



MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE

(An Autonomous Institution)
Kodambakkam, Chennai-600024.



MERN STACK POWERED BY MONGO DB

DEPARTMENT
OF
ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

TOPIC: SB Foods - Food ordering application

CATEGORY: FULLSTACK DEVELOPMENT – MERN

A PROJECT REPORT

Submitted by

| TEAM | NAME | REG.NO |
|-------------|-------------------------|--------------|
| TEAM LEADER | SRIYA VAISHNAVI. V | 311521243055 |
| TEAM MATE 2 | KONAR SARANYA. S | 311521243025 |
| TEAM MATE 3 | KUNGUMA DHARSHINI. M | 311521243028 |
| TEAM MATE 4 | SABARISH | 311521243040 |

SB FOODS: FOOD ORDERING APPLICATION

CATEGORY: FULLSTACK DEVELOPMENT - MERN

ABSTRACT

The SB Food Ordering App is a dynamic and user-friendly platform developed using the MERN (MongoDB, Express.js, React.js, Node.js) stack, designed to revolutionize the food ordering experience. This application bridges the gap between restaurants and customers by offering a seamless and efficient online food ordering system.

The app features a responsive and intuitive interface that allows users to browse diverse menus, customize their orders, and track them in real-time. Leveraging MongoDB for a robust database, it ensures efficient management of user profiles, restaurant data, and order histories. Express.js and Node.js provide a scalable backend, ensuring secure and smooth API interactions. React.js powers the frontend, offering a dynamic user experience and fast-loading pages.

With features such as user authentication, payment integration, and personalized recommendations, the app delivers convenience for users while supporting restaurants with order management and analytics tools. The SB Food Ordering App exemplifies the potential of full-stack development to create innovative solutions in the growing online food delivery market.

TABLE OF CONTENTS

| S.NO | TITLE |
|------|------------------------------|
| 1 | Project description |
| 2 | Problem scenario |
| 3 | Requirements |
| 4 | Architecture design |
| 5 | Application flow |
| 6 | Features and functionalities |
| 7 | Database design |
| 8 | Implementation |
| 9 | Testing and deployment |
| 10 | Output screenshots |
| 11 | Challenges and solutions |
| 12 | Future enhancements |
| 13 | Conclusions |
| 14 | References |

PROJECT DESCRIPTION:

The SB Food Ordering App is a robust and scalable web application developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js), designed to simplify and enhance the food ordering process for customers and restaurants. The app features a user-friendly interface, enabling customers to browse dynamic restaurant menus, search and filter items, and place orders effortlessly. Secure user authentication using JWT ensures personalized experiences and data privacy, while real-time order tracking and instant notifications enhance operational efficiency. Integrated with a secure payment gateway, the app supports various payment methods, making transactions seamless. Restaurants benefit from an admin dashboard that allows them to manage menus, track orders, and analyze sales data. The technical stack ensures a responsive frontend with React.js, a scalable backend with Node.js and Express.js, and structured data storage with MongoDB.

- User-Friendly Interface
- Dynamic Restaurant Listings
- User Authentication
- Real-Time Order Management
- Admin Dashboard

PROBLEM SCENARIO: Late night craving resolution

Meet Lisa, a college student burning the midnight oil to finish her assignment. As the clock strikes midnight, her stomach grumbles, reminding her that she skipped dinner. Lisa doesn't want to interrupt her workflow by cooking, nor does she have the energy to venture outside in search of food.

Solution with Food Ordering App:

1. Lisa opens the Food Ordering App on her smartphone and navigates to the late-night delivery section, where she finds a variety of eateries still open for orders.
2. She scrolls through the options, browsing menus and checking reviews until she spots her favorite local diner offering comfort food classics.
3. Lisa selects a hearty bowl of chicken noodle soup and a side of garlic bread, craving warmth and satisfaction in each bite.
4. With a few taps, she adds the items to her cart, specifies her delivery address, and chooses her preferred payment method.
5. Lisa double-checks her order details on the confirmation page, ensuring everything looks correct, before tapping the "Place Order" button.
6. Within minutes, she receives a notification confirming her order and estimated delivery time, allowing her to continue working with peace of mind.
7. As promised, the delivery arrives promptly at her doorstep, and Lisa eagerly digs into her piping hot meal, grateful for the convenience and comfort provided by the Food Ordering App during her late-night study session.

This scenario illustrates how a Food Ordering App caters to users' needs, even during unconventional hours, by offering a seamless and convenient solution for satisfying late-night cravings without compromising on quality or convenience.

SYSTEM REQUIREMENTS:

To develop a full-stack food ordering app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm:

Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: `npm install express`

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide:

<https://reactjs.org/docs/create-a-new-react-app.html>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

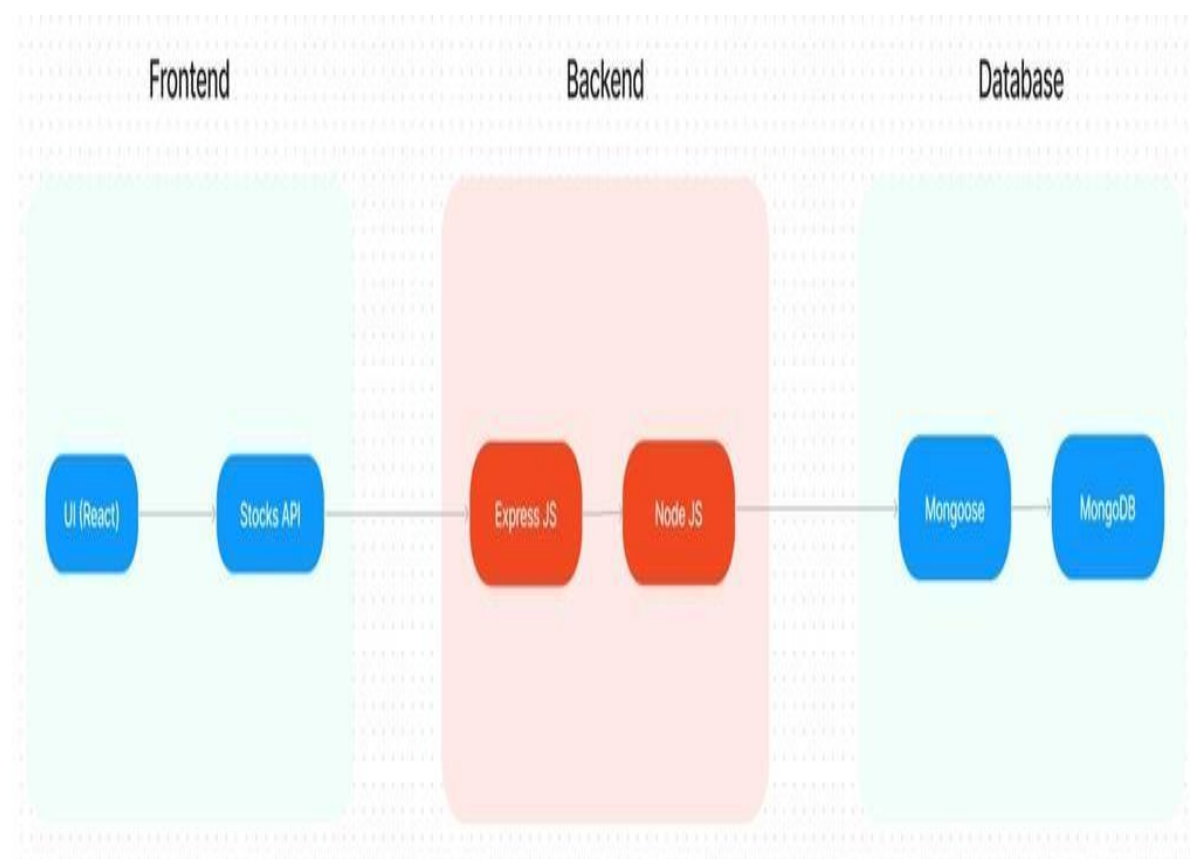
Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

TECHNICAL ARCHITECTURE:



In this architecture diagram:

Frontend

- **Technologies Used:** React.js, HTML, CSS
- **Components:**
 - **User Interface (UI):** Responsive and interactive interface for seamless navigation.
 - **Product Listings:** Filters, search functionality, and recommendation systems.
 - **Authentication:** User registration, login, and role-based access (admin/user).
 - **Cart:** Dynamic cart updates with pricing and quantity management.
 - **Profile Management:** User profile editing and order history.

