AALIM MUHAMMED SALEGH COLLEGE OF ENGINEERING
Approved by All India Council for Technical Education - New Delhi, Affiliated to Anna University, Chennai
NAAC Accredited Institution
"Nizara Educational Campus", Muthapudupet, Avadi - IAF, Chennai - 600 055.
ANNA UNIVERSITY COUNSELLING CODE : 1101
NBA ACCREDITED COURSES (Mech Engg, ECE, CSE & IT)

MERN Stack SB Foods - Food Ordering App

Department of Information Technology

Submitted By

1.FIRNAS FATHIMA.A 110121205011

2.FURQANA.F 110121205012

3. RAHIMA SHIRIN.B 110121205045

4. SHOBITHA.P 110121205051

AALIM MUHAMMED SALEGH COLLEGE OF ENGINEERING
Approved by All India Council for Technical Education - New Delhi, Affiliated to Anna University, Chennai
NAAC Accredited Institution
"Nizara Educational Campus", Muthapudupet, Avadi - IAF, Chennai - 600 055.
ANNA UNIVERSITY COUNSELLING CODE : 1101
NBA ACCREDITED COURSES (Mech Engg, ECE, CSE & IT)

BONAFIDE CERTIFICATE

Certified that this project report on "**MERN STACK SB Foods - Food Ordering App**" is the Bonafide record of work done by Firnas Fathima.A (110121205011), Furqana. F(110121205012), Rahima Shirin.B(110121205045), Shobitha. P (110121205051).

From the Department of Information Technology by Anna University,Chennai

Internal Guide

Head of the Department

Internal Examiner

External Examiner

TABLE OF CONTENT

Abstract:

The Food Delivery App is a comprehensive, web-based platform developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js), designed to simplify and enhance the experience of ordering and managing food deliveries. As online food delivery grows increasingly popular, this project addresses the needs of modern consumers by providing a streamlined, accessible platform where users can browse menus, place orders, and enjoy food from a variety of restaurants.

The application includes core features such as user registration, restaurant browsing, detailed menu descriptions, and secure payment options. Upon placing an order, users can track their delivery in real-time, view order history, and manage their favorite restaurants and dishes. The use of React.js ensures a dynamic and responsive user interface, while Node.js and Express.js handle the backend logic and API requests. MongoDB serves as the database to store and retrieve user information, restaurant details, menu items, and order histories.

Together, these technologies offer a scalable and efficient solution for real-time data handling and seamless user interaction.

The Food Delivery App not only demonstrates the technical capabilities of the MERN stack but also provides a practical foundation that can be expanded to include additional features such as personalized restaurant recommendations, user reviews, and promotional offers. This application serves as a base for a scalable, user-friendly food delivery platform, catering to the demands of the growing online food delivery industry.

Introduction:

The Food Delivery App is a cutting-edge web-based platform designed to revolutionize the way people order and enjoy meals from their favorite restaurants. Built using the robust and versatile MERN stack (MongoDB, Express.js, React.js, and Node.js), the app offers a seamless, user-friendly experience for browsing menus, placing orders, and managing food deliveries. As online food delivery becomes an integral part of modern lifestyles, this application aims to meet the growing demand for convenience and variety in meal choices.

The app provides essential features such as user registration, restaurant and menu browsing, secure payment options, and real-time order tracking. Customers can explore diverse cuisines, customize their orders, and track their delivery status effortlessly. With a dynamic and responsive interface powered by React.js and a robust backend managed by Node.js and Express.js, the application ensures smooth and efficient performance. MongoDB, as the database, securely stores user profiles, restaurant details, menu items, and order histories, ensuring scalability and reliability.

Designed to cater to the evolving needs of the food delivery industry, the app offers a foundation for future enhancements, including personalized restaurant recommendations, loyalty programs, and exclusive deals. By combining the technical strengths of the MERN stack with a focus on user convenience, the Food Delivery App sets the stage for a modern, efficient, and enjoyable dining experience at the click of a button.

