**Introduction**

**Project Title:** **SB Foods A Revolution in Online Food Ordering**

**Team Members:**

**Sharath J :** Frontend Developer, responsible for the design and implementation of the user interface using React, state management with Redux, and ensuring responsive design across devices.

**Vishal Krishna L A:** Backend Developer, responsible for designing the API endpoints, implementing serverside logic, handling authentication with JWT, and managing realtime notifications for order updates.

**Kamal Raj:** Database Administrator, responsible for schema design and database optimization, ensuring data consistency and performing CRUD operations on MongoDB for user data, product details, and order management.

**Shanmugam P:** Fullstack Developer, overseeing integration of the frontend and backend, deploying the application, ensuring seamless interaction between the clientside and serverside, and implementing security features like tokenbased authentication.

**Project Overview**

**Purpose:**

SB Foods is an innovative platform that simplifies the food ordering experience for users. The purpose of the application is to provide a seamless, intuitive interface for customers to browse, select, and customize their food orders while ensuring quick order processing and efficient management for restaurants. The goal is to provide a personalized food ordering experience by offering dynamic product categories, customizable meal options, and realtime order tracking.

**Features:**

* **User Authentication:** A secure authentication system that allows users to register, log in, and access their personalized accounts using JWTbased session management.
* **Product Browsing:** A categorically organized product catalog that allows users to search and filter meals based on preferences such as cuisine, dietary requirements (vegetarian, vegan), and meal size.
* **Order Customization:** Users can customize their meals, adjusting ingredients, portion sizes, and special instructions before adding them to the cart.
* **RealTime Notifications:** The application sends push notifications to the user’s device, providing realtime updates on the order's status, such as order confirmation, preparation, dispatch, and delivery.
* **Admin Dashboard:** An intuitive interface for restaurant owners and admins to manage their products, view customer orders, monitor inventory, and analyze sales data.
* **Payment Integration (Planned):** Integration with a payment gateway to allow users to securely pay for their orders online, with support for various payment methods like credit cards, digital wallets, and bank transfers.
* **Order History and Analytics:** Users can view past orders, reorder items, and get personalized recommendations based on their order history.

**Architecture**

**Frontend:**

* The frontend is designed to provide an engaging, userfriendly interface with React:
* ComponentBased Architecture: The frontend is broken down into reusable components such as `Navbar`, `ProductCard`, `Cart`, and `OrderSummary`.
* React Router: React Router is used for clientside routing, ensuring smooth transitions between different views like Home, Product List, Cart, and Checkout.
* State Management with Redux: Redux is used for managing the global state of the application, such as the shopping cart, user authentication status, and current order details. Redux actions and reducers are employed to manage these states efficiently across the entire app.
* Axios: Axios is used for asynchronous data fetching from the backend API, including user registration, product fetching, order placement, and user authentication.

**Backend:**

* The backend is implemented using Node.js and Express.js, providing a robust and scalable server to handle client requests:
* RESTful API: The backend follows the REST architectural pattern, ensuring that different entities like users, products, and orders are represented as resources that can be accessed through standard HTTP methods (GET, POST, PUT, DELETE).
* JWT Authentication: User authentication is handled via JWT (JSON Web Tokens), which are issued upon successful login and are used to verify the identity of users for protected routes.
* Express Middleware: Middleware is used for logging, error handling, and validating the JWT token for protected routes.

**Database:**

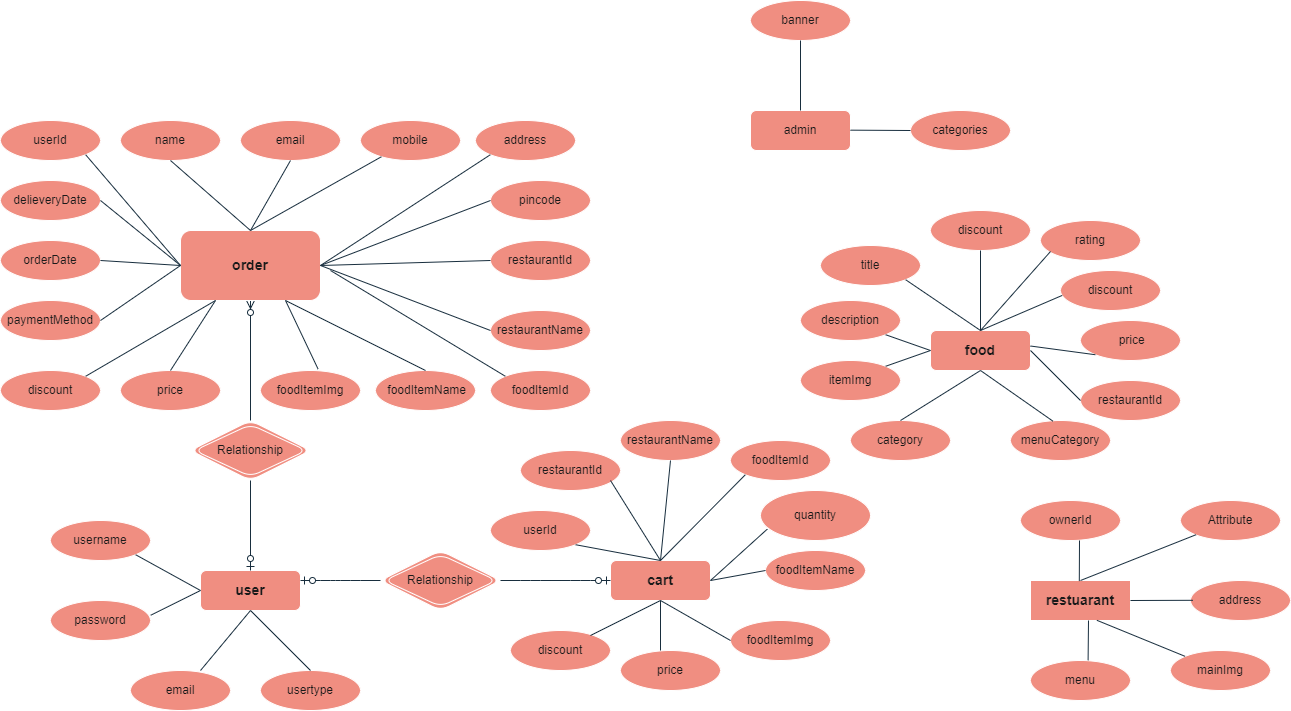
* The backend interacts with a MongoDB database to store and manage data:
* MongoDB Schema:
* User Schema: Contains user details such as `email`, `password`, `address`, `orderHistory`, and `wishlist`.
* Product Schema: Defines the structure for product entries, including `name`, `description`, `price`, `category`, `ingredients`, and `imageUrl`.
* Order Schema: Stores information about each order, including `userId`, `products`, `totalAmount`, `deliveryAddress`, `orderStatus`, and `paymentStatus`.
* Admin Schema: Stores admin credentials and access privileges.