Backend

- **Technologies Used:** Node.js, Express.js
- **API Endpoints:**
 - **User Management:** Registration, login, and role-based authentication.
 - **Order Management:** CRUD operations for orders.
 - **Product Management:** CRUD operations for product listings.
 - **Admin Dashboard:** Admin-specific APIs for analytics and manage

Database

- **Technologies Used:** MongoDB, Mongoose ORM
- **Data Collections:**
 - **Users:** Stores user information and authentication credentials.
 - **Products:** Stores product details, images, prices, and categories.
 - **Orders:** Tracks order details, including status and user data.
 - **Cart:** Temporarily stores user-selected products before checkout.

APPLICATION FLOW

1. User Flow:

- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available products in the platform.
- Users can add the products they wish to their carts and order.
- They can then proceed by entering address and payment details. • After ordering, they can check them in the profile section.

2. Restaurant Flow:

- Restaurants start by authenticating with their credentials.
- They need to get approval from the admin to start listing the products.
They can add/edit the food items.

3. Admin Flow:

- Admins start by logging in with their credentials.
- Once logged in, they are directed to the Admin Dashboard.
- Admins can access the users list, products, orders, etc.,

FEATURES AND FUNCTIONALITIES:

The SB Food Ordering App offers a comprehensive set of features tailored to provide a seamless and efficient experience for customers and restaurant administrators.

1. Features for Customers

1.1. User-Friendly Interface:

- A responsive and visually appealing design for easy navigation.
- Mobile-friendly layout for accessibility across devices.

1.2. Menu and Product Listings:

- Dynamic menus with detailed product descriptions, images, and prices.
- Filters and search functionality to quickly find desired items.
- AI-based recommendations to enhance the ordering experience.

1.3 Authentication and User Profiles:

- Secure user registration and login using JWT authentication.
- Role-based access control to distinguish between users and admins.
- Profile management, including the ability to update personal information and view order history.

1.4 Order Management:

- Real-time order placement and status tracking.
- Option to save favorite items for quick future orders.
-

1.5 Cart System:

- Dynamic cart updates with item quantity management.
- Real-time price calculations, including taxes and discounts.

1.6 Payment Integration:

- Multiple payment options, including credit/debit cards, UPI, and wallets.
- Secure payment gateways for hassle-free transactions.

2. Admin Features

2.1 Dashboard:

- A centralized admin panel for managing products, orders, and customer data.
- Real-time analytics for monitoring sales, revenue, and trends.

2.2 Product Management:

- CRUD operations for adding, updating, and deleting menu items.
- Real-time updates to product availability and pricing.

2.3 Order Management:

View and manage customer orders, including updates to order status (e.g., confirmed, prepared, delivered).

2.4 Insights and Analytics:

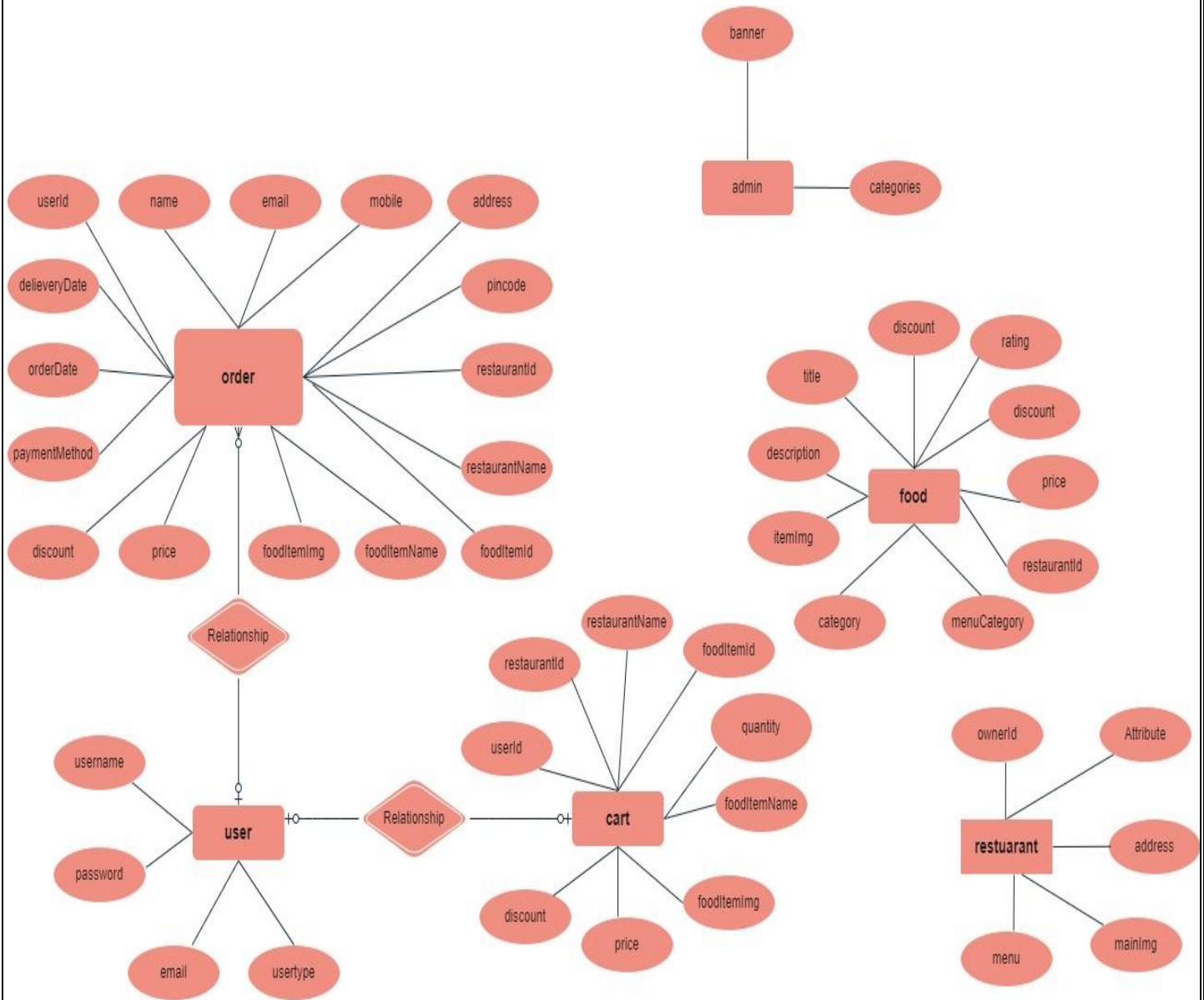
Generate sales reports and monitor restaurant performance metrics.

Additional Functionalities

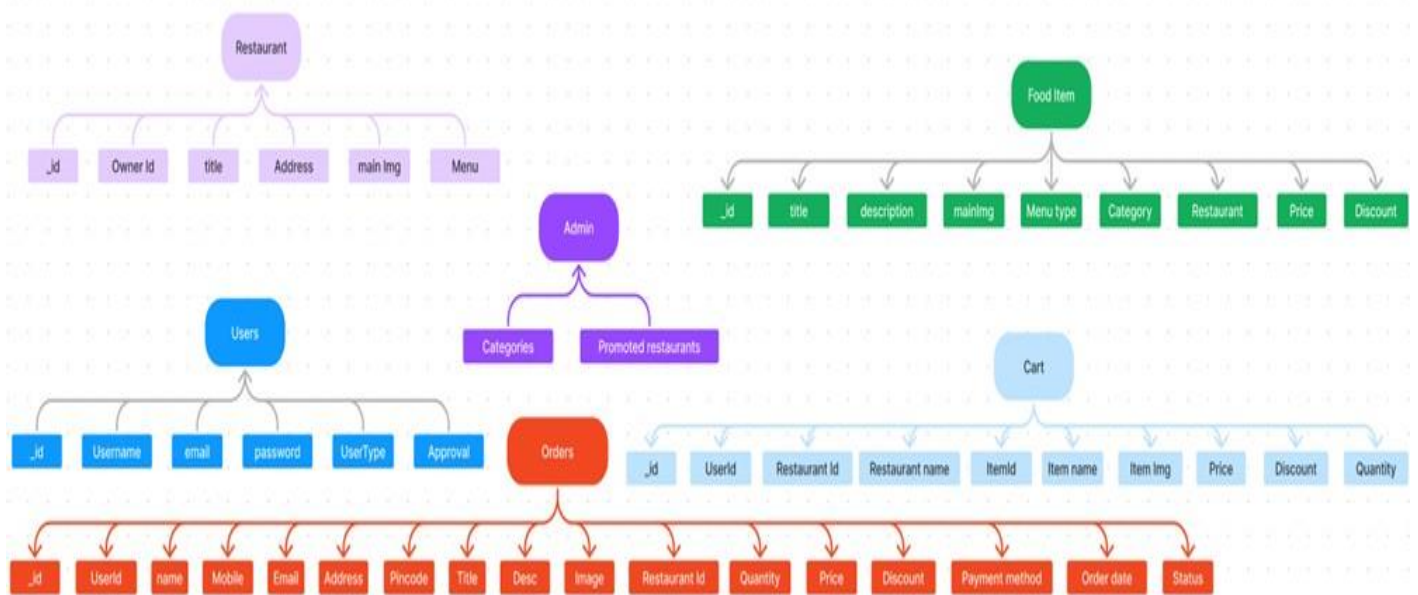
- **Real-Time Notifications:** Alerts for new orders and order status updates.
- **Multi-Language Support:** (Planned) To cater to a diverse audience.
- **Chat Support:** (Planned) Integrated chatbot for customer queries and support.
- **Location-Based Services:** (Planned) Delivery tracking and nearby restaurant suggestions.
- The SB Food Ordering App is designed to cater to both customer convenience and restaurant operational efficiency, making it an all-in-one solution for the food delivery ecosystem.

