**SB Foods WebApp**

**Project Documentation**

**1. Introduction**

**Project Title:**

**SB Foods - Online Food Ordering Web Application**

**Team Members:**

* **Sowmiya Lakshmee** L– Frontend development
* **Mathiyarasi.M-** Backend development
* **Sandhiya G-** DataBase management
* **Deepa.R-** Testing and deployment

**2. Project Overview**

**Purpose:**

SBFoods is an advanced online food ordering system aimed at streamlining the process of browsing, selecting, and ordering food, especially focusing on late-night availability. The project addresses the growing demand for convenient, fast, and reliable food delivery services in the digital age.

The platform is designed to cater to both users and administrators with specific goals for each:

* **For Users**:
  + Provide an intuitive and user-friendly interface for food browsing and ordering.
  + Offer a variety of late-night food options, addressing the needs of night owls and late-shift workers.
  + Enable seamless tracking of orders from placement to delivery, enhancing the user experience.
* **For Administrators**:
  + Simplify the management of products, orders, and user accounts through a centralized dashboard.
  + Provide tools for maintaining inventory, processing orders, and monitoring user activity.

The system ensures smooth interactions between users and administrators while maintaining high security and reliability standards through robust backend and database management.

**Features of SB Foods Project**

The SB Foods platform incorporates multiple advanced features to enhance user experience and operational efficiency:

**User Registration and Login**

* Allows users to create personal accounts securely with unique credentials.
* Implements JWT (JSON Web Token) authentication for safe login and session management.
* Ensures data privacy with encrypted storage of sensitive information like passwords.

**Browse and Search for Food Items**

* Displays an extensive menu of food items with filtering options by category, price, or availability.
* Implements a search feature to allow users to find specific dishes quickly.
* Dynamic UI updates to reflect real-time changes in product availability or offers.

**Add/Remove Items to/from the Cart**

* Provides a persistent cart system where users can add or remove items.
* Displays real-time calculations for subtotal, taxes, and final amount.
* Ensures cart contents are saved across sessions for logged-in users.

**Place Orders and Track Delivery**

* Facilitates seamless order placement with a confirmation system.
* Integrates payment gateway support for secure transactions.
* Includes a delivery tracking feature that updates users on the status of their orders in real time.

**Admin Dashboard for Managing Users, Products, and Orders**

* Centralized admin interface for adding, updating, or deleting food items.
* Provides an overview of active orders, their statuses, and customer details.
* Features user management tools for monitoring customer activity and resolving account-related issues.
* Offers analytics and reporting capabilities for business insights, such as sales trends and popular items.

**3. Architecture**

**Frontend Architecture**

The frontend of the SB Foods platform is developed using **React.js**, a powerful JavaScript library for building dynamic and responsive user interfaces. React ensures seamless user interaction and a highly interactive experience by implementing a component-based architecture.

**Key Frontend Components**

1. **User Authentication**
   * Provides secure login and signup functionalities.
   * Integrates validation for input fields (e.g., email, password) to enhance security and usability.
   * Stores authentication tokens (JWT) securely using HTTP-only cookies or local storage.
   * Redirects users post-login/signup to the appropriate pages (e.g., homepage or dashboard).
2. **Cart**
   * Displays items selected by the user for purchase.
   * Updates item quantities dynamically, providing real-time price calculations (subtotal, taxes, and total).
   * Allows users to remove items or adjust quantities directly in the cart.
   * Saves the cart state for logged-in users, ensuring persistence across sessions.
3. **Products**
   * Fetches and displays the menu items from the backend API.
   * Includes sorting and filtering options (e.g., by category, price, availability).
   * Implements a search bar for users to quickly find specific dishes.
   * Displays product details like name, description, price, and availability.
4. **Profile**
   * Shows user-specific details such as name, email, and order history.
   * Allows users to update their profile information securely.
   * Displays past orders with their statuses and total amounts.
5. **Admin Dashboard**
   * Provides a centralized interface for managing the platform.
   * Allows admins to add, update, or delete food items in the menu.
   * Offers tools to monitor and manage user accounts.
   * Displays active and past orders with their statuses for order management.
   * Integrates charts and tables to provide insights into business performance, such as sales trends and popular dishes.

**Backend Architecture**

The backend is built with **Node.js** and **Express.js**, providing a robust and scalable server-side architecture. It features RESTful APIs to ensure smooth communication between the frontend and backend, while business logic is encapsulated within controllers.

**Key Backend Features**

1. **Authentication (JWT-based)**
   * Implements JSON Web Tokens (JWT) for secure authentication and authorization.
   * Includes middleware to protect sensitive routes (e.g., cart, profile, admin).
   * Features secure token storage and expiration mechanisms.
2. **CRUD Operations**
   * **Users**: APIs for creating, retrieving, updating, and deleting user profiles.
   * **Products**: APIs for managing food items, including category, price, and availability.
   * **Orders**: APIs for creating and retrieving order details, updating order statuses, and deleting canceled orders.
3. **Cart and Order Management**
   * Logic to handle cart operations, such as adding, removing, and updating items.
   * Validates product availability before adding items to the cart.
   * Ensures accurate order placement with transaction-like mechanisms to avoid errors.
   * Updates order status (e.g., pending, confirmed, delivered) based on user or admin actions.

**Database Design**

The database uses **MongoDB**, a NoSQL database known for its flexibility and scalability. The schema is modeled to handle user, product, cart, and order data effectively.

**Key Collections**

1. **Users**
   * Stores user credentials (email, password) and profile details (name, phone number).
   * Includes fields for role management (e.g., admin, user).
   * Schema example:

{

"name": "John Doe",

"email": "john.doe@example.com",

"password": "hashed\_password",

"role": "user"

}

1. **Products**

* Stores details about food items available on the platform.
* Fields include product name, description, price, category, and availability status.
* Schema example:

{

"name": "Cheese Pizza",

"description": "A delicious cheesy pizza",

"price": 12.99,

"category": "Pizza",

"availability": true

}

1. **Cart**

* Tracks items selected by users before placing an order.
* Each cart document is associated with a user ID and contains a list of products with their quantities.
* Schema example:

{

"userId": "unique\_user\_id",

"items": [

{

"productId": "unique\_product\_id",

"quantity": 2

}

]

}

**Orders**

* Records all orders placed by users, including details of purchased items, total cost, and delivery status.
* Includes timestamps for order creation and updates.
* Schema example:

{

"userId": "unique\_user\_id",

"items": [

{

"productId": "unique\_product\_id",

"quantity": 2,

"price": 12.99

}

],

"totalAmount": 25.98,

"status": "Pending",

"createdAt": "2024-11-27T10:30:00Z"

}

**4. Setup Instructions**

The following instructions guide you through setting up the SB Foods project on your local development environment.

