**Food Ordering App**

**A Custom-Built Food Ordering Platform**

**MERN Stack Project**

**Swathi P**

**Karthika D**

**Lavanya R**

**Hari Nandhini M**

**13th November 2024**

**Abstract**

The Food Ordering App is a full-stack application built on the MERN (MongoDB, Express.js, React, and Node.js) framework, designed to offer a seamless online food ordering experience for customers and robust management tools for administrators. The app allows users to browse food items, add them to a cart, and securely place orders. Additionally, it provides an admin interface for managing the menu, tracking orders, and analyzing sales data. By leveraging a modern, component-based frontend with React and a scalable backend with Express and MongoDB, the application ensures a responsive user experience across devices. This document details the app’s architecture, setup, key features, and areas for future enhancement, establishing it as a foundational platform for online food ordering systems.

**Keywords**

MongoDB, Express.js, React, Node.js, Food Ordering Application, E-commerce Platform, Online Ordering System, User Authentication, Payment Gateway Integration, Admin Dashboard, Full Stack Development, Responsive Design, Scalable Architecture

**1. Introduction**

**Project Title:** Food Ordering App

**Team Members:**

* **Full Stack Developer:** Swathi P
* **Frontend Developer:** Karthika D
* **Backend Developer:** Lavanya R
* **Database Developer:** Hari Nandhini M

**2. Project Overview**

**Purpose:**  
The Food Ordering App project aims to create a scalable, feature-rich online food ordering platform where users can browse menus, add items to their cart, and place orders. It also includes an admin interface for managing food items, monitoring orders, and viewing sales data.

**Features:**

* **Customer Experience:** Intuitive navigation through categories and items, order tracking, and a secure checkout process.
* **Admin Dashboard:** Tools for managing menus, processing orders, and viewing analytics.
* **Secure Transactions:** Integrated payment gateway and JWT-based authentication for security.
* **Responsive Design:** Accessible across various devices including desktops, tablets, and mobile devices.

**3. Architecture**

**Frontend (React):**

* The frontend is built using React and organized into reusable components to ensure modularity and scalability.
* **Components:** Organized by function, such as Menu, Cart, OrderHistory, Login, and Checkout.
* **Routing and Guards:** React Router is used to manage navigation, with route guards ensuring only authorized users access specific pages.
* **UI Frameworks:** Styled with Bootstrap and custom CSS for responsive, consistent UI across all screen sizes.

**Backend (Node.js and Express.js):**

* The backend handles business logic, database interactions, and API endpoints using Express.js.
* **Modular Structure:** Organized into controllers, services, and routes for each feature (e.g., menu, order, user).
* **API Gateway:** A centralized gateway processes client requests and routes them to appropriate services.
* **Middleware:** Includes JWT-based authentication and role-based authorization.

**Database (MongoDB):**

* **Schema Design:** Utilizes Mongoose to define schemas for User, MenuItem, Order, and Cart.
  + **User Schema:** Stores user credentials, role (customer or admin), and order history.
  + **MenuItem Schema:** Includes item name, category, price, and availability.
  + **Order Schema:** Captures order details, including items, quantities, customer info, and order status.
  + **Cart Schema:** Temporarily stores items before checkout.

**4. Setup Instructions**

**Prerequisites:**

* **Software Requirements:** Node.js, MongoDB (local or cloud-hosted like MongoDB Atlas)

**Installation:**

**Installation Process for the Food Ordering App (MERN Stack)**

1. **Download the Project Files:**
   * Begin by accessing the project repository, typically hosted on a platform like GitHub, and download or clone the files to your local environment. This will provide you with the entire codebase, including both frontend (React) and backend (Node.js) components.
2. **Install Dependencies:**
   * The project dependencies are divided between the client (React frontend) and server (Node.js backend). Navigate to each of these directories separately and install the necessary libraries and packages. This step ensures that all required modules are available for both client and server environments.
3. **Set Environment Variables:**
   * Configure environment variables to manage sensitive information, such as the database connection URI, authentication secret keys, and payment gateway credentials.
   * Create environment variable files (e.g., .env) in the client and server directories. These files will store information specific to your application setup, including the backend server’s base URL for the frontend and the database connection URI and authentication keys for the backend.
4. **Start MongoDB:**
   * Ensure MongoDB is running on your system, either as a local installation or through a cloud-hosted database like MongoDB Atlas. This database will store information for the Food Ordering App, such as menu items, orders, and user data.
5. **Run the Application:**
   * Backend Server: Start the backend server to enable the API, which handles requests from the frontend. The backend typically runs on a specified port (e.g., 5000).
   * Frontend Server: Start the frontend server, which will launch the React application. The frontend typically runs on a separate port (e.g., 3000), allowing it to interact with the backend server and display the user interface.
6. **Access the Application:**
   * Open your web browser to access the frontend of the Food Ordering App. This will allow you to explore the app’s features, such as browsing menus, placing orders, and viewing order history.
   * The backend API can also be accessed via its URL, allowing for direct interaction with endpoints, which is useful for testing and development purposes.