DATABASE DESIGN:

ER MODEL



- The Database section represents the database that stores collections for Users, Admin, Cart, Orders and products



The SB Foods ER-diagram represents the entities and relationships involved in an food ordering e-commerce system. It illustrates how users, restaurants, products, carts, and orders are interconnected. Here is a breakdown of the entities and their relationships:

User: Represents the individuals or entities who are registered in the platform.

Restaurant: This represents the collection of details of each restaurant in the platform.

Admin: Represents a collection with important details such as promoted restaurants and Categories.

Products: Represents a collection of all the food items available in the platform.

Cart: This collection stores all the products that are added to the cart by users. Here, the elements in the cart are differentiated by the user Id.

Orders: This collection stores all the orders that are made by the users in the platform.

FEATURES:

1. **Comprehensive Product Catalog:** SB Foods boasts an extensive catalog of food items from various restaurants, offering a diverse range of items and options for shoppers. You can effortlessly explore and discover various products, complete with detailed descriptions, customer reviews, pricing, and available discounts, to find the perfect food for your hunger.
2. **Order Details Page:** Upon clicking the "Shop Now" button, you will be directed to an order details page. Here, you can provide relevant information such as your shipping address, preferred payment method, and any specific product requirements.
3. **Secure and Efficient Checkout Process:** SB Foods guarantees a secure and efficient checkout process. Your personal information will be handled with the utmost security, and we strive to make the purchasing process as swift and trouble-free as possible.

4. **Order Confirmation and Details:** After successfully placing an order, you will receive a confirmation notification. Subsequently, you will be directed to an order details page, where you can review all pertinent information about your order, including shipping details, payment method, and any specific product requests you specified.

In addition to these user-centric features, SB Foods provides a robust restaurant dashboard, offering restaurants an array of functionalities to efficiently manage their products and sales. With the restaurant dashboard, restaurants can add and oversee multiple product listings, view order history, monitor customer activity, and access order details for all purchases.

SB Foods is designed to elevate your online food ordering experience by providing a seamless and user-friendly way to discover your desired foods. With our efficient checkout process, comprehensive product catalog, and robust restaurant dashboard, we ensure a convenient and enjoyable online shopping experience for both shoppers and restaurants alike.

IMPLEMENTATION:

The implementation of the SB Food Ordering App involves integrating the MERN stack to create a seamless user experience. The frontend, developed using React.js with responsive design, includes components for navigation, product listings, cart management, and user profiles, all powered by API communication with the backend. The backend, built with Node.js and Express.js, provides secure RESTful APIs for user authentication, product management, and order processing, with robust error handling and JWT-based security. MongoDB, managed via Mongoose ORM, handles data storage with schemas for users, products, orders, and carts, ensuring efficient queries and scalability. The application is integrated and tested thoroughly before deployment for a polished user experience.

PROJECT SETUP AND CONFIGURATION

The project setup began with installing **Node.js** and **npm** to manage dependencies, along with **MongoDB** as the database solution. Development was carried out using **Visual Studio Code**, which served as the primary IDE. Two main directories, frontend (for the React-based user interface) and backend (for the Express.js server), were created to organize the project. Each directory was initialized using `npm init` to generate package.json files.

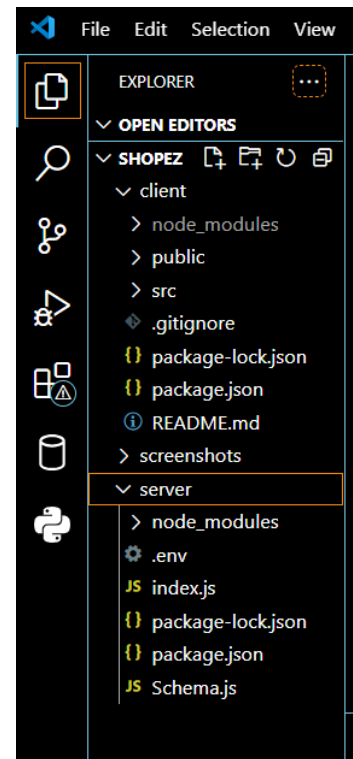
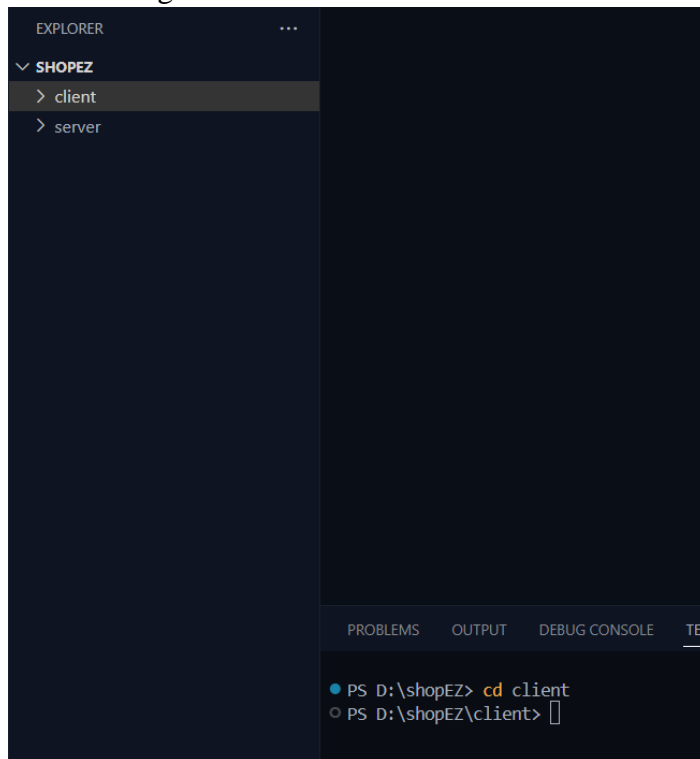
1. Install required tools and software:

- Node.js.
- Git.

2. Create project folders and files:

- Client folders.
- Server folders

Referral Image:



BACKEND DEVELOPMENT

1. Setup express server:

- Create index.js file.
- Create an express server on your desired port number.
- Define API's

Set Up Project Structure:

- Create a new directory for your project and set up a package.json file using the npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.

Reference Images:

The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure: SHOPEZ, client, server, node_modules, package-lock.json, and package.json. The main editor area shows the content of package.json, which includes fields for name, version, description, main, scripts, keywords, author, license, and dependencies. The dependencies section lists bcrypt, body-parser, cors, dotenv, express, and mongoose with their respective version constraints. Below the editor, the TERMINAL tab is active, showing the command 'npm install express mongoose body-parser dotenv' and its output, which includes the number of packages added, audited, and the time taken, as well as a note about vulnerabilities.

```
server > {} package.json > {} dependencies
1  {
2    "name": "server",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "bcrypt": "^5.1.1",
14     "body-parser": "^1.20.2",
15     "cors": "^2.8.5",
16     "dotenv": "^16.4.5",
17     "express": "^4.19.1",
18     "mongoose": "^8.2.3"
19   }
20 }
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS D:\shopEZ\server> npm install express mongoose body-parser dotenv

added 85 packages, and audited 86 packages in 11s

14 packages are looking for funding
run `npm fund` for details

found 0 vulnerabilities

● PS D:\shopEZ\server> npm i bcrypt cors

added 61 packages, and audited 147 packages in 9s

The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure: SHOPEZ, client, server, node_modules, JS index.js, package-lock.json, and package.json. The main editor area shows the content of index.js, which imports express, creates an app, uses express.json(), and listens on port 3001. Below the editor, the TERMINAL tab is active, showing the commands 'cd server' and 'node index.js', and the output 'App server is running on port 3001'.