Scenario based Case study:

Scenario-Based Case Study:
Ravi is a busy software engineer who frequently relies on online food delivery for his meals. One evening, he craves Italian cuisine and opens the Food Delivery App. Using the app's intuitive interface, he browses a variety of nearby Italian restaurants, reads detailed menu descriptions, and selects a pasta dish. The app’s secure payment system allows him to complete the order seamlessly. As the order progresses, Ravi tracks the delivery in real-time through the app. Within 30 minutes, the meal arrives, and he leaves a review for the restaurant, appreciating the app’s efficiency and ease of use.

Problem Statement:

The Food Delivery App aims to address common challenges faced by users in the online food delivery ecosystem by providing a seamless and efficient platform. Built using the MERN stack, the app focuses on user convenience, diverse restaurant options, secure transactions, and a personalized experience. It strives to eliminate inefficiencies and enhance the overall food ordering and delivery process for modern users.

1. **Diverse Restaurant and Cuisine Options**

   Users can explore a wide variety of restaurants and cuisines, ensuring something for every taste and preference.

2. **Real-Time Order Tracking**

   Integrated tracking tools allow users to monitor the status and location of their orders in real-time, offering transparency and reliability.

3. **User-Friendly Interface**

   A clean, intuitive interface designed with React.js ensures easy navigation for browsing menus, placing orders, and managing accounts.

4. **Secure Payment Processing**

   Robust payment gateways safeguard user data and provide hassle-free payment options for a trustworthy transaction experience.

5. **Personalized Recommendations**
   The app leverages user data to suggest restaurants, dishes, and offers tailored to individual preferences, enhancing satisfaction and engagement.

Project Scope:

Target Audience:

- The food delivery Store caters to readers of all demographics who seek convenience and flexibility in purchasing and reading books. The platform is also ideal for:
  - Students requiring academic books on demand.
  - Casual readers interested in novels, self-help, or niche topics.
  - Professionals seeking reference materials or industry-specific literature.

Use Cases:

- A user can create an account to save their preferences and purchase history.
- The application allows users to browse food by Chinese,Indian etc
- Purchased foods are stored in a virtual library for easy access.
- Users can securely purchase food using integrated payment methods.

Project Setup and Configuration:

1. Install required tools and software:

a) Node.js
b) Mango DB
c) Create-react-App

2. Create project folders and files:

a. Client folders
b. Server folders

3. Install Packages:

a. Frontend npm Packages

i. Axios
ii. React-Router –dom
iii. Bootstrap
iv. React-Bootstrap

b. Backend npm Packages

i. Express
ii. Mongoose
iii. coros

Frontend Development:

- **Setup React Application:**

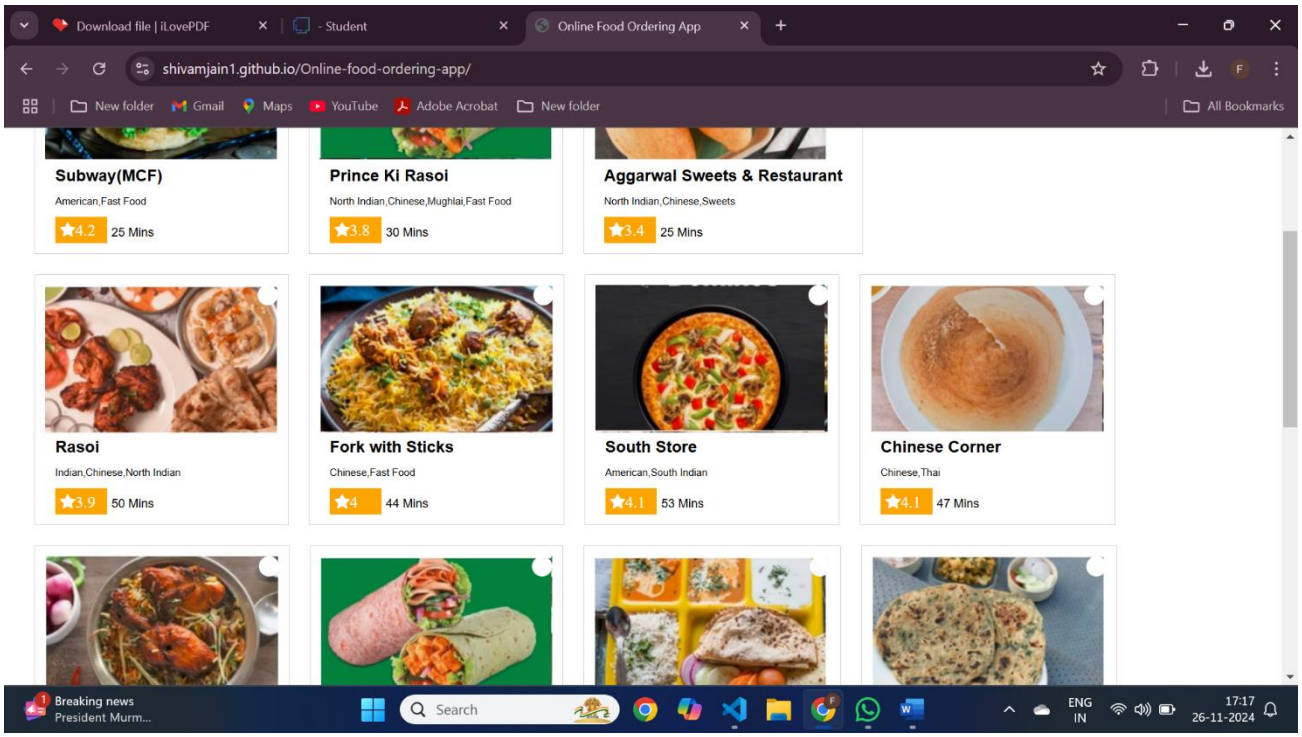
a. Create React application.
b. Configure Routing.
c. Install required libraries.

