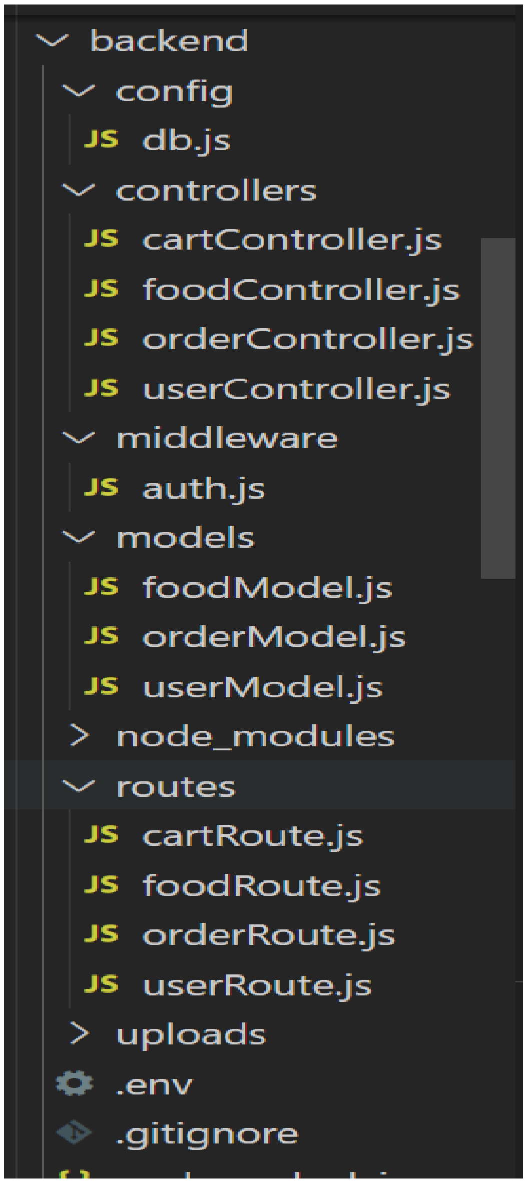
Objective

The objective of this report is to document the API development progress and key aspects of the backend services implementation for the "FOOD ORDERING – FOOD MINE" project.

Technologies Used

- **Backend Framework:** Node.js with Express.js
- **Database:** MongoDB
- **Authentication:** JWT

Project Structure


```
backend
  config
    db.js
  controllers
    cartController.js
    foodController.js
    orderController.js
    userController.js
  middleware
    auth.js
  models
    foodModel.js
    orderModel.js
    userModel.js
  node_modules
  routes
    cartRoute.js
    foodRoute.js
    orderRoute.js
    userRoute.js
  uploads
  .env
  .gitignore
```

Key Directories and Files

1. **/controllers**

   Contains functions to handle requests and responses.

   - **cartControllers.js:** Manages requests related to shopping cart operations, such as adding items, removing items, and viewing the cart.

- **foodControllers.js:** Handles requests for food-related operations, including fetching food items, adding new items, and updating or deleting existing ones.
- **orderControllers.js:** Manages order-related requests, including creating new orders, updating order status, and retrieving order details.
- **userControllers.js:** Responsible for handling user-related operations, such as user registration, login, profile updates, and password management.

2. **/models**

Includes Mongoose schemas and models for MongoDB collections.

- **foodModel.js:** Defines the schema and model for food items, including attributes like name, price, and description.
- **orderModel.js:** Contains the schema and model for orders, detailing information such as order status, user details, and item quantities.
- **userModel.js:** Specifies the schema and model for user accounts, including fields for user credentials, profile information, and roles.

3. **/routes**

Defines the API endpoints and links them to controller functions.

- **cartRoute.js:** Sets up endpoints related to cart operations and connects them to the corresponding controller functions.
- **foodRoute.js:** Defines endpoints for food-related actions and links them to the foodControllers.
- **orderRoute.js:** Establishes routes for order management and associates them with orderControllers.
- **userRouter.js:** Manages routes for user operations, connecting them to userControllers.

4. **/middlewares**

Custom middleware functions for request processing.

- **auth.js:** Implements authentication middleware to verify and authorize users based on their JWT tokens.

5. **/config**
   - **db.js:** Contains configuration settings for connecting to the MongoDB database, including connection strings and options.

API Endpoints

A summary of the main API endpoints and their purposes:

User Authentication

- **GET /api/server/register** - Registers a new user.
- **GET /api/server/login** - Authenticates a user and returns a token.
- **GET/api/server/cart-** Adding the items in the cart
-

User Management

- **POST /api/user/Register/-** Retrieves user information by ID.
- **POST/api/user/Login/**- Login user information by ID.

Cart Management:

- **GET/api/cart/-**Retrieves all cart for the authenticated user
- **POST/api/cart-**Adding to the cart
- **POST/api/cart/remove/-** Remove items from the cart

Food Management:

- **GET/api/food/list-** retrving the food list of the order
- **POST/api/food/add/-** adding the food items into cart
- **POST/api/food/remove-** Removing food from the cart.