```
server > JS index.js > ...
1  import express from "express";
2
3  const app = express();
4  app.use(express.json());
5
6  app.listen(3001, () => {
7    console.log("App server is running on port 3001");
8  });
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS D:\shopEZ> cd server

○ PS D:\shopEZ\server> node index.js

App server is running on port 3001

2. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for admin, users, products, orders and other relevant data.

3. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

4. Define API Routes:

- Create separate route files for different API functionalities such as users, orders, and authentication.
- Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

5. Implement Data Models:

- Define Mongoose schemas for the different data entities like products, users, and orders.
- Create corresponding Mongoose models to interact with the MongoDB database.
 - Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

7. Handle new products and Orders:

- Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.
- Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

- Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.
- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

FRONTEND DEVELOPMENT

1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

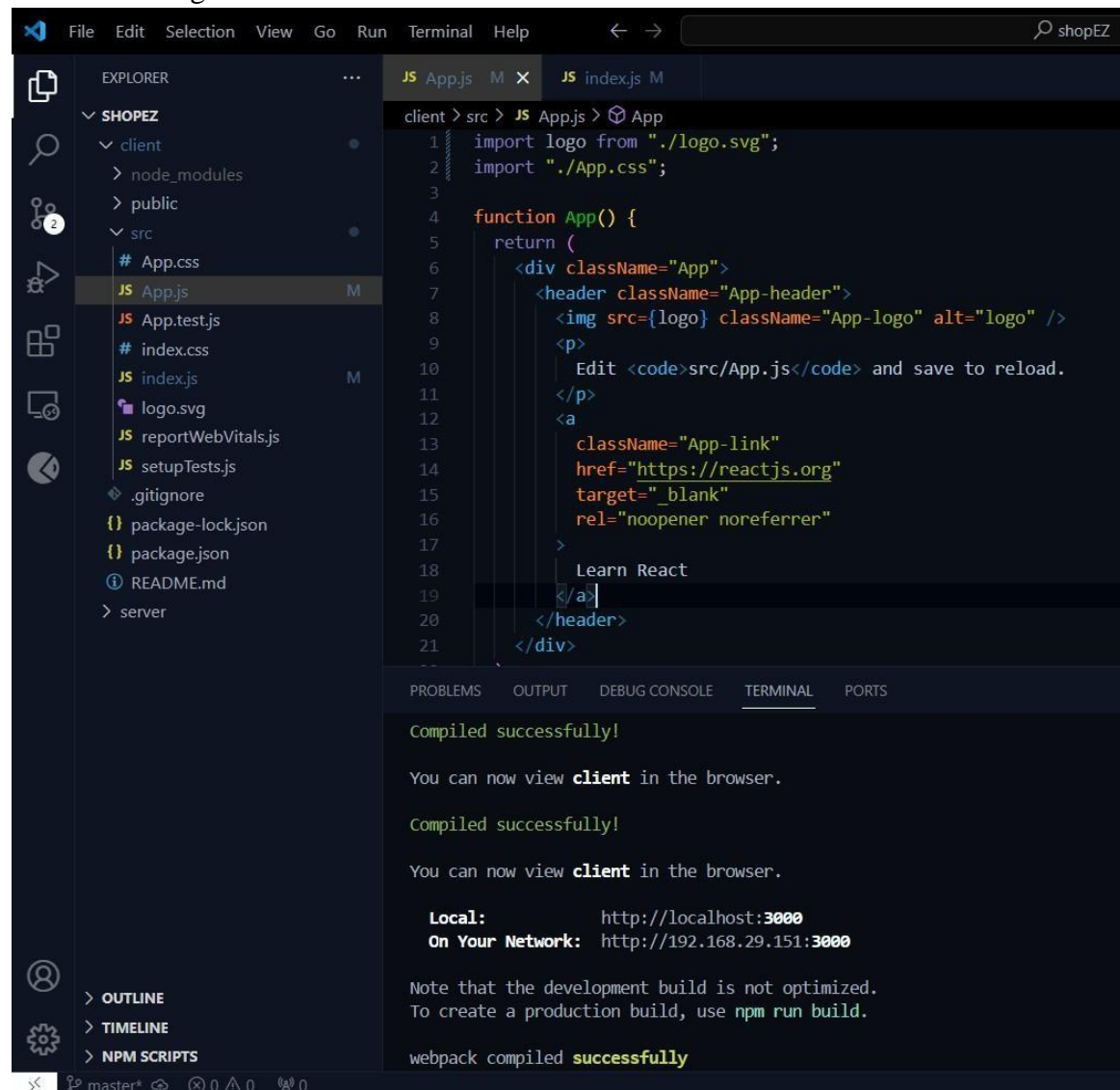
2. Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

3. Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Reference Image:

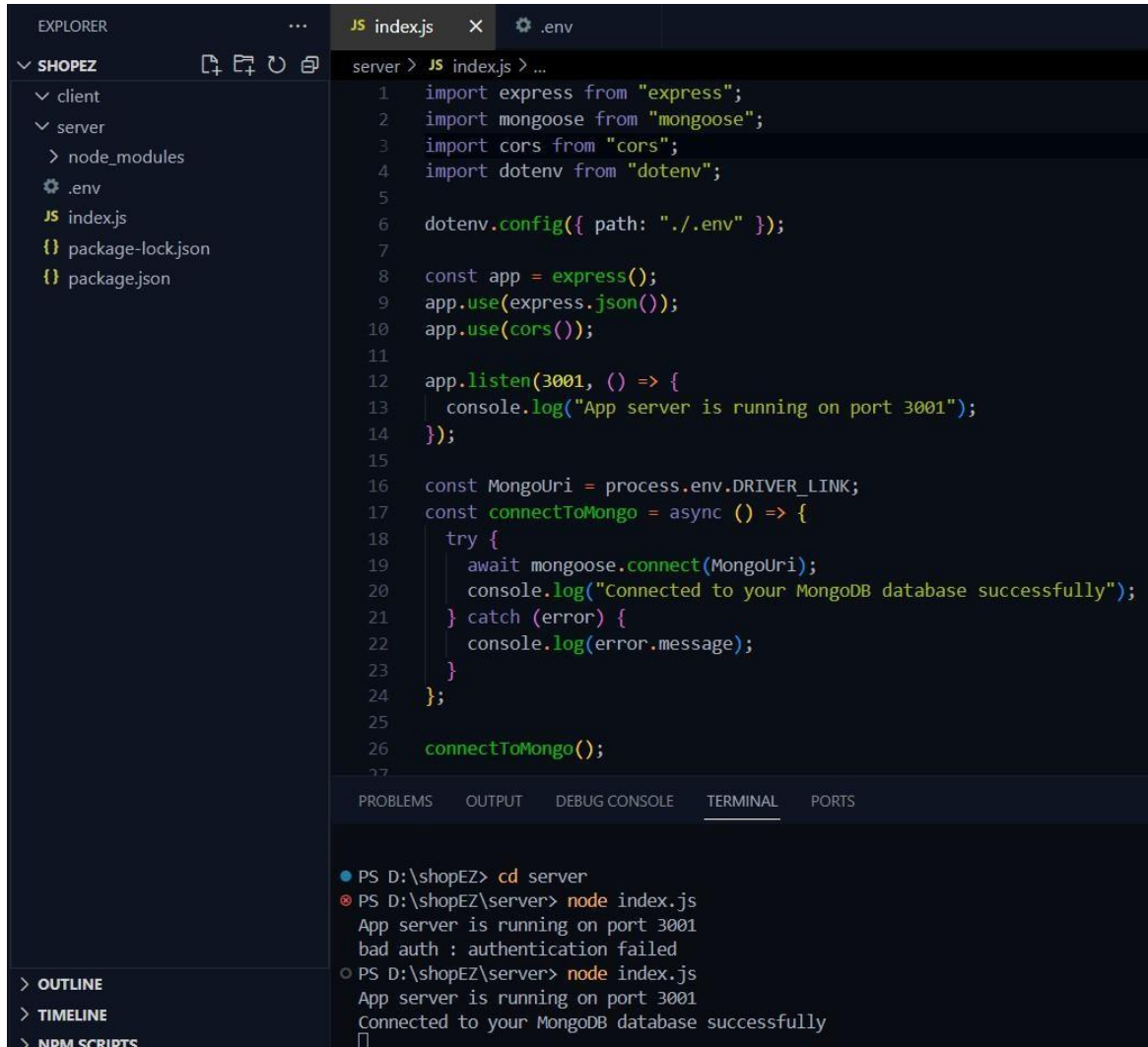


DATABASE DEVELOPMENT

Create database in cloud

- Install Mongoose.
- Create database connection.