**Data Relationships:**

**Users to Orders:** A onetomany relationship where a user can have multiple orders.

**Orders to Products:** A manytomany relationship where each order can have multiple products and each product can appear in many orders.

**Database Diagram:**



**Setup Instructions**

**Prerequisites:**

Node.js (version 16.0 or higher)

MongoDB (either local or hosted on MongoDB Atlas)

Git (for cloning the repository)

**Installation:**

1. Clone the repository:

git clone https://github.com/yourrepo/sbfoods.git

2. Install frontend dependencies:

cd sbfoods/client

npm install

3. Install backend dependencies:

cd sbfoods/server

npm install

4. Set up environment variables:

Create a `.env` file in the root directory of the `server` and add the following:

MONGODB\_URI=mongodb://localhost:27017/sbfoods

JWT\_SECRET=yourjwtsecretkey

5. Run the application:

Start the frontend server:

cd sbfoods/client

npm start

Start the backend server:

cd sbfoods/server

npm start

**Folder Structure**

**Client:**

public/: Contains public assets, such as the `index.html` and static files.

src/

components/: React components like `Navbar`, `ProductList`, `Cart`, `Checkout`.

redux/: Contains actions, reducers, and the Redux store setup for state management.

assets/: Images, styles, and other media assets.

App.js: Main component that initializes routing and global configurations.

services/: Contains Axios API requests to interact with the backend.

**Server:**

controllers/: Functions to handle API logic like user registration, login, product management, and order processing.

models/: Mongoose models defining data schemas for Users, Products, and Orders.

routes/: Express routes that map to specific controllers for different API requests.

middleware/: Custom middleware functions, including authentication validation and error handling.

config/: Configuration files for setting up the database connection and managing environment variables.

**Running the Application**

To run the application locally, follow these steps:

**Frontend:**

1. Navigate to the `client` directory:

cd sbfoods/client

2. Start the frontend server:

npm start

1. Navigate to the `server` directory:

cd sbfoods/server

2. Start the backend server:

npm start

**API Documentation**

**User Endpoints:**

POST /api/users/register: Registers a new user.

Request Body: `{ name, email, password, address }`

Response: `{ token, userId }`

POST /api/users/login: Authenticates a user.

Request Body: `{ email, password }`

Response: `{ token }`

**Product Endpoints:**

GET /api/products: Fetches a list of all products.

Response: `{ products: [{ id, name, description, price, image }] }`

POST /api/products: Adds a new product (Admin Only).

Request Body: `{ name, description, price, category, ingredients, imageUrl }`

Response: `{ message: "Product added successfully" }`

**Order Endpoints:**

POST /api/orders: Places a new order.

Request Body: `{ userId, products, deliveryAddress, paymentMethod }`

Response: `{ orderId, status, estimatedDeliveryTime }`

**Authentication**

* For authentication, we use JWT (JSON Web Token). The user is required to log in with their email and password to receive a token. This token is used for authorizing access to protected routes like order placement, product management, and order history.

**Challenges and Solutions**

* **RealTime Notifications:** Initially, we faced challenges in implementing realtime notifications using Socket.io. After research and testing, we successfully integrated Socket.io with both the frontend and backend, ensuring that users received realtime updates on their orders.
* **Database Performance:** We optimized database queries by using indexes on frequently searched fields like product name, category, and user email, significantly improving the response time for fetching products and user orders.
* This extended documentation gives a clearer picture of how the project is structured, the technologies used, and the features implemented. You can continue to enhance this by adding additional sections based on specific implementation details, challenges faced, or testing strategies employed.