- **Design UI components:**

a. Create Components.
b. Implement layout and styling.
c. Add navigation.

- **Implement frontend logic:**

    a. Integration with API endpoints.
    b. Implement data binding.



Backend Development:

- **Setup express server**

    a. Create index.js file in the server (backend folder).
    b. Create a .env file and define port number to access it globally.
    c. Configure the server by adding cors, body-parser.

- **User Authentication:**

    a. Create routes and middleware for user registration, login, and logout.
    b. Setup authentication middleware to protect routes that require user authentication.

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

- **Implement Data Models:**

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

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

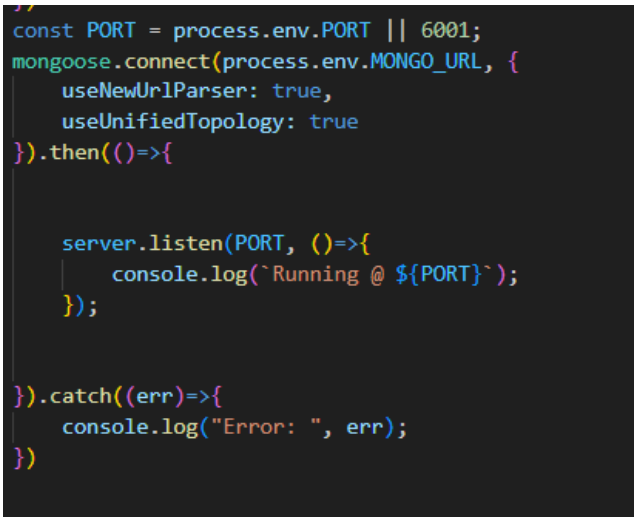
Database:

- **Configure MongoDB:**

    a. Install Mongoose.
    b. Create database connection.
    c. Create Schemas & Models.

- **Connect database to backend:**

    Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.

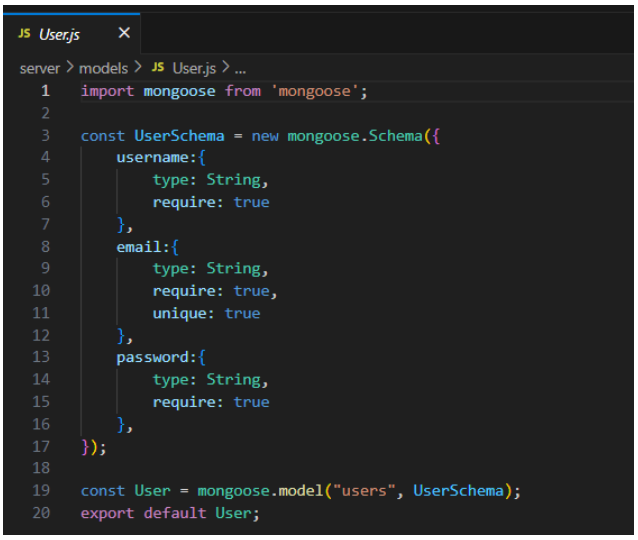


```
const PORT = process.env.PORT || 6001;
mongoose.connect(process.env.MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(()=>{


    server.listen(PORT, ()=>{
        console.log(`Running @ ${PORT}`);
    });


}).catch((err)=>{
    console.log("Error: ", err);
})
```