Reference Image:



```
server > JS index.js > ...
1  import express from "express";
2  import mongoose from "mongoose";
3  import cors from "cors";
4  import dotenv from "dotenv";
5
6  dotenv.config({ path: "./.env" });
7
8  const app = express();
9  app.use(express.json());
10 app.use(cors());
11
12 app.listen(3001, () => {
13   console.log("App server is running on port 3001");
14 });
15
16 const MongoUri = process.env.DRIVER_LINK;
17 const connectToMongo = async () => {
18   try {
19     await mongoose.connect(MongoUri);
20     console.log("Connected to your MongoDB database successfully");
21   } catch (error) {
22     console.log(error.message);
23   }
24 };
25
26 connectToMongo();
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS D:\shopEZ> cd server
● PS D:\shopEZ\server> node index.js
App server is running on port 3001
bad auth : authentication failed
○ PS D:\shopEZ\server> node index.js
App server is running on port 3001
Connected to your MongoDB database successfully
```

SCHEMA USE-CASE:

1. User Schema:

- Schema: userSchema
- Model: 'User'
- The User schema represents the user data and includes fields such as username, email, and password.
- It is used to store user information for registration and authentication purposes.
- The email field is marked as unique to ensure that each user has a unique email address

2. Product Schema:

- Schema: productSchema
- Model: 'Product'
- The Product schema represents the data of all the products in the platform.
- It is used to store information about the product details, which will later be useful for ordering

3. Orders Schema:

- Schema: ordersSchema
- Model: 'Orders'
- The Orders schema represents the orders data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,
- It is used to store information about the orders made by users.
- The user Id field is a reference to the user who made the order.

4. Cart Schema:

- Schema: cartSchema
- Model: 'Cart'
- The Cart schema represents the cart data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,
- It is used to store information about the products added to the cart by users.
- The user Id field is a reference to the user who has the product in cart.

5. Admin Schema:

- Schema: adminSchema
- Model: 'Admin'
- The admin schema has essential data such as categories, banner.

6. Restaurant Schema:

- Schema: restaurantSchema
- Model: 'Restaurant'
- The restaurant schema has the info about the restaurant and it's menu

EXECUTION AND DEPLOYMENT:

The execution of the SB Food Ordering App begins with local development and testing of individual components. The frontend, built with React.js, is executed using npm start, enabling developers to test the UI and functionality. The backend, powered by Node.js and Express.js, is run using npm run dev or similar scripts, with tools like Postman used for API testing. MongoDB is connected via Mongoose, with database operations validated during development. Unit tests and integration tests are performed to ensure reliability and performance.

For deployment, the backend is hosted on platforms like **Heroku** or **AWS**, and the frontend is deployed on **Netlify** or **Vercel**. The MongoDB database is hosted on **MongoDB Atlas** for scalability and security. Environment variables such as API keys and database credentials are managed securely using .env files. Continuous Integration/Continuous Deployment (CI/CD) pipelines are implemented using tools like GitHub Actions to automate testing and deployment processes, ensuring the application is always up-to-date and accessible to users.

TESTING AND DEPLOYMENT

TESTING

Testing is a crucial phase in the development of any application to ensure that all functionalities are working as expected and to prevent any potential issues from reaching end-users. The testing of the SB foods platform was comprehensive, including both manual and automated methods. This phase validated the functionality, security, and usability of the system.

Unit Testing

Unit testing focused on verifying that individual components of the application functioned correctly in isolation. The SB Food Ordering App undergoes a systematic testing and deployment process to ensure functionality, security, and performance in production. Here's the detailed breakdown:

Frontend:

- Individual components like navigation bars, product cards, and cart functionalities are tested using **Jest** or **React Testing Library**.
- Example: Validating that the cart component correctly updates when items are added or removed.

Backend:

- API endpoints are tested using **Mocha** or **Chai** to ensure they return correct responses.
- Example: Verifying user registration endpoint returns success and error messages appropriately.

Integration Testing

- Tests the interaction between frontend, backend, and database to ensure smooth data flow.
- Example: Submitting an order from the frontend updates the database and reflects correctly in the admin dashboard.
- Tools like **Postman** and **Supertest** are used to verify API requests and responses.

UI/UX Testing

- Conducted to ensure that the user interface is responsive across various devices and browsers (mobile, tablet, desktop).
- Validates user workflows like login, menu browsing, and order placement.

Automated Testing:

- Tools like **Selenium** or **Cypress** are used to simulate user behavior and detect interface issues.

Performance Testing

- Evaluates the app's behavior under load and stress.
- Example: Simulating 1,000 concurrent users placing orders to measure response time and database performance.
- Tools: **Apache JMeter** or **Loader.io**.

End-to-End Testing

- Simulates real user scenarios from start to finish.
- Example: A user logs in, browses the menu, adds items to the cart, checks out, and tracks the order.
- Tools: **Cypress**, **Puppeteer**, or **TestCafe**.

Security Testing

- Validates the app's protection against vulnerabilities.
- Example: Ensuring JWT tokens are properly encrypted and API endpoints are protected from unauthorized access.
- Tools: **OWASP ZAP** or **Burp Suite**.

DEPLOYMENT

Backend Deployment

Hosting Platform:

- Use **Heroku**, **AWS Elastic Beanstalk**, or **Render** for scalable and reliable hosting of the Node.js backend.
- MongoDB is hosted on **MongoDB Atlas** for secure and distributed data management.

Steps:

- Push the backend code to a version control repository like GitHub.
- Set up deployment pipelines or manual deployment processes.
- Configure environment variables (e.g., database URI, API keys) using platform tools like Heroku Config Vars.
- Test the backend deployment using API testing tools like Postman.

Frontend Deployment

Hosting Platform:

- Deploy the React.js frontend on **Netlify**, **Vercel**, or **AWS S3 + CloudFront** for fast and secure delivery.

Steps:

- Generate a production build using `npm run build` to optimize assets for deployment
- Upload the build folder to the hosting platform.
- Configure routing rules for single-page application (SPA) navigation.
- Link the frontend with the deployed backend by updating API endpoints.
- Test the live app for responsiveness and feature functionality.

Domain Setup and SSL

- Register a custom domain using platforms like **GoDaddy** or **Google Domains**.
- Configure DNS settings to point to the frontend and backend servers.
- Enable **SSL/TLS certificates** (usually provided by hosting platforms) to secure user data with HTTPS.
- **Continuous Integration/Continuous Deployment (CI/CD)**
- **Tools Used:** GitHub Actions, Jenkins, or GitLab CI/CD.

Process:

1. Automate the testing pipeline: Code is tested automatically when pushed to the repository.
2. Automate deployment: Upon passing tests, the app is deployed to staging or production environments.
3. Rollback Mechanism: In case of deployment failure, previous versions can be restored automatically.

Monitoring and Maintenance

- **Monitoring Tools:** Use **New Relic** or **Datadog** for real-time performance tracking.
- **Error Tracking:** Implement tools like **Sentry** or **LogRocket** to capture errors and debug issues.
- **Updates and Patches:** Regularly update dependencies and perform security patches to keep the app secure and efficient.