**Prerequisites**

Ensure the following software is installed on your system:

1. **Node.js** (Version >= 14.x): [Download and Install Node.js](https://nodejs.org/)
2. **MongoDB**:
   * Local installation: [Download MongoDB](https://www.mongodb.com/try/download/community)
   * Alternatively, use a cloud-based solution like [MongoDB Atlas](https://www.mongodb.com/cloud/atlas).
3. **npm** (Node Package Manager): Comes bundled with Node.js.

**Installation Instructions**

**1. Clone the Repository**

Clone the project repository to your local machine using the following command:

git clone [repository- https://github.com/sowmiyalogeswaran/NM-SB-Foods-app.git]

**2. Navigate to the Project Directory**

Move into the project folder where the codebase resides:

cd [project-directory]

**3. Install Dependencies**

The project is split into two parts: frontend (client) and backend (server). Each has its dependencies.

**Frontend Setup**

1. Navigate to the client directory:

cd client

1. Install dependencies:

npm install

1. Return to the root directory:

cd ..

**Backend Setup**

1. Navigate to the server directory:

cd server

1. Install dependencies:

npm install

**4. Set Up Environment Variables**

The backend requires environment variables to run properly. These variables configure the server and database connection.

1. Create a .env file in the server directory.
2. Define the following variables in the .env file:

PORT=5000

MONGO\_URI=[Your MongoDB URI]

JWT\_SECRET=[Your Secret Key]

* + **PORT**: Specifies the port on which the server will run (default: 5000).
  + **MONGO\_URI**
  + **JWT\_SECRET**: A secure key used for signing and verifying JWT tokens. Choose a long, random string for this value.

**5. Running the Application**

**Start the Backend Server**

1. Navigate to the server directory:

cd server

1. Start the server:

npm start

By default, the backend server runs on http://localhost:5000.

**Start the Frontend Server**

1. Open a new terminal and navigate to the client directory:

cd client

1. Start the React development server:

npm start

The frontend server runs on http://localhost:3000 by default.

**6. Verify the Application**

1. Open your browser and navigate to http://localhost:3000 to access the frontend.
2. Ensure the backend is running and connected to the MongoDB database. Verify by performing user registration or fetching products from the database.

**5. Folder Structure**

Client:

client/

├── public/

├── src/

│ ├── components/

│ ├── pages/

│ ├── utils/

│ ├── App.js

│ ├── index.js

├── package.json

Server:

server/

├── models/

├── routes/

├── controllers/

├── middleware/

├── config/

├── server.js

├── package.json

**6. Running the Application**

Follow the steps below to start the SB Foods application locally. This involves running both the **frontend** (React.js) and **backend** (Node.js and Express.js) servers simultaneously.

**Start the Frontend Server**

The frontend is built using React.js and is hosted on a local development server. Here’s how to start it:

1. **Navigate to the client directory:**  
   cd client
2. **Install dependencies:**  
   Ensure that all required dependencies are installed. If not already done, run:

npm install

1. **Start the React development server:**  
   Launch the frontend server using the following command:

npm start

1. **Access the frontend application:**
   * By default, the React development server runs at http://localhost:3000.
   * Open this URL in your browser to view the application.

**Start the Backend Server**

The backend, powered by Node.js and Express.js, manages API endpoints and database interactions. Follow these steps:

1. **Navigate to the server directory**
2. **Install dependencies**
3. **Set up environment variables**
4. **Verify the backend is running**

**Verify Frontend-Backend Integration**

1. Ensure both the frontend and backend servers are running simultaneously.
   * Frontend: http://localhost:3000
   * Backend: http://localhost:5000
2. Perform the following actions to test integration:
   * Register as a new user on the frontend.
   * Browse products, add items to the cart, and place an order.
   * Ensure data is correctly stored in the database (MongoDB).

**Common Issues and Troubleshooting**

1. **Frontend Not Loading:**
   * Ensure the frontend server is running (npm start in the client directory).
   * Check if the port 3000 is occupied by another application.
2. **Backend Fails to Start:**
   * Ensure the .env file has the correct MONGO\_URI and PORT.
   * Verify MongoDB is running or accessible if using a local instance.
3. **Frontend-Backend Connection Issues:**
   * Check if the API base URL is configured correctly in the frontend (/src/utils/api.js).
   * If you’re using a proxy in the React app, verify the proxy field in client/package.json:

"proxy": "http://localhost:5000"

1. **MongoDB Connection Issues:**
   * Ensure that MongoDB is running locally or that your cloud-based MongoDB cluster is accessible.
   * If using MongoDB Atlas, verify IP whitelisting and connection string accuracy.

**7. API Documentation**

This section provides a detailed description of the API endpoints used in the SB Foods project. Each endpoint includes the HTTP method, route, required/request body (if applicable), and an example response.

**Users API**

**1. Register a New User**

**2. User Login**

**Products API**

**1. Fetch All Products**

**2. Fetch a Product by ID**

**3. Add a New Product (Admin Only)**

**Orders API**

**1. Place an Order**.

**2. Get All Orders (Admin Only)**

**8. Authentication**

Authentication is a crucial component of the SB Foods platform to ensure that users can securely log in, access personalized data, and perform actions like ordering and managing their profile. The platform uses **JWT (JSON Web Token)** for secure authentication and authorization of users.

**JWT Authentication**

JWT is used for securely transmitting information between the server and the client. In the case of SB Foods, JWT is used to authenticate users when they log in, ensuring that only authorized users can access protected routes (such as placing an order or accessing user profiles).

**How JWT Authentication Works**

1. **User Login:**  
   When a user logs in, they provide their credentials (email and password) through the POST /api/users/login endpoint. The server verifies the credentials and, if correct, generates a JWT token.
2. **Storing the Token:** The generated JWT is sent to the client. The client typically stores the token in one of the following places:
   * **Cookies:** A secure and HttpOnly cookie can be used to store the JWT, ensuring that the token is not accessible via JavaScript (mitigating cross-site scripting attacks).
   * **LocalStorage:** The JWT can also be stored in the browser's localStorage for ease of access, though this method is less secure than using cookies (especially regarding cross-site scripting attacks).
3. **Making Authenticated Requests:** When the client needs to make a request to a protected route (such as placing an order or updating the profile), it sends the JWT in the Authorization header as a **Bearer token**.
4. **Token Validation:** When the server receives a request with the JWT, it validates the token to ensure it is authentic and not tampered with. If the token is valid, the request is processed; otherwise, the server responds with a **401 Unauthorized** status.