- **Configure Schema:**
    Firstly, configure the Schemas for MongoDB database, to store the data in such pattern. Use the data from the ER diagrams to create the schemas. The Schemas for this application look alike to the one provided below.



User.js

server > models > User.js > ...

```
import mongoose from 'mongoose';

const UserSchema = new mongoose.Schema({
    username:{
        type: String,
        require: true
    },
    email:{
        type: String,
        require: true,
        unique: true
    },
    password:{
        type: String,
        require: true
    },
});

const User = mongoose.model("users", UserSchema);
export default User;
```

Software Requirement:

Code Editor:

Visual Studio Code (VS Code): A lightweight and powerful code editor that supports JavaScript, React.js, Node.js, and other web technologies. It includes useful features like IntelliSense, debugging, and integrated terminal, making it ideal for MERN stack development.

Node.js:

A runtime environment for JavaScript that allows developers to run JavaScript code outside the browser. It is used to build the backend of the application with Express.js.

MongoDB:

A NoSQL database to store user data, book details, and purchase history. MongoDB's flexible document-based storage is perfect for handling the dynamic structure of e-books and user information.

npm (Node Package Manager):

The default package manager for Node.js, which helps manage project dependencies like React, Express, and other libraries used in the project.

Express.js:

A web application framework for Node.js used to create the API that handles routing, requests, and interactions between the frontend and database.

React.js:

A JavaScript library for building user interfaces, particularly single-page applications. React will be used to build the frontend of the E-book Store, ensuring a dynamic and responsive user experience.

Postman:

A powerful API testing tool used for testing the Express.js API endpoints. It helps in debugging and ensuring that the backend logic is working as expected.

Git and GitHub:

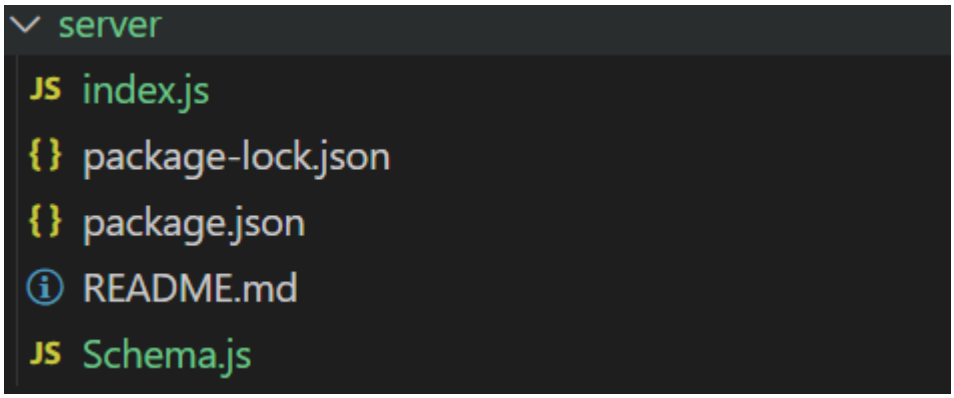
Git: A version control system used to manage and track changes to the project files.