OUTPUT SCREENSHOTS:

Fig.1: Home page

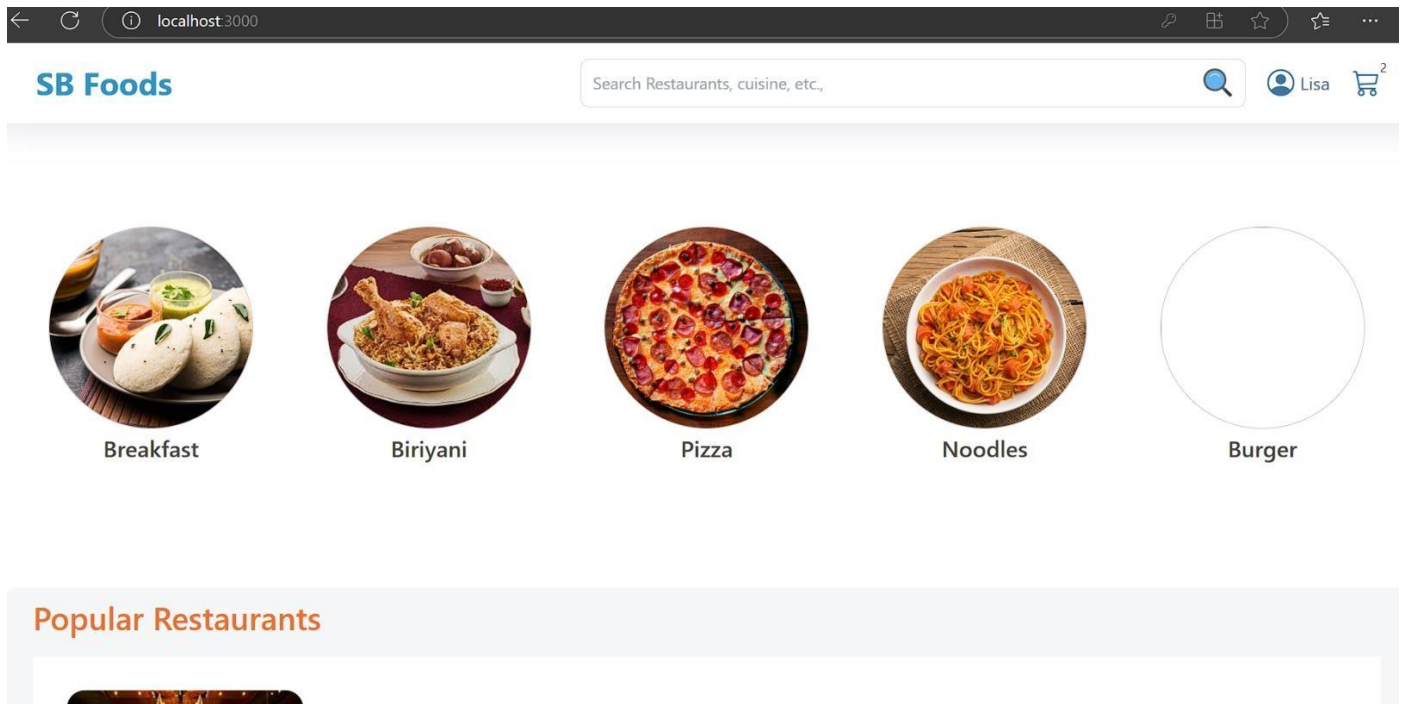


Fig.2: Restaurant Admin Page

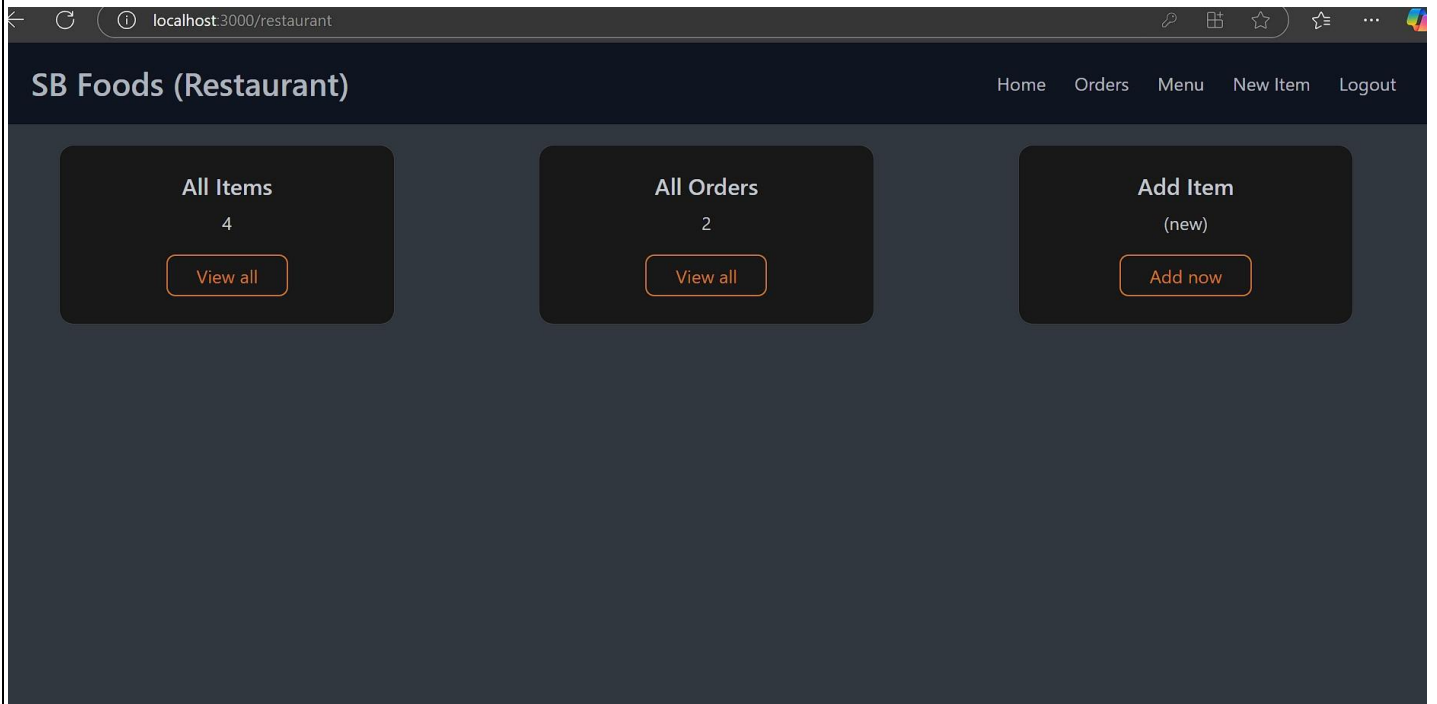


Fig.3: Cart

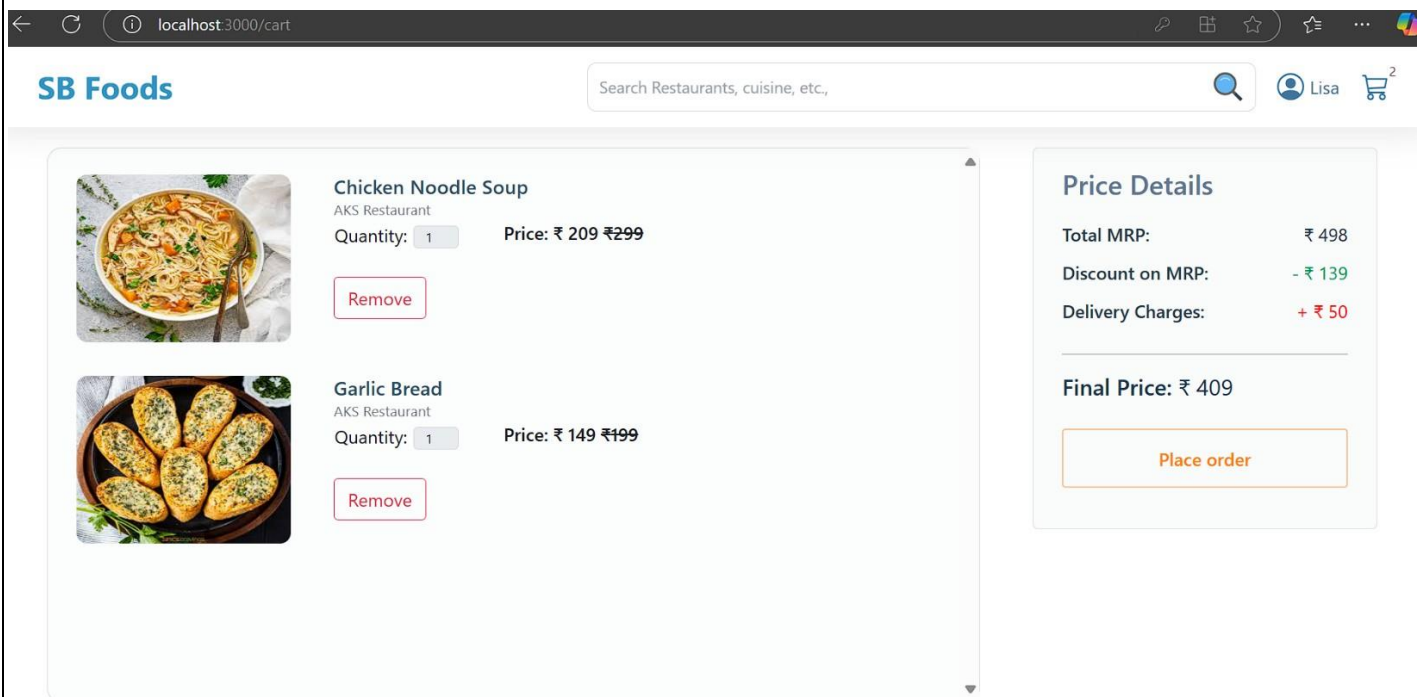



Fig.4: Order

SB Foods


Search Restaurants, cuisine, etc.,

Username: Lisa
Email: lisa@gmail.com
Orders: 2
Logout

Orders



Garlic Bread
AKS Restaurant
Quantity: 1 Total Price: ₹ 149 ₹199 Payment mode: cod
Ordered on: 2024-11-23 Time: 07:59 status: delivered