**Middleware for Authentication**

To ensure that only authorized users can access specific routes (such as placing orders or managing user profiles), the server uses **middleware** to check if a valid JWT is included in the request.

1. **Authorization Middleware:** This middleware intercepts incoming requests to protected routes and verifies the JWT. If the JWT is valid, the middleware allows the request to proceed; otherwise, it blocks access and sends a **401 Unauthorized** };
2. **Using Middleware for Protected Routes:** The middleware is applied to any route that requires authentication.
3. **Explanation:**
   * **authenticateToken** ensures that only users with a valid token can place an order.
   * If the token is missing or invalid, the request is blocked, and the user receives a 401 Unauthorized error.

**Token Expiry and Refresh**

JWTs are typically configured to expire after a certain period (e.g., 1 hour). Once the token expires, the user will need to log in again to obtain a new token. This expiration ensures that the system remains secure even if a token is compromised.

For a better user experience, a **refresh token** can be used. The refresh token allows the user to obtain a new access token without having to log in again.

1. **Access Token (Short-Lived):**
   * Used to access protected routes.
   * Expires after a short period (e.g., 1 hour).
2. **Refresh Token (Long-Lived):**
   * Used to obtain a new access token when it expires.
   * Typically stored securely (e.g., in HttpOnly cookies) and is issued alongside the access token.

**Security Considerations**

* **Store tokens securely:**  
  Tokens should always be stored in **secure, HttpOnly cookies** to prevent access via JavaScript and mitigate cross-site scripting (XSS) attacks. LocalStorage should be avoided for highly sensitive applications.
* **Use HTTPS:**  
  Always use **HTTPS** to encrypt communication between the client and server, ensuring that tokens and sensitive data are not exposed to man-in-the-middle attacks.
* **Regularly expire and rotate tokens:**  
  Regular token expiry and using refresh tokens reduce the risk of long-term exposure in case a token is compromised.

**9. User Interface**

The **SB Foods** web application features a simple and user-friendly interface designed to provide a smooth food ordering experience. The interface is designed with responsiveness in mind, ensuring that the application works seamlessly on both desktop and mobile devices.

The user interface is divided into several key pages, each offering a distinct functionality. These pages are designed to enhance the user experience, from browsing food options to managing the cart and placing orders.

**Key Pages of the User Interface**

**1. Home Page**

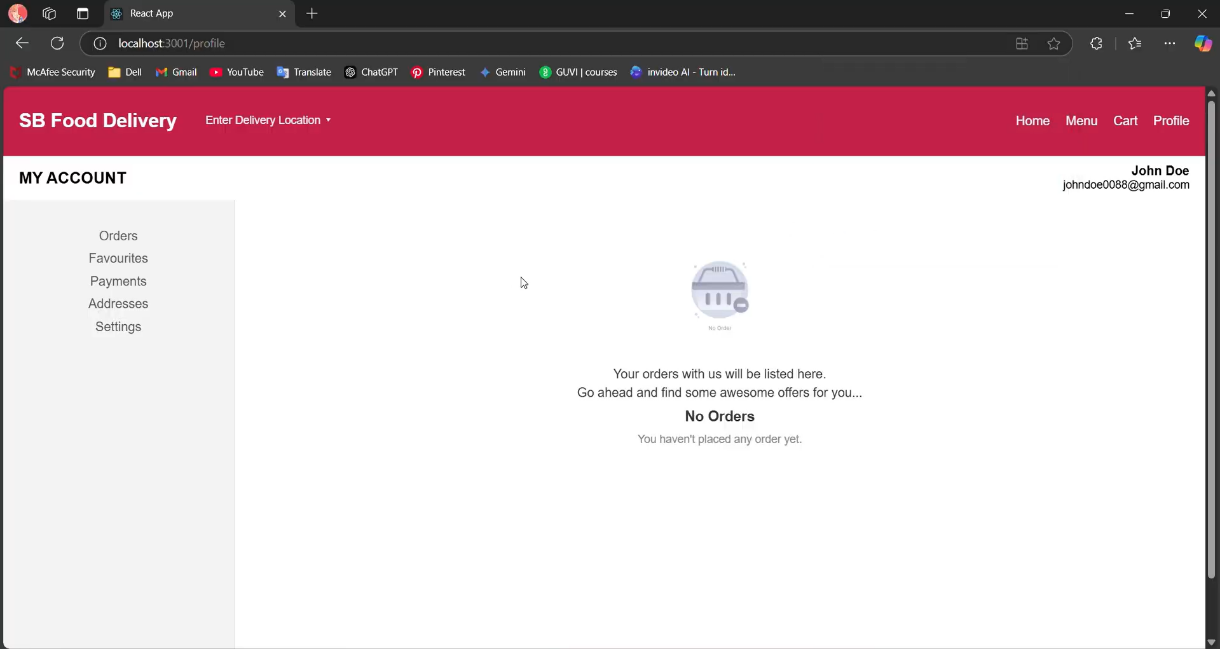
The **Home Page** serves as the main entry point for users. It provides the following features:

* **Menu Display:** The page shows all available food items in a grid or list format, making it easy for users to browse options.
* **Search and Filter Options:** Users can filter items by category (e.g., pizza, burgers, etc.), price range, and popularity. There’s also a search bar to allow users to quickly find specific food items.
* **Navigation:** The homepage includes navigation links to other pages such as the cart, order history, and the admin dashboard (for admins).

**Components:**

* + **Category Filters:** Filters by food type (e.g., vegetarian, non-vegetarian).
  + **Search Bar:** Allows users to search for specific food items.
  + **Product Cards:** Each food item is represented by a card with an image, name, and price.