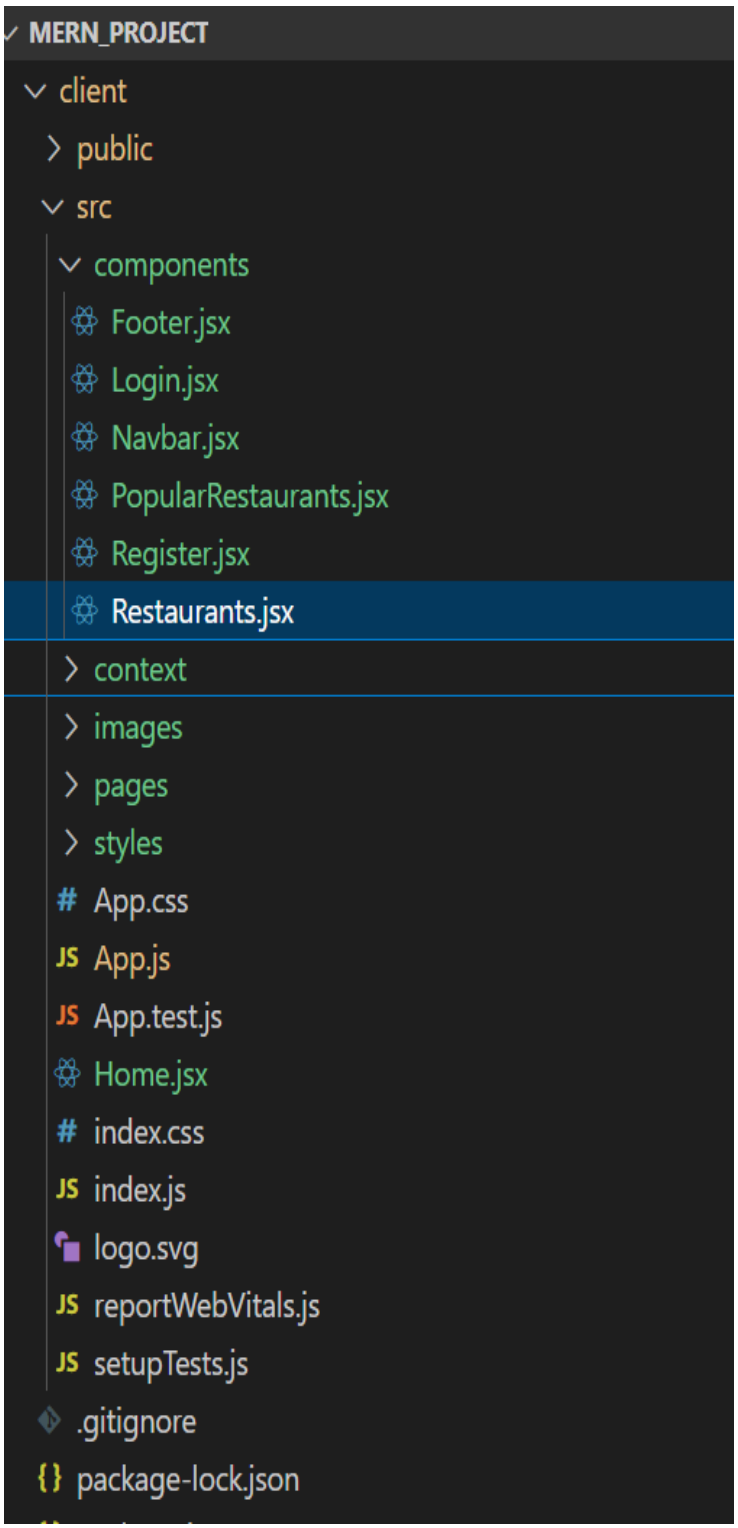
GitHub: A cloud-based platform to host the project's code, enabling collaboration and version control.

Google Chrome / Mozilla Firefox:

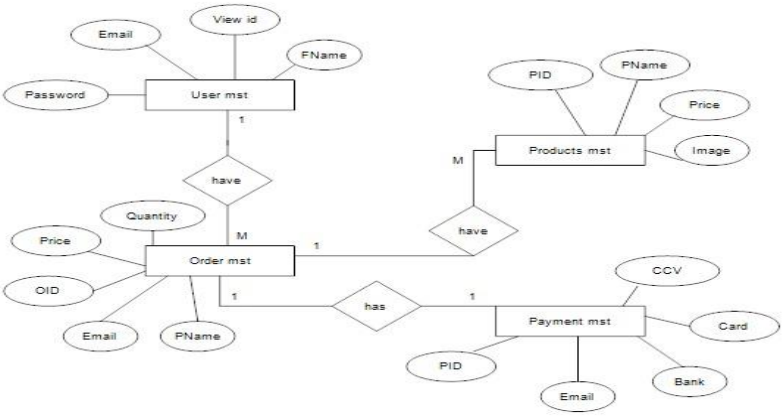
These are popular web browsers for testing and running the application in real-time to ensure the frontend's responsiveness and functionality.