Chicken Noodle Soup
AKS Restaurant
Quantity: 1 Total Price: ₹ 209 ₹299 Payment mode: cod
Ordered on: 2024-11-23 Time: 07:59 status:

Fig.5: Restaurant Page

SB Foods

Search Restaurants, cuisine, etc.,

AKS Restaurant

Kodambakkam, Chennai

All Items

Filters

Sort By


- ☐ Popularity
- ☐ low-price
- ☐ high-price
- ☐ Discount
- ☐ Rating

Food Type


- ☐ Veg
- ☐ Non Veg
- ☐ Beverages

Categories


- ☐ Non Veg




Chicken Noodle Soup
₹ 209 299
Add item



Garlic Bread
₹ 149 199
Add item

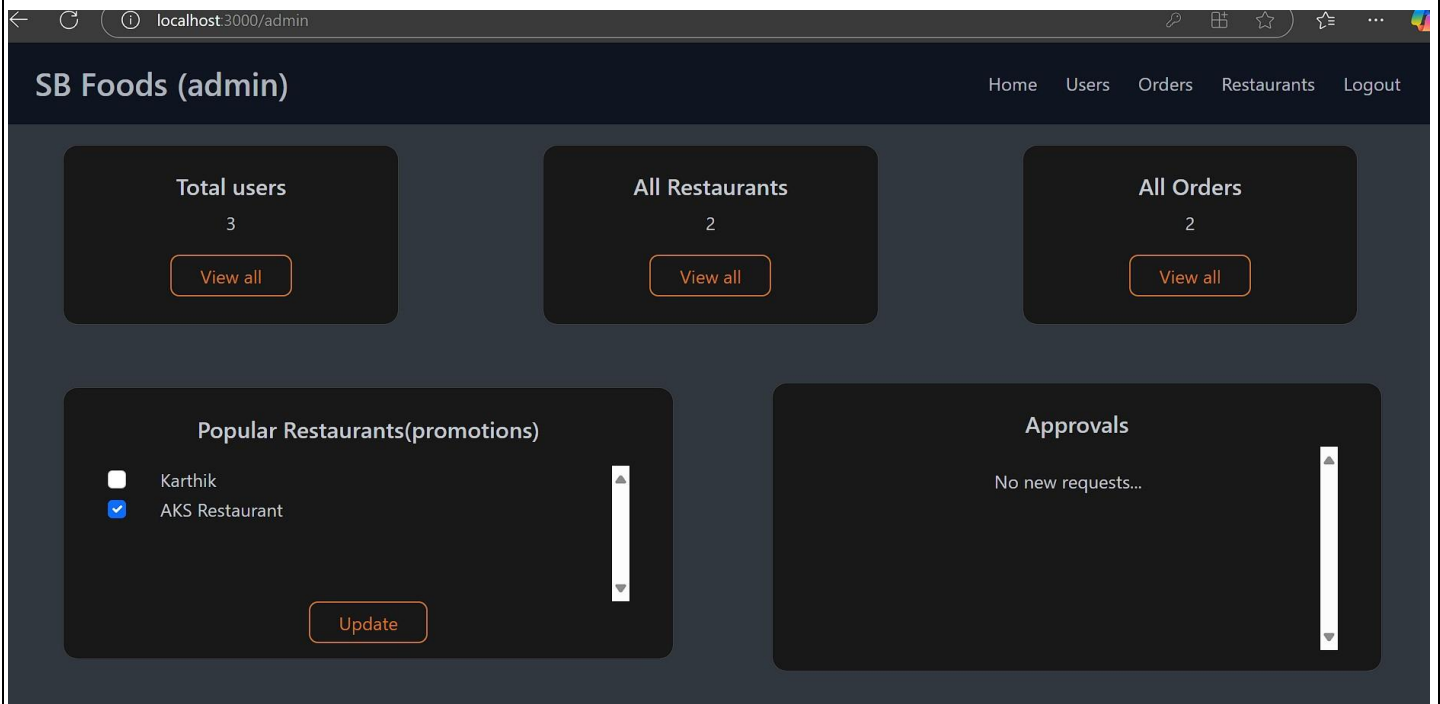


Veg Noodle Soup
₹ 149 299
Add item



Chicken Burger
₹ 239 299
Add item

Fig.6: Admin Page



CHALLENGES AND SOLUTIONS:

The development of the **SB Food Ordering App** involved several challenges that required innovative solutions to ensure the app's functionality, performance, and scalability. Here are some key challenges faced during development, along with the solutions implemented:

1. Scalability of the Application

Challenge:

As the app's user base grows, the ability to scale both the frontend and backend to handle increased traffic and data was a major concern. The app needed to handle hundreds of concurrent users placing orders without slowing down.

Solution:

Backend: The backend was built using **Node.js** and **Express.js**, which is known for its scalability. To further ensure scalability, the app was hosted on platforms like **AWS** and **Heroku** that support auto-scaling based on traffic volume.

Database: **MongoDB Atlas** was used to provide scalable database services that automatically scale as the number of users and data grows. Sharding and indexing were implemented to optimize database performance under heavy loads.

2. Real-Time Order Management

Challenge:

Managing and updating orders in real-time was crucial for both customers and restaurant admins. Ensuring that orders reflect updates immediately (e.g., order status, payment status) for both parties proved complex.

Solution:

WebSockets: **Socket.io** was implemented for real-time communication between the client and server. This allowed the app to push live updates about order statuses (e.g., "Preparing," "Out for Delivery") to customers and restaurant admins instantly without the need for refreshing the page.

Polling: For users who may not have WebSocket support, periodic polling was implemented to check for new order status updates.

3. Secure Payment Integration

Challenge:

Ensuring secure and smooth payment processing was critical. Handling sensitive user data, including payment information, required stringent security measures to prevent fraud and breaches.

Solution:

Payment Gateway Integration: The app integrates with secure payment gateways like **Razorpay**, **Stripe**, or **PayPal** that provide encrypted transactions and secure processing of payments.

SSL Encryption: The app uses **SSL/TLS certificates** to ensure that all communication between the frontend and backend is encrypted. **JWT** authentication is used to secure user sessions.

PCI-DSS Compliance: The payment systems comply with the **Payment Card Industry Data Security Standard (PCI-DSS)** to ensure safe handling of credit card information.

4. User Authentication and Role Management

Challenge:

The app requires different levels of access for users (customers) and admins (restaurant operators), and managing user roles securely posed a significant challenge.

Solution:

JWT Authentication: Users are authenticated using **JWT (JSON Web Tokens)**, which ensures that only authorized users can access certain features.

Role-Based Access Control (RBAC): The app was designed with role-based access control, so users and admins have different privileges and functionalities. Customers can place orders and track them, while admins can manage menus, view analytics, and update order statuses.

OAuth Integration (Planned): For future expansion, integrating OAuth 2.0 (e.g., Google or Facebook login) will make the login process even more secure and convenient for users.

5. Maintaining Data Consistency Between Frontend and Backend

Challenge:

Ensuring that data displayed on the frontend (e.g., order history, menu items) is always consistent with the backend was a persistent issue, especially with real-time updates and changing inventory.

Solution:

State Management: React's **Context API** and **Redux** were used for efficient state management, ensuring that the frontend always reflects the current data from the backend.

RESTful APIs: The backend uses **RESTful APIs** to handle CRUD operations for orders, users, and products. Any changes in the backend are instantly updated and synchronized with the frontend.

Caching: For frequently accessed data, such as product listings, caching strategies using **Redis** are employed to reduce load on the database and enhance performance.