**Design:** Clean, minimalistic design to make browsing and selection intuitive.



**2. Cart Page**

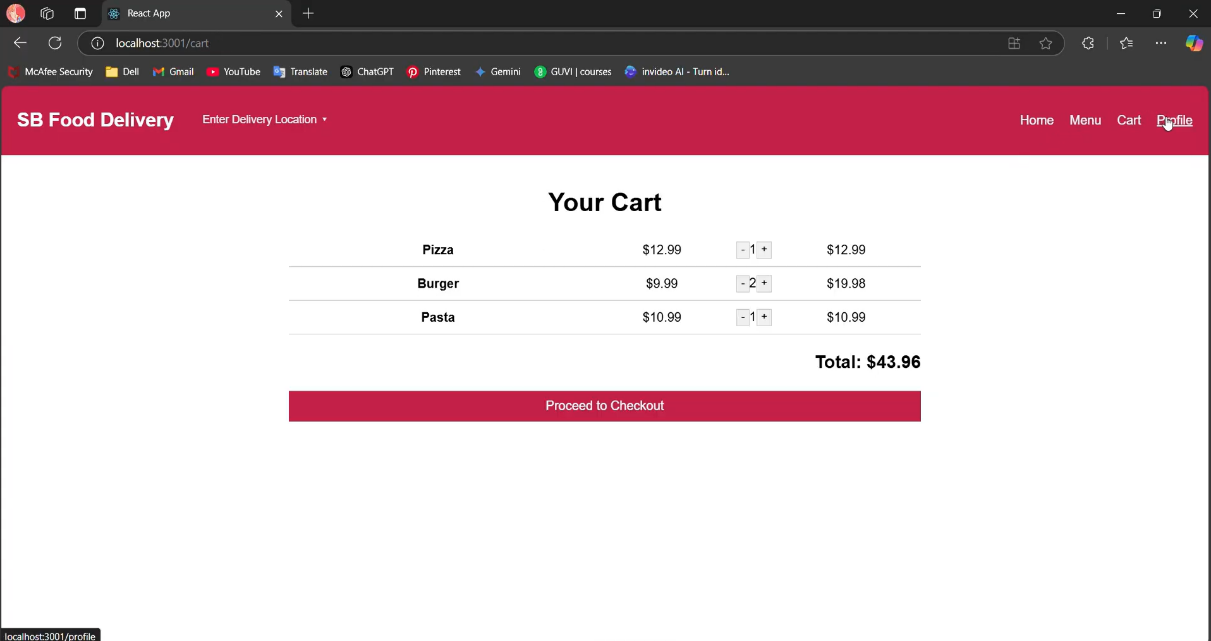
The **Cart Page** is where users can view and manage the items they’ve selected for purchase. Features of this page include:

* **Item List:** Displays all items added to the cart with their names, prices, and quantities.
* **Quantity Adjustments:** Users can increase or decrease the quantity of each item in the cart.
* **Remove Items:** Users can remove items from the cart.
* **Total Price:** The total cost of all selected items is displayed at the bottom.
* **Proceed to Checkout Button:** Users can proceed to checkout to finalize their orders.

**Components:**

* + **Item List:** Shows the name, image, quantity, and price of each item in the cart.
  + **Quantity Selector:** Users can adjust the quantity for each item.
  + **Remove Button:** Allows users to remove items from their cart.
  + **Checkout Button:** Directs users to the order confirmation page.

**Design:** Simple, with a focus on usability. The cart updates in real-time when items are added, removed, or quantities are adjusted.



**3. Order History Page**

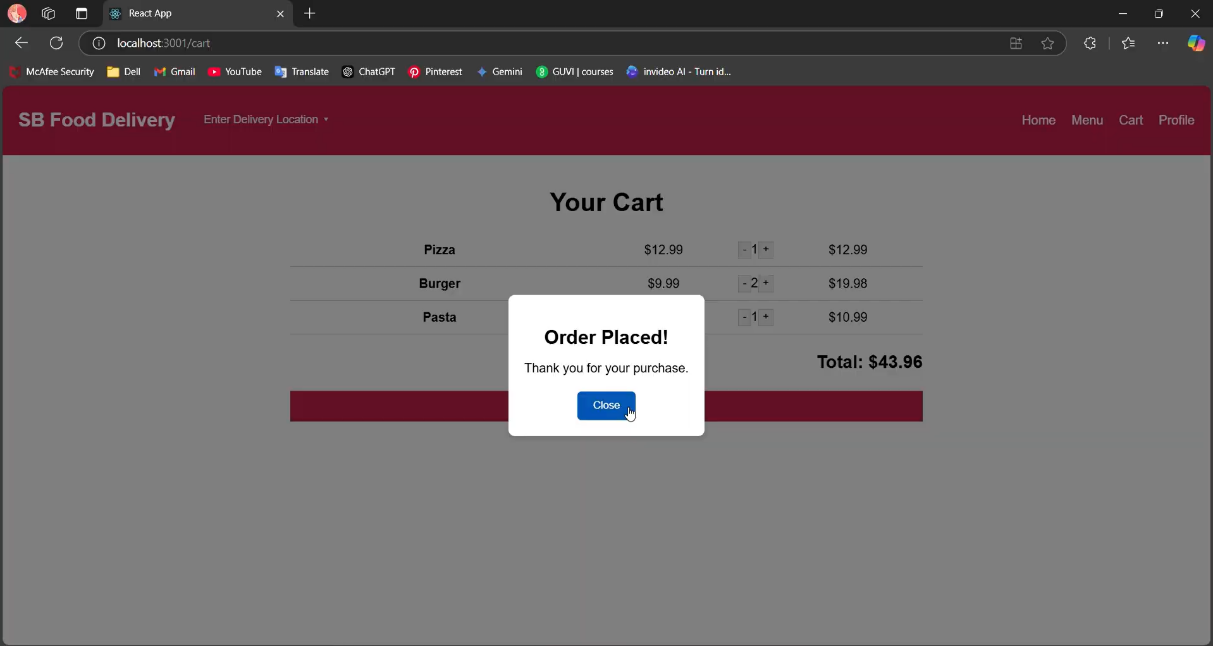
The **Order History Page** is accessible by logged-in users and provides a record of their previous orders. Features include:

* **Past Orders:** Users can view the status, date, and items of each past order.
* **Reorder Option:** Users can reorder the same items from previous orders with one click.
* **Order Details:** Clicking on an order reveals a detailed view, showing the individual items, quantities, and total cost.

**Components:**

* + **Order List:** Displays a list of past orders with order dates and status.
  + **Reorder Button:** Allows users to reorder the same items from a past order.
  + **Details Button:** Shows detailed order information, including a breakdown of the items and quantities.

**Design:** The page is organized to clearly display each past order, making it easy for users to revisit their previous choices.



**4. Admin Dashboard**

The **Admin Dashboard** provides administrative tools for managing the platform's users, products, and orders. It is only accessible by users with admin privileges. Features include:

* **User Management:** Admins can view a list of all registered users and their details. Admins can also delete users or modify their status.
* **Product Management:** Admins can add new products to the menu, edit existing product details (name, price, description), or remove products.
* **Order Management:** Admins can view a list of all orders, check the status of each order, and modify or cancel orders if needed.