**5. Folder Structure**

**Client (React Frontend):**

* src/components/: Reusable components for UI elements (e.g., Menu, Cart, OrderHistory, Login).
* src/pages/: Pages such as Home, Menu, Cart, Checkout, and AdminDashboard.
* src/services/: Contains API calls and utilities for interacting with backend services.
* public/: Static files like images and the root HTML file.

**Server (Node.js Backend):**

* src/controllers/: Business logic for handling requests (e.g., menuController, orderController).
* src/models/: Mongoose schemas (e.g., User.js, MenuItem.js, Order.js).
* src/routes/: API endpoints grouped by resource (e.g., menuRoutes, orderRoutes, userRoutes).
* src/middleware/: Middleware functions, such as authMiddleware.js for JWT-based authentication.

**6. Running the Application**

1. **Start MongoDB** (if using locally): mongod
2. **Run the Backend**:
   * Navigate to the server directory and run:

npm run dev

* + This starts the backend server on http://localhost:5000.

1. **Run the Frontend**:
   * Navigate to the client directory and run:

npm start

* + Access the frontend at http://localhost:3000.

**7. API Documentation**

**Authentication APIs**:

* **POST /api/auth/register**: Registers a new user.
* **POST /api/auth/login**: Logs in a user, issuing a JWT token.

**Menu APIs**:

* **GET /api/menu**: Retrieves all menu items.
* **POST /api/menu**: Adds a new item to the menu (admin only).

**Order APIs**:

* **POST /api/orders**: Creates a new order based on the cart.
* **GET /api/orders/**

: Retrieves order details by order ID.

**Example Request and Response**

Each endpoint is documented with request methods, required parameters, and example responses.

**8. Authentication and Authorization**

**JWT-Based Authentication**:

* **Token Generation**: Upon login, a JWT token is issued to the user.
* **Token Validation**: Tokens are validated with each request to ensure secure access.

**Role-Based Access Control (RBAC)**:

* **Roles**: User and Admin roles control access to features like menu management and analytics.
* **Route Protection**: Middleware restricts access to certain routes based on user roles.

**9. User Interface**

**Component-Based UI**:

* **Navigation Flow**: Intuitive navigation with categorized browsing for food items.
* **Order Flow**: Allows users to add items to the cart, view the cart, and proceed to checkout.
* **Admin Dashboard**: Provides management options for menu items and viewing order analytics.

**Screenshots**

Include screenshots of major interfaces like the menu page, cart, checkout, and admin dashboard.

**10. Testing**

**Testing Strategy**:

* **Frontend Testing**: Jest and React Testing Library are used for unit tests on components and integration tests on services.
* **Backend Testing**: Jest is used to test API endpoints and business logic in controllers.
* **End-to-End Testing**: Cypress is used for simulating user interactions from login to checkout.

**11. Screenshots or Demo**

Provide screenshots showcasing different UI features or a demo link for live testing of the app.

**12. Known Issues**

* **Token Expiration**: Users may need to re-login if their JWT expires.
* **Database Optimization**: As orders grow, query performance could be improved by indexing common fields.
* **CORS Policy**: Strict CORS policies may block requests from some non-trusted origins.

**13. Future Enhancements**

* **Push Notifications**: Notify users of order status changes.
* **Personalized Recommendations**: Recommend items based on past orders or popular choices.
* **Multi-Payment Gateway Support**: Expand payment options to include digital wallets like Apple Pay.
* **Mobile Application**: Develop a mobile app version to enhance accessibility.

**Conclusion**

The Food Ordering App is a comprehensive, secure, and scalable platform for ordering food online. This documentation outlines each component's structure and functionality, providing a guide for further development and enhancements. The app offers a seamless experience for both users and administrators, and it has room for feature expansions to enhance user engagement.