Order Management:

- **GET/api/order/list-** Retriving the list in the order.
- **POST/api/order/userorder-** Retriving the data of orders list
- **POST/api/order/place** – Retriving the places what we order
- **POST/api/order/status –** Track the orders.

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

- **User Authentication:**

  The provided code defines an authentication middleware function for a Node.js application. It imports the `jsonwebtoken` library and uses it to verify the presence and validity of a JWT token from the request headers. If the token is missing, it responds with an error message indicating the user needs to log in again. If the token is present, it verifies the token using a secret key stored in the environment variables. Upon successful verification, it adds the user ID from the decoded token to the request body and calls the `next()` function to proceed to the next middleware or route handler. If the token verification fails, it responds with an error message.

- **Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction. These components often use functions such as fetch or libraries like axios to send HTTP requests to backend APIs. Upon receiving the data, they update the component's state, triggering a re-render to display the fetched information to the user. This process allows dynamic content to be loaded and updated without needing to refresh the entire page, enhancing the user experience by providing real-time data updates and interactions.

- **Getting Data:**
  The backend services for the "foodmine-react-js" project handle GET requests to retrieve data from the server. For example, to fetch a list of food items, the backend might have an endpoint that queries the database and returns the data in JSON format. The frontend component sends a GET request to this endpoint, and upon receiving the response, it updates the state to display the fetched food items.

- **Posting Data:**

  POST requests are used to send new data to the server. For instance, when adding a new food item to the menu, the frontend component collects the necessary information from the user and sends it to the backend via a POST request. The backend endpoint processes this data, saves it to the database, and responds with a confirmation or the newly created item.

- **BACKEND SERVICES:**

  The backend services for the "foodmine-react-js" project use RESTful APIs to manage data interactions. These services handle the CRUD (Create, Read, Update, Delete) operations through appropriately defined endpoints. The integration of these services allows the frontend to dynamically interact with the backend, ensuring a seamless user experience by enabling the creation, retrieval, updating, and deletion of data. For detailed implementation, refer to the repository

Error Handling and Validation

Describe the error handling strategy and validation mechanisms:

- **Error Handling:** Centralized error handling using middleware. The provided code defines an authentication middleware for a Node.js application, where JSON Web Tokens (JWT) are used to verify user authorization. The middleware checks for the presence of a token in the request headers and, if absent, responds with an error message indicating the need to log in again. If a token is present, it attempts to verify the token using a secret key from the environment variables. Upon successful verification, it extracts the user ID from the decoded token and attaches it to the request body before calling the next() function to proceed to the next middleware or route handler. If the token verification fails, it catches the error and responds with an appropriate error message. This approach not only handles authentication but also demonstrates centralized error handling using middleware, ensuring that all token-related errors are managed consistently.
- **Validation:** several key libraries are utilized to build and enhance the application. **React** forms the backbone of the user interface, enabling the creation of dynamic and interactive components. **Redux** is employed for state management, ensuring that the application state is predictable and centralized. **React Router** handles routing and navigation, allowing for the development of single-page applications with multiple views. **Axios** is used for making HTTP requests to communicate with backend services. **Bootstrap** provides styling and responsive design components, ensuring a polished and user-friendly interface. Additionally, **jsonwebtoken** is utilized for handling authentication with JSON Web Tokens, ensuring secure user verification and session management. These libraries collectively contribute to a robust and efficient application.

Security Considerations

Outline the security measures implemented:

- **Authentication:** Secure token-based authentication. project, security is enhanced through secure token-based authentication. This approach involves issuing JSON Web Tokens (JWT) to authenticated users, which are then used to verify their identity in subsequent requests. The tokens are signed with a secret key to prevent tampering and ensure their integrity. When a user logs in, a token is generated and sent to the client, which stores it securely (e.g., in HTTP-only cookies or local storage). Subsequent requests include this token in the headers, allowing the backend to validate the user's identity and authorize access to protected resources. This method ensures that only authenticated users can interact with the application's secure endpoints, providing a robust mechanism to safeguard against unauthorized access and maintain the overall security of the system.

- **Data Encryption:** Encrypt sensitive data at rest and in transit. data encryption is employed to protect sensitive information both at rest and in transit. **Encryption at rest** involves securing stored data, such as user credentials and other sensitive information, by using strong encryption algorithms. This ensures that even if unauthorized access to the database occurs, the encrypted data remains protected and unreadable without the appropriate decryption keys. **Encryption in transit** involves encrypting data as it is transmitted between the client and server. This is typically achieved using HTTPS, which secures communication channels with SSL/TLS certificates, preventing eavesdropping and tampering during data transfer. By implementing both types of encryption, the project ensures comprehensive protection of sensitive data against unauthorized access and data breaches, enhancing overall security and privacy.