**Components:**

* + **User Management Section:** Shows a list of users and provides options to modify or delete users.
  + **Product Management Section:** Allows adding, editing, or deleting food items in the menu.
  + **Order Management Section:** Displays all orders with options to update the status (e.g., pending, completed, canceled).

**Design:** The admin dashboard has a more functional design, with dropdown menus, tables, and action buttons for managing users, products, and orders.

**User Interface Design Considerations:**

* **Mobile Responsiveness:** The design adapts to different screen sizes, ensuring that users can interact with the app comfortably on mobile, tablet, or desktop devices.
* **User-Focused Design:** The interface is designed to be intuitive and easy to navigate. Key actions (e.g., adding items to the cart, logging in, and checking out) are prominent and accessible.
* **Consistent Branding:** The color scheme, typography, and layout are consistent across all pages, ensuring a unified and professional look.

**10. Testing**

Testing is an essential part of the software development lifecycle, ensuring that the application works as expected and providing confidence that new changes do not introduce bugs or regressions. In the **SB Foods** web application, a combination of unit testing, integration testing, and manual testing is employed to validate the functionality of both the frontend and backend.

**Tools Used**

1. **Jest (Unit Testing)**
   * **Jest** is a popular testing framework for JavaScript, primarily used for unit and integration testing of React components and JavaScript functions. It provides built-in functionalities for assertions, mocks, and spies, making it an ideal choice for testing the application logic, especially on the frontend.
   * **Key Features:**
     + Provides fast and reliable unit tests.
     + Built-in mocking, spying, and assertions to test components and functions in isolation.
     + Works well with React components, enabling developers to test component logic and lifecycle methods.
     + Supports snapshot testing for UI components.

**Use Cases:**

* + **Frontend Components:** Test if React components render correctly, handle events properly, and update states as expected.
  + **Helper Functions:** Test utility functions and business logic to ensure that they perform the expected actions.

1. **Postman (API Testing)**
   * **Postman** is a popular tool for testing API endpoints, allowing you to send requests to your backend and verify responses. It supports a wide range of HTTP methods (GET, POST, PUT, DELETE, etc.), making it easy to test the different API functionalities of the **SB Foods** app.
   * **Key Features:**
     + Allows sending requests with various HTTP methods and parameters.
     + Supports validation of API responses, including status codes, response bodies, and headers.
     + Can be used to create collections of API requests for batch testing.
     + Supports automated testing with the Postman Collection Runner.

**Use Cases:**

* + **API Endpoints:** Test that all API endpoints (e.g., register user, fetch products, place order) return the correct status code and response body.
  + **Error Handling:** Test edge cases where invalid input or server issues occur to ensure proper error handling.

**Testing Strategy**

To ensure the **SB Foods** web application works as intended, a structured testing approach is adopted, focusing on different layers of the application:

1. **Frontend Testing:**
   * **Component Testing:** Individual React components are tested to ensure they render correctly, handle user interactions, and update the state as expected.
   * **Event Handling:** Test the behavior of event handlers, such as clicking buttons, submitting forms, and changing input values, to ensure proper user interaction.
   * **UI Rendering:** Test if the UI components appear as expected, particularly when fetching data (e.g., product lists, cart items).
   * **Snapshot Testing:** Create snapshots of React components to verify that their structure does not change unexpectedly after updates.

**Example Strategy for Frontend Testing:**

* + Test that the **Home Page** loads correctly with product items and filters.
  + Test that clicking the **Add to Cart** button updates the cart count and item list.
  + Test that the **Cart Page** displays the correct items and allows users to modify quantities.

1. **Backend Testing:**
   * **API Endpoint Testing:** Each API endpoint is tested using **Postman** to ensure that the backend logic works correctly, with expected status codes and response formats.
   * **Database Interaction:** Test database CRUD operations to verify that data is correctly stored, retrieved, updated, and deleted.
   * **Authorization & Authentication:** Test that authentication routes (e.g., login and registration) are functioning correctly and that users are only able to access restricted routes if they are authenticated and authorized.
   * **Error Handling:** Ensure that the API handles invalid inputs, missing parameters, and server errors properly, returning appropriate error messages.

**Example Strategy for Backend Testing:**

* + Test **User Registration** API to ensure valid data creates a new user, and invalid data returns appropriate errors.
  + Test **Product Fetching** API to ensure all products are returned correctly.
  + Test **Order Creation** API to verify that the order is placed correctly and updates the database.

1. **End-to-End Testing:**
   * **Integration Testing:** Verify that the frontend and backend work together correctly by simulating user flows (e.g., adding items to the cart, proceeding to checkout, and placing an order).
   * **Real-World Scenario Testing:** Test the entire application workflow, from user registration to order placement, to ensure that all components interact seamlessly and the user experience is smooth.

**Manual Testing**

In addition to automated testing with Jest and Postman, manual testing is performed during development to catch any issues that may not be easily detected through automated tests. Manual testing includes:

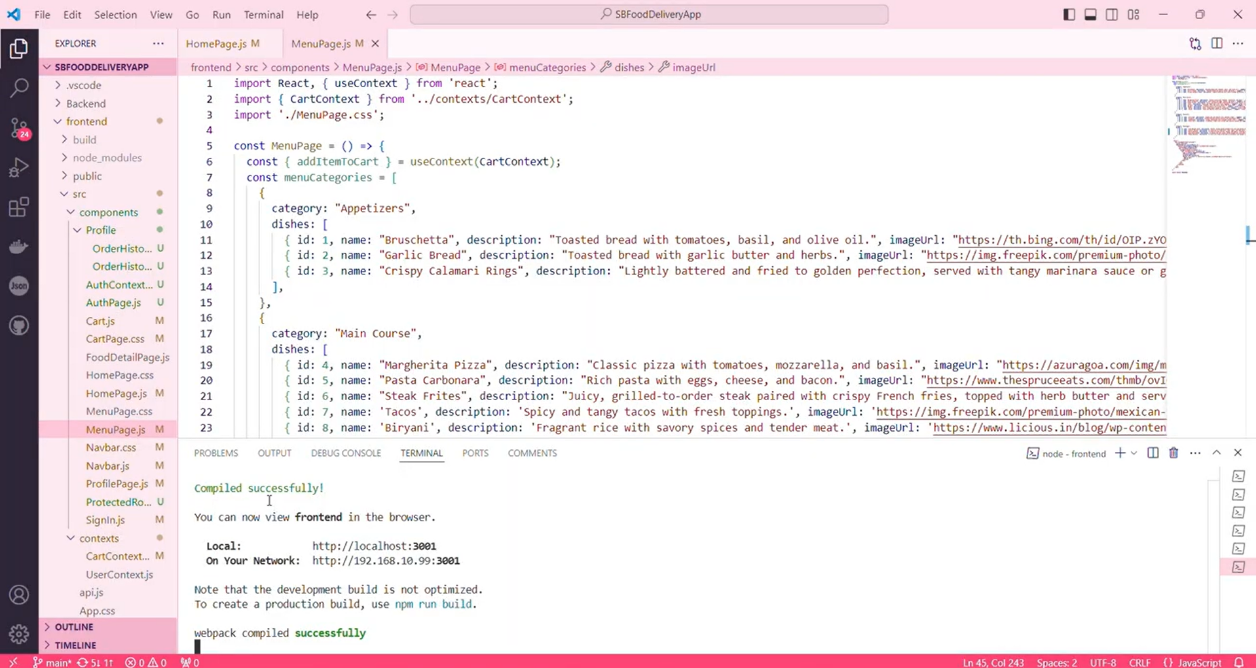
* **User Acceptance Testing (UAT):** A small group of users or testers are asked to perform real-world tasks on the app to identify any usability issues.
* **Cross-Browser Testing:** Ensures that the application works properly on different web browsers, such as Chrome, Firefox, and Edge.
* **Responsive Design Testing:** Ensures that the application is fully functional and visually appealing on various screen sizes (desktop, tablet, mobile).

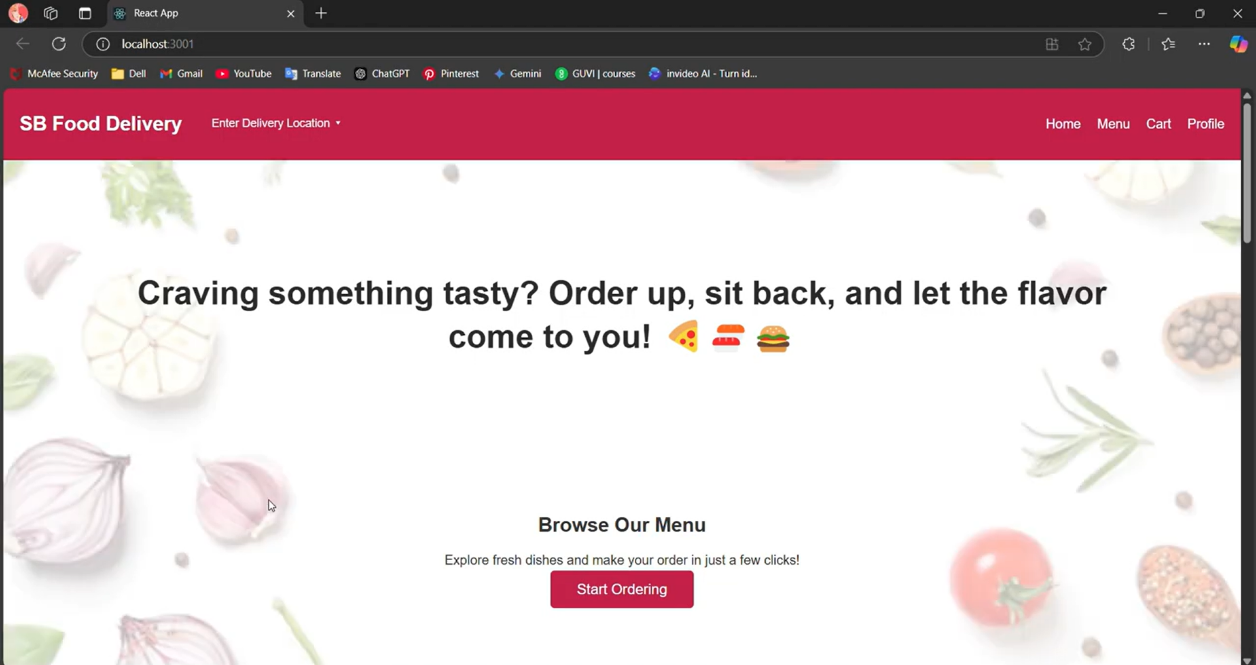
**Testing Workflow**

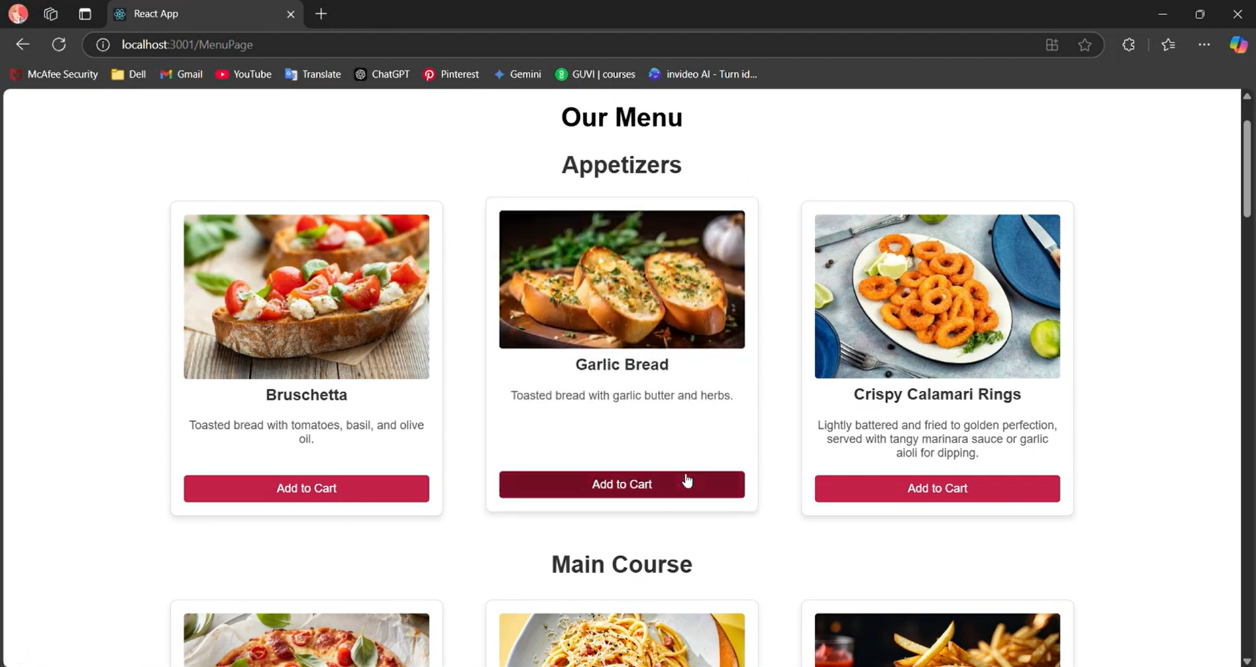
1. **Test API Endpoints** using **Postman** to ensure correct responses for various request types (GET, POST, PUT, DELETE).
2. **Write Unit Tests** for React components and helper functions using **Jest**.
3. **Perform Integration Tests** to ensure the frontend and backend communicate correctly.
4. **Manually Test** the entire application for usability, cross-browser compatibility, and responsiveness.
5. **Run Jest Tests** and **Postman Tests** in CI/CD pipelines to ensure automated tests run on every commit.