6. Ensuring Cross-Platform Compatibility

Challenge:

The app needed to function seamlessly on various devices and browsers, from desktops to mobile phones, without losing its performance or user experience.

Solution:

Responsive Design: The app was built using **CSS Flexbox** and **Grid** for responsive layouts, ensuring compatibility across screen sizes.

Cross-Browser Compatibility: Tools like **BrowserStack** were used for testing and ensuring that the app works correctly on popular browsers (Chrome, Firefox, Safari, Edge).

Mobile Optimization: The frontend was optimized for mobile, ensuring fast load times and usability on devices with varying screen sizes and resolutions.

7. Managing High Traffic and Server Load

Challenge:

Handling high traffic during peak hours (such as lunch and dinner times) posed a significant challenge, especially for large restaurants with thousands of users placing orders simultaneously.

Solution:

Load Balancing: The app was deployed with load balancing to distribute traffic across multiple servers, reducing the risk of downtime and slow performance.

Auto-Scaling: Platforms like **AWS EC2** provide auto-scaling capabilities, automatically adding resources to the app during high traffic and scaling down during off-peak hours.

Caching and Content Delivery Networks (CDNs): Static content (images, product data) is cached and served via a **Content Delivery Network (CDN)** like **Cloudflare** to reduce server load and improve response times.

8. Data Backup and Recovery

Challenge:

Ensuring that user data, including orders and payment details, is safely backed up in case of server failures or crashes.

Solution:

Regular Backups: Automated backups of the MongoDB database are scheduled at regular intervals, with daily and weekly backups stored on **AWS S3** or another secure cloud storage solution.

Disaster Recovery Plan: A disaster recovery plan is in place to restore data quickly in case of any system failure, ensuring minimal downtime.

These challenges were met with a combination of best practices, cutting-edge technologies, and thoughtful solutions to ensure that the SB Food Ordering App functions optimally, securely, and efficiently at scale.

FUTURE ENHANCEMENTS:

To stay competitive and continue providing an exceptional user experience, several future enhancements can be implemented in the **SB Food Ordering App**. These improvements will focus on increasing functionality, user engagement, and operational efficiency.

1. AI-Based Personalization

Enhancement:

Implement AI-powered recommendation systems to provide personalized product suggestions based on user behavior, order history, and preferences. This could increase order frequency and improve customer satisfaction. **Details:**

Machine Learning Models can analyze user activity and suggest meals or special offers.

Personalization could also extend to dynamic menus, adjusting based on the time of day, user preferences, or local trends.

2. Multi-Language and Multi-Currency Support

Enhancement:

Expand the app's reach by introducing multi-language and multi-currency support, making it accessible to a wider audience, especially in diverse regions. **Details:**

Language Support: Add local language options for users in different regions (e.g., Hindi, Tamil, French, Spanish) to cater to non-English speaking customers.

Currency Support: Allow users to pay in their local currency, making the app accessible globally.

3. Voice-Activated Ordering

Enhancement:

Integrate voice-assisted ordering features, where users can place orders using voice commands. This enhances user convenience, particularly for hands-free use while cooking or on the go. **Details:**

Voice Recognition: Integrate with platforms like **Google Assistant** or **Amazon Alexa** to enable voice-based searches and orders.

Hands-Free UI: Voice prompts will guide users through browsing the menu and completing orders without touching the device.

4. Advanced Analytics and Reporting for Admins

Enhancement:

Provide detailed insights and analytics to restaurant admins and business owners to help them understand sales trends, customer preferences, and inventory usage. **Details:**

AI-Powered Analytics: Use machine learning algorithms to predict demand, identify peak hours, and provide actionable insights for business growth.

Real-Time Reporting: Admins can get real-time reports on order volumes, sales, and even customer feedback to make data-driven decisions.

5. Subscription-Based Meal Plans

Enhancement:

Introduce subscription-based meal plans where users can opt for daily, weekly, or monthly meal deliveries at discounted rates. **Details:**

Subscription Tiers: Offer various subscription options based on meal frequency (e.g., 3 meals/week, 5 meals/week).

Discounts and Offers: Provide users with discounted rates or exclusive access to new menu items as part of their subscription.

6. Integration with IoT for Smart Kitchens

Enhancement:

Integrate with **Internet of Things (IoT)** devices in partner kitchens, allowing real-time updates about meal preparation, cooking times, and delivery status. **Details:**

Real-Time Monitoring: Allow users to see how long it will take for their food to be prepared, with notifications when it's ready to be dispatched.

Smart Kitchen Features: Kitchen staff can receive real-time updates about orders and cooking instructions through IoT-enabled devices, optimizing the food preparation process.

7. Loyalty Programs and Gamification

Enhancement:

Introduce a gamification feature that rewards customers with points, badges, or discounts based on their ordering activity. **Details:**

Loyalty Points System: Users earn points for every order, which can be redeemed for discounts, free items, or exclusive offers.

Gamified Experience: Users can unlock achievements (e.g., "10 orders completed," "Early bird special") to encourage more frequent interactions and engagement.

8. Integration with Delivery Partners and Tracking

Enhancement:

Enhance order tracking by integrating directly with popular delivery services (like Uber Eats, Swiggy, or Zomato) for real-time tracking and status updates. **Details:**

Unified Tracking System: Users can track their orders in real-time within the app, providing visibility from preparation to delivery.

Delivery Optimization: Work with delivery partners to optimize routes and delivery times, ensuring faster and more efficient deliveries.

9. Improved Customer Support with Chatbots

Enhancement:

Implement a **24/7 chatbot** powered by AI to assist users with common queries, order status, and troubleshooting in real-time. **Details:**

AI Chatbot: Train the bot to handle FAQs, track orders, and even suggest products, improving response times and reducing the load on customer service.

Live Chat Integration: When needed, users can seamlessly transition from the chatbot to a live support agent for more complex issues.

10. Augmented Reality (AR) Menu Previews

Enhancement:

Incorporate **Augmented Reality (AR)** to allow users to see a 3D preview of dishes before ordering. This could improve decision-making and add an engaging, interactive layer to the app. **Details:**

Users can point their camera at the screen and view virtual models of dishes to get a better idea of what the food will look like.

This feature can help users make more informed choices and increase engagement with the app.

11. Environmental Sustainability Features

Enhancement:

Promote eco-friendly practices by allowing users to choose sustainable packaging options and providing information on the carbon footprint of their orders. **Details:**

Sustainable Packaging Options: Users can opt for eco-friendly packaging at checkout, and restaurants can promote their sustainability practices.

Carbon Footprint Tracker: Show users the environmental impact of their order, encouraging more eco-conscious choices.

12. Social Media Integration and Sharing

Enhancement:

Allow users to share their meals, orders, or experiences directly to social media platforms (e.g., Instagram, Facebook) with in-app social sharing buttons. **Details:**

Users can share photos of their meals, write reviews, and tag the restaurant, boosting app visibility through word of mouth.

Referral Program: Implement a referral program where users can earn discounts or rewards by inviting friends to use the app.

CONCLUSION:

The SB Food Ordering App is a robust and scalable solution that leverages the power of the MERN stack to deliver a seamless and efficient food ordering experience for both customers and restaurants. From its user-friendly interface and secure authentication to real-time order tracking and comprehensive admin management tools, the app successfully meets the needs of modern consumers and restaurant operators.

By implementing a structured testing and deployment process, the app ensures reliability, performance, and security. Through rigorous unit, integration, and performance testing, the app's functionality is thoroughly validated. The deployment process, utilizing popular platforms such as Heroku, AWS, and Netlify, guarantees that the app is accessible, fast, and secure across different environments.

With its focus on user satisfaction and operational efficiency, the SB Food Ordering App is well-positioned to meet the growing demand in the online food delivery market. Future enhancements, including AI-driven recommendations and multi-language support, will further elevate the user experience, making the app a powerful tool for both users and businesses.

REFERENCES

1. **MERN Stack Documentation** - MongoDB, Express, React, Node.js: <https://www.mongodb.com/mern-stack>
2. **JWT Authentication in Node.js**: <https://jwt.io/introduction/>
3. **Mongoose Documentation**: <https://mongoosejs.com/docs/>
4. **React Documentation**: <https://reactjs.org/docs/getting-started.html>
5. **MongoDB Atlas Documentation**: <https://www.mongodb.com/cloud/atlas>

These references provide detailed guides and documentation for the technologies and tools used to build, deploy, and optimize the ShopEZ platform.

