MADRAS INSTITUTE OF

TECHNOLOGYANNA UNIVERSITY

CHENNAI-600044

DEPARTMENT OF INFORMATION TECHNOLOGY

NAAN MUDHALVAN

MERN STACK POWERED BY MONGODB

TITLE

SB FOODS-FOOD ORDERING SYSTEM

Team Members

Abinaya V (2021506003)

Divya Dharshini K (2021506019)

Elakiya I (2021506021)

Krupa Janani G (2021506040)

Project Overview

In this project, we've created a food delivery application that allows users to browse restaurants, view menus, and add items to a cart. The application utilizes the MERN stack, leveraging MongoDB for data management, Express and Node.js for the backend server, and React for the frontend.

Tech Stack

1. **Frontend**: React with Context API, functional components, and libraries like axios and react-router-dom.
2. **Backend**: Node.js and Express.js with mongoose for MongoDB integration.
3. **Database**: MongoDB to store restaurant, menu, and cart data.

Frontend (React)

1. **Project Initialization**:
    - After creating the frontend project with npx create-react-app client, the necessary dependencies are installed:
2. **Folder Structure**:
    - **/src**:
        - **components/**: Contains UI components like RestaurantList, RestaurantCard, DishesMenu, DishCard, and Cart.
        - **contexts/**: Holds RestaurantContext.js, which provides shared state management across the app.
        - **App.js**: Main application component, which uses React Router to navigate between screens (like the restaurant list and cart).
        - **index.js**: Wraps the app in the RestaurantProvider, allowing access to the context throughout.
3. **Context API Setup**:
    - **RestaurantContext**:
        - Defines global state management for restaurant and cart data.
        - Enables components to access and update shared states like the selected restaurant, menu items, and cart contents.
        - Uses React.createContext() to initialize and useContext to access data within components.