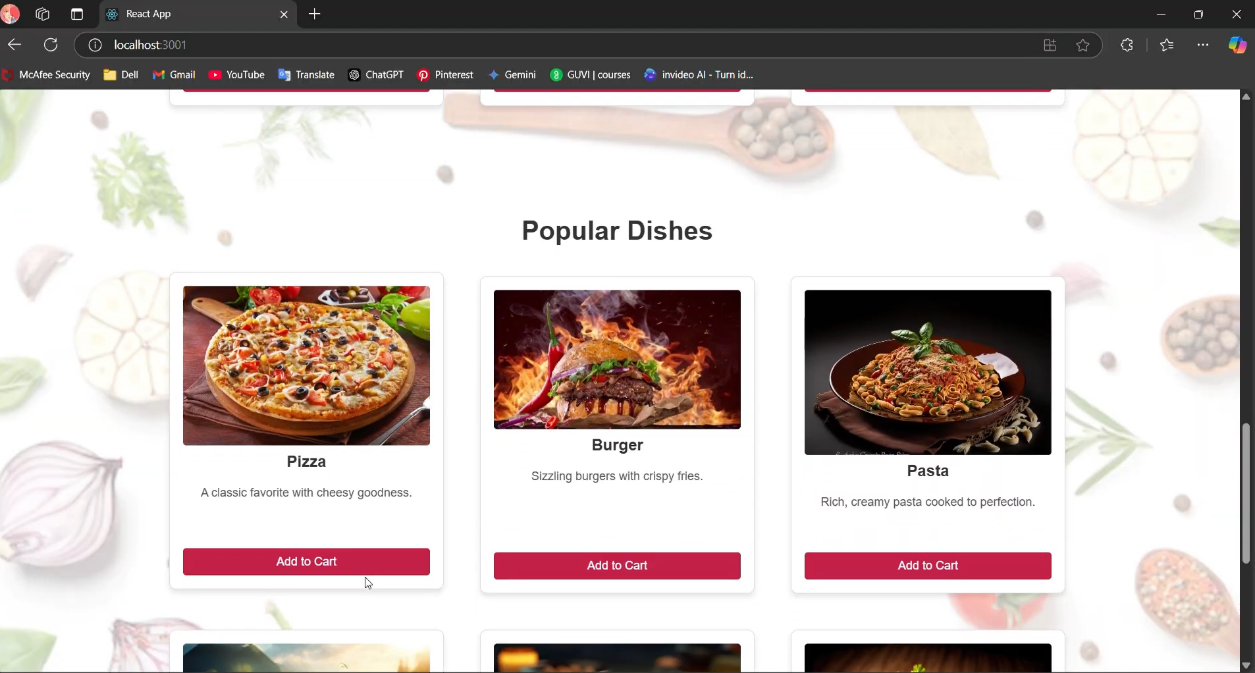
**11. Screenshots or Demo**

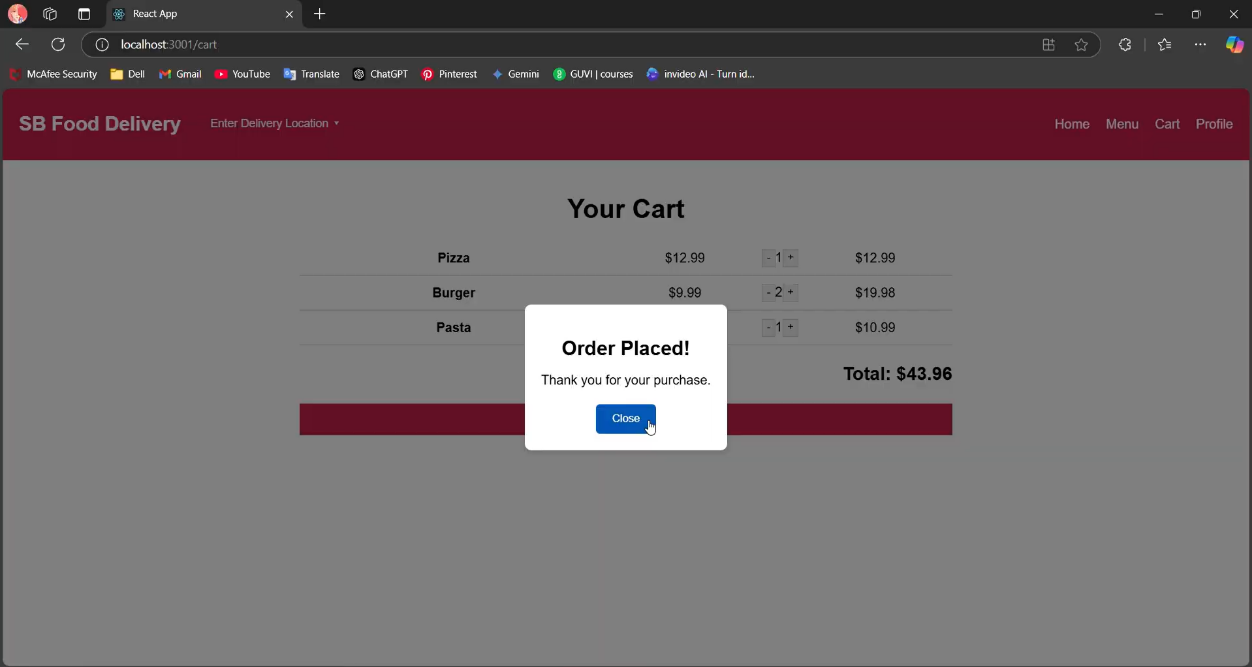
* Link to demo: <https://drive.google.com/file/d/1E6PktBRlmX5WZC7-yj-3Is04XrnGxb7m/view?usp=sharing>
* screenshots:

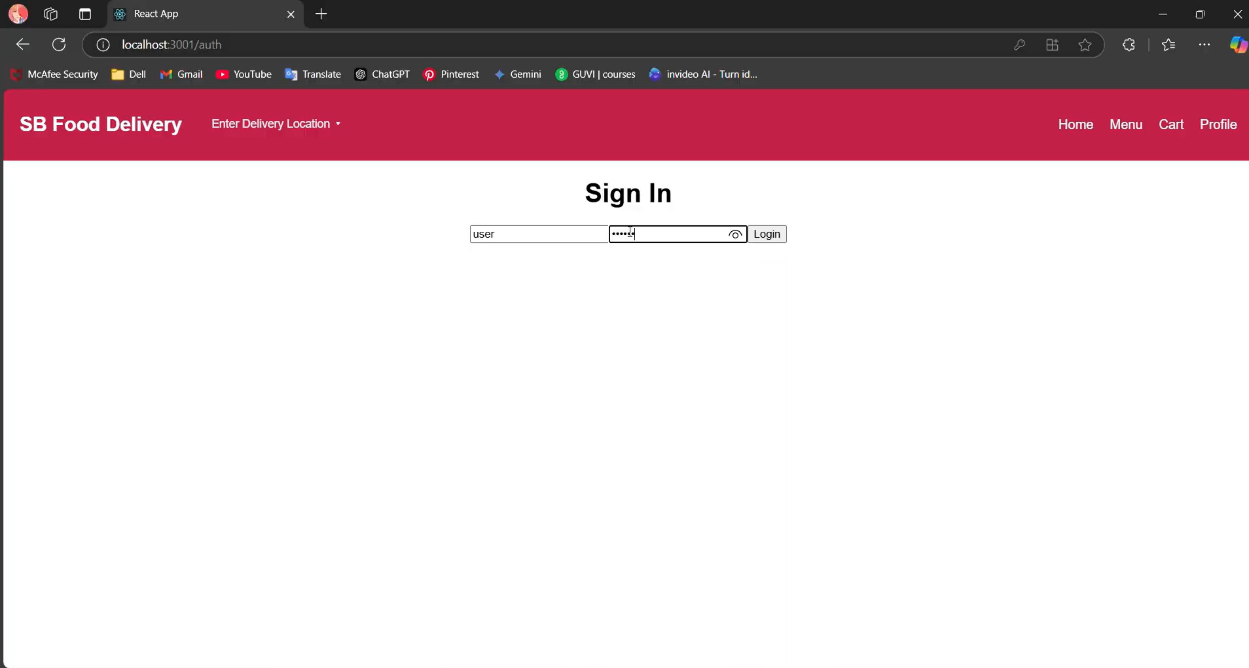


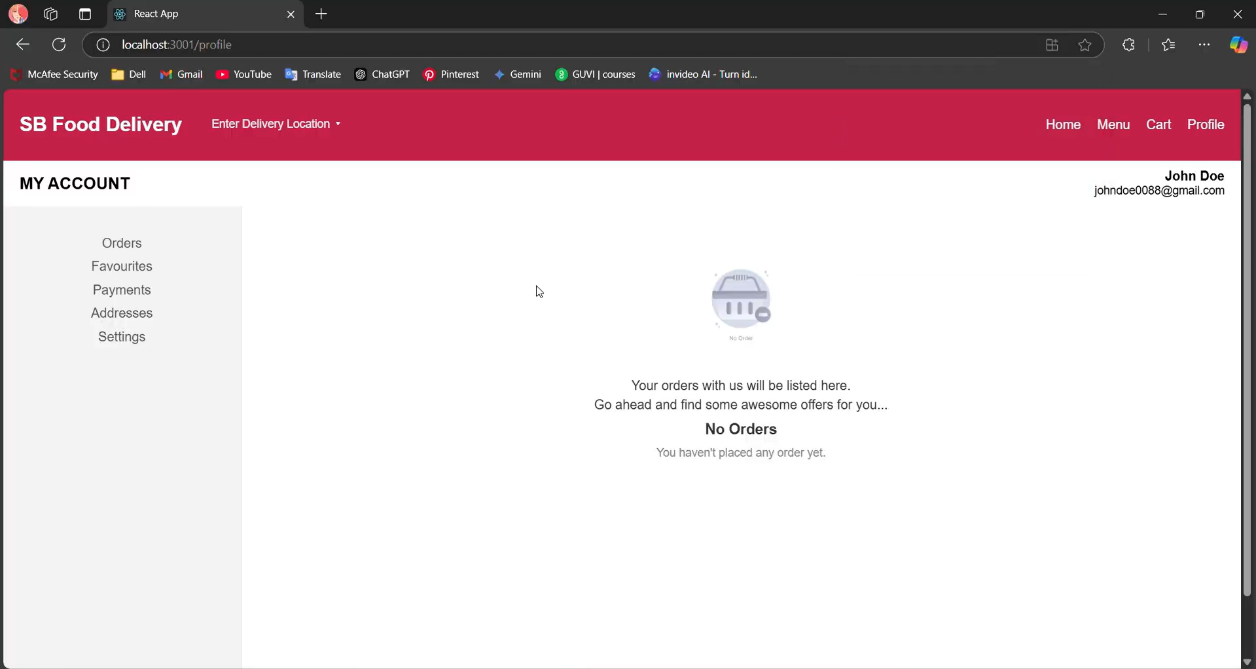












**12. Known Issues**

Despite the **SB Foods** web application being robust and functional, there are a few known issues that need attention to ensure an optimal user experience under various conditions. These issues, while not critical, may affect user experience and system performance, especially in certain scenarios.

1. Cart Update Delay Under High Load Conditions

2. UI Responsiveness on Smaller Screens

3. Performance Issues During Heavy Product Load

4. Inconsistent State Management on Cart Page

5. Admin Dashboard Performance on Large Datasets

**13. Future Enhancements**

As the **SB Foods** web application continues to evolve, there are several key areas where additional features and improvements are planned to further enhance the user experience and broaden the platform's capabilities. These future enhancements aim to improve customer engagement, streamline the ordering process, and provide more robust functionality to both users and administrators. Here are the primary enhancements that are in the pipeline:

1. Add a live chat feature for customer support.
2. Implement advanced search filters for food items.
3. Integrate real-time delivery tracking with GPS.