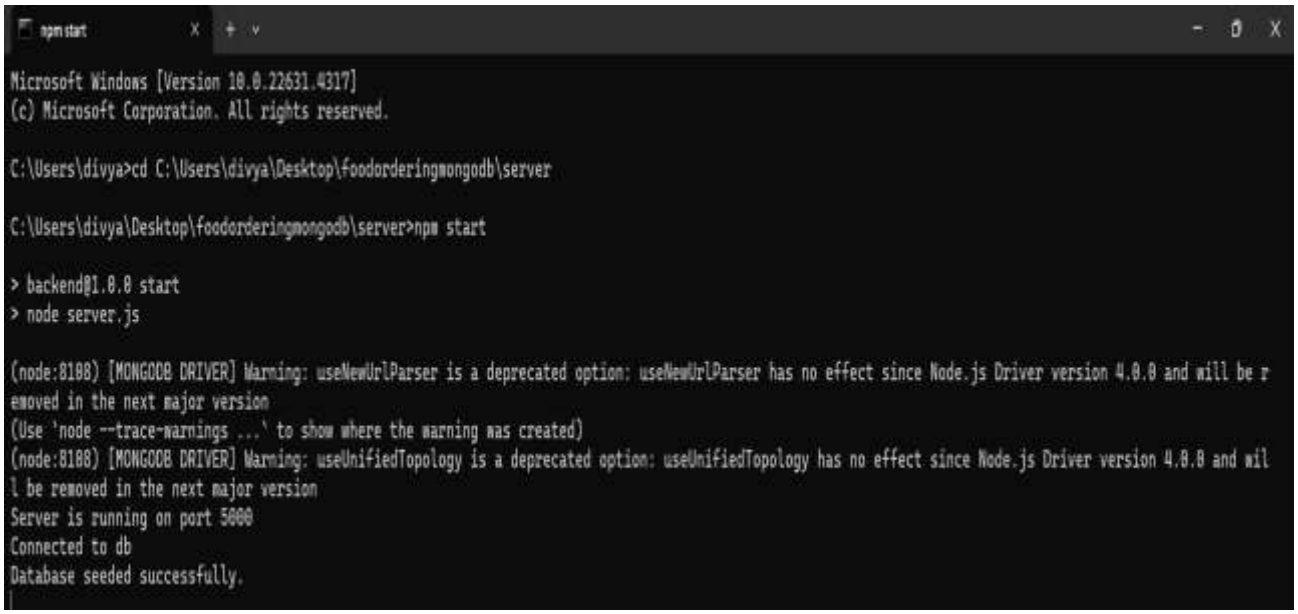
4. **Component Structure**:
    - **RestaurantList**:
        - Displays a list of all available restaurants, using RestaurantCard to render individual items.
    - **RestaurantCard**:
        - Shows details about each restaurant (e.g., name, location).
        - Clicking a restaurant could trigger navigation to the restaurant's menu page.
    - **DishesMenu**:
        - Shows the dishes available for the selected restaurant.
        - Uses DishCard to render individual dishes, with an "Add to Cart" button.
    - **Cart**:
        - Lists items added to the cart and provides options to increase/decrease item quantities or remove items.
        - Total cost calculation is handled here.
    - **App.js**:
        - Routes are managed here, linking pages like the restaurant list, menu, and cart.
        - BrowserRouter and Route from react-router-dom provide navigation.

```
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Backend (Node.js + Express)

1. **Project Initialization**:
    - Set up the backend by creating a folder and initializing a package with npm init -y, then installing required dependencies:

2. **Folder Structure**:
    - **server.js**: Main server file where Express is configured.
    - **/models**: Contains Mongoose models like Restaurant.js and Dish.js, which define schemas for storing data in MongoDB.
    - **/routes**: Contains route files (e.g., restaurantRoutes.js, menuRoutes.js), which define endpoints for fetching and manipulating restaurant, menu, and cart data.
    - **/controllers**: Optional, but recommended. Contains logic for handling requests (e.g., fetching restaurant data or handling cart actions).
3. **Backend Dependencies**:
    - **cors**: Middleware for enabling cross-origin requests from the frontend.
    - **express**: Framework for creating server and handling routes.
    - **mongoose**: ODM for MongoDB, making data interaction more efficient.
    - **nodemon**: Development tool for automatically restarting the server on file changes.



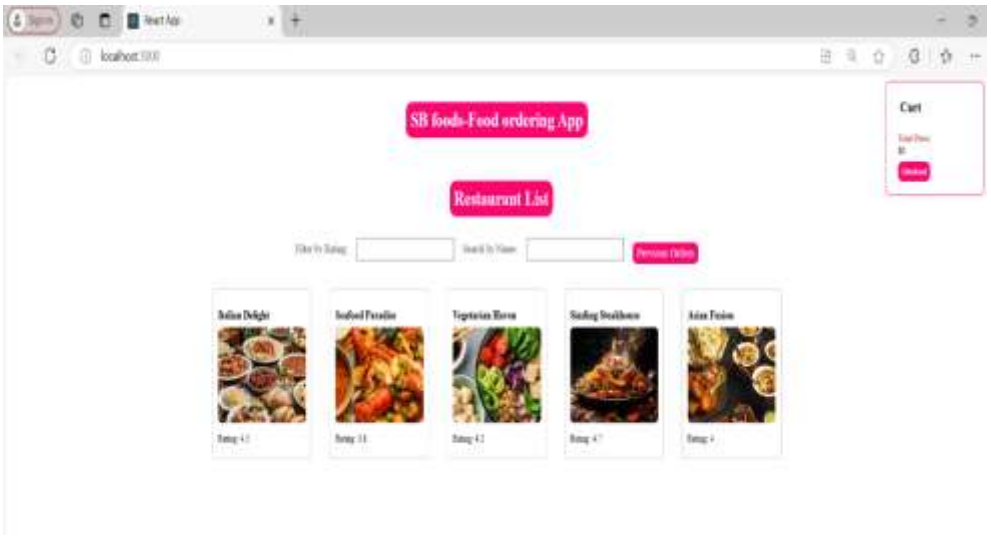
Application Walkthrough

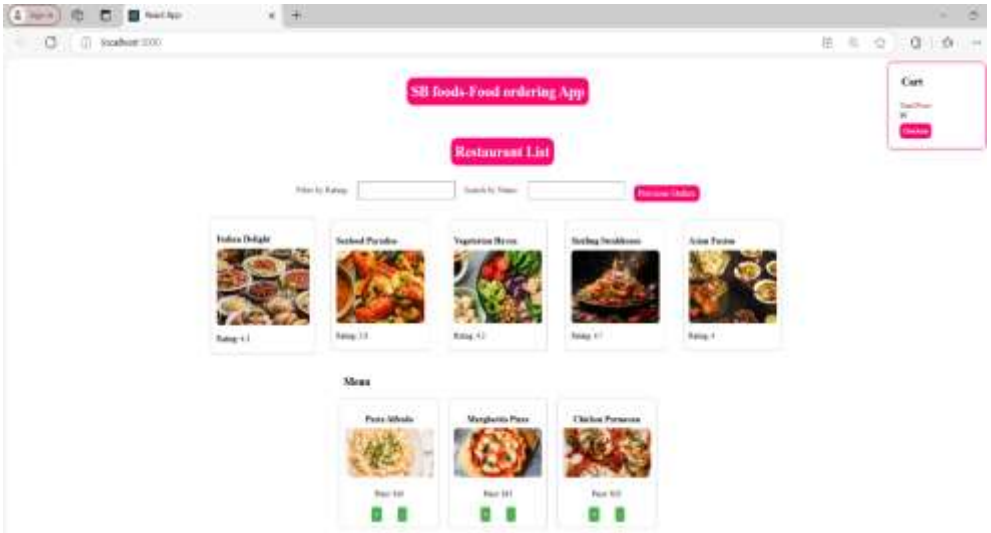
1. **Index File Setup**:
    - index.js initializes the app, wrapping it in RestaurantProvider, giving all components access to the restaurant context for shared state management.
2. **Backend API Structure**:
    - **Restaurant and Dish APIs**: Endpoints are likely defined to retrieve restaurant lists, menu details, and to add or update items in the cart.
    - **Database Schema**: MongoDB collections are used to store restaurant and dish information.

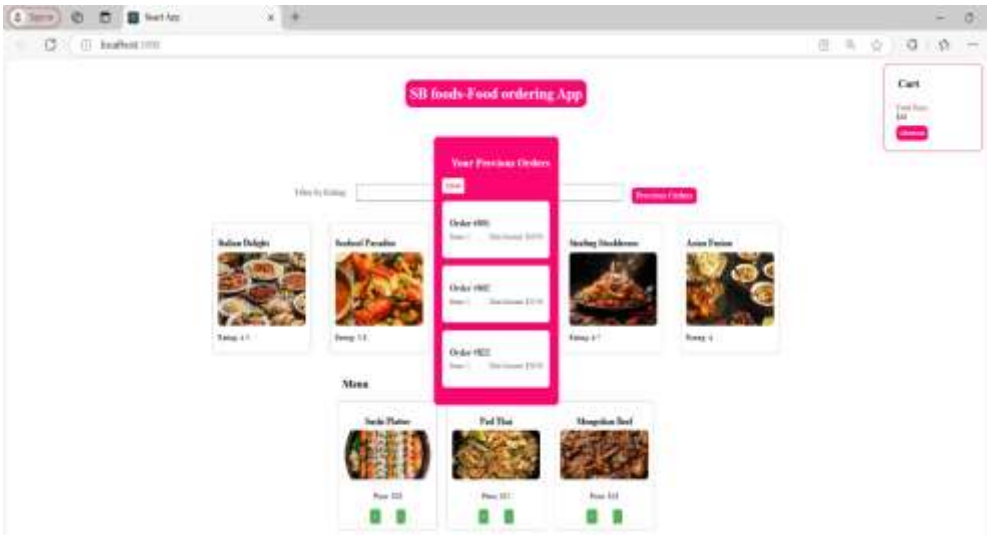
3. **Frontend Flow**:
    - **Restaurant List**: Displays all restaurants using RestaurantList and RestaurantCard components.
    - **Menu and Cart**: DishesMenu and DishCard allow users to view and add items to their cart.

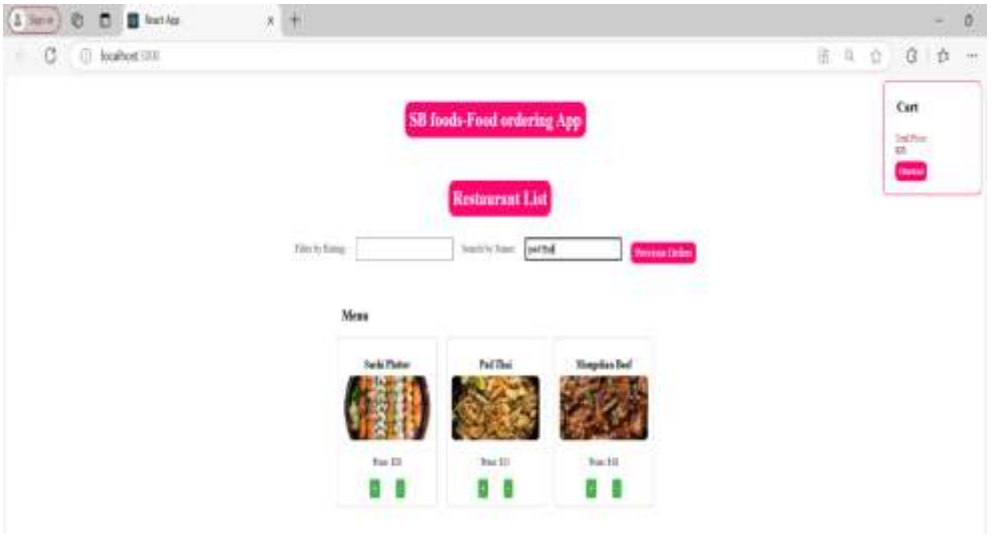
Outcome

- **Functionality**: Users can browse through restaurants, view menu items, and manage their cart with ease. The app efficiently handles state using Context API.
- **Backend**: Node.js server interfaces with MongoDB through Mongoose, ensuring smooth data storage and retrieval.









Conclusion

This project demonstrates a full-stack approach to building a dynamic, data-driven application using the MERN stack. The frontend's Context API offers an efficient way to manage state, while the backend uses a robust database model with Express and MongoDB, allowing seamless communication and data handling. This project provides a practical, scalable solution for a food delivery application.