Project Structure:




MERN_PROJECT
client
public
src
components
Footer.jsx
Login.jsx
Navbar.jsx
PopularRestaurants.jsx
Register.jsx
Restaurants.jsx
context
images
pages
styles
App.css
App.js
App.test.js
Home.jsx
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.gitignore
package-lock.json

E-R DIAGRAM:



.

Entities:

1. User_mst: Represents the users of the system, likely customers.
2. Products_mst: Represents the products available for purchase.
3. Order_mst: Represents the orders placed by users.
4. Payment_mst: Represents the payment information associated with orders.

Relationships:

1. User_mst to Order_mst (One-to-Many): One user can place many orders.
2. Order_mst to Products_mst (Many-to-Many): An order can contain multiple products, and a product can be included in multiple orders.
3. Order_mst to Payment_mst (One-to-One): Each order has exactly one associated payment.

Explanation:

- User_mst to Order_mst: A single user can initiate multiple orders. This is represented by the "have" diamond with a "1" on the User_mst side and "M" on the Order_mst side.
- Order_mst to Products_mst: A single order can include multiple products, and a single product can be included in multiple orders. This is a many-to-many relationship, which is typically represented by an intermediate entity or a junction table. In this case, the "have" diamond with "M" on both sides signifies this relationship.
- Order_mst to Payment_mst: Each order is associated with a single payment, and each payment is for a single order. This is a one-to-one relationship, represented by the "has" diamond with "1" on both side.
  and payments in the system.

Working of the System:

The food delivery Store application is designed to provide a seamless, user-friendly experience for browsing, purchasing, and managing restuarants. The system consists oftwo main components: Frontend (React.js) and Backend (Node.js, Express.js, MongoDB), with communication between them via RESTful APIs.

1. User Registration & Authentication:

- The user can sign up or log in to the system through a secure registration process. This involves submitting their email and password.
- Upon successful registration, the user's data (email, password) is saved in the MongoDB database. The password is hashed using Bcrypt.js for security.
- A JWT (JSON Web Token) is generated and sent back to the frontend, which is used to authenticate the user for subsequent requests.

2. Browsing Foods:

- Upon logging in, the user is directed to the **home page**, where a list of available restuarants is displayed.
- The frontend (built with React.js) fetches the book data from the backend using API calls. The book information (such as variety of food,restuarants) is stored in MongoDB.

3. Detail Page:

- When a user clicks on a food, the system shows detailed information on a separate page, including a description, food details, reviews, and price.
- The frontend makes an API call to fetch the detailed data for the specific food from the backend.

4. Shopping Cart:

- The user can add food to their **shopping cart**. The cart holds the e-books selected for purchase.
- The cart data is stored temporarily on the frontend (in the local state or Redux if implemented).
- Each time a user adds a food to the cart, the frontend updates the cart view.

5. Purchase & Checkout Process:

- Once the user is ready to purchase, they proceed to the checkout page.
- The system confirms the user's billing details (if implemented).

- A payment gateway can be integrated (e.g., UPI, PayPal) for handling the payment. However, for simplicity, this step can initially be simulated.

- Upon successful payment, an order is placed, and the order details are saved in the backend (MongoDB) under the user's profile.
- A success message is displayed, and the books will be delivered to the person.

6. Digital System:

Once the purchase is complete, the user can visit their orders, where theycan access and repurchased their order.

The library is stored in the backend as part of the user's profile.

The frontend fetches this data to show the purchased food order in the user's library.

7. Admin Panel (Optional Feature):

- An admin panel can be created for the site administrators to manage the collection of resturants (e.g., add, edit, delete food).
- Admins can also manage user data and monitor transactions, providing them with complete control over the food delivery app.

8. Technologies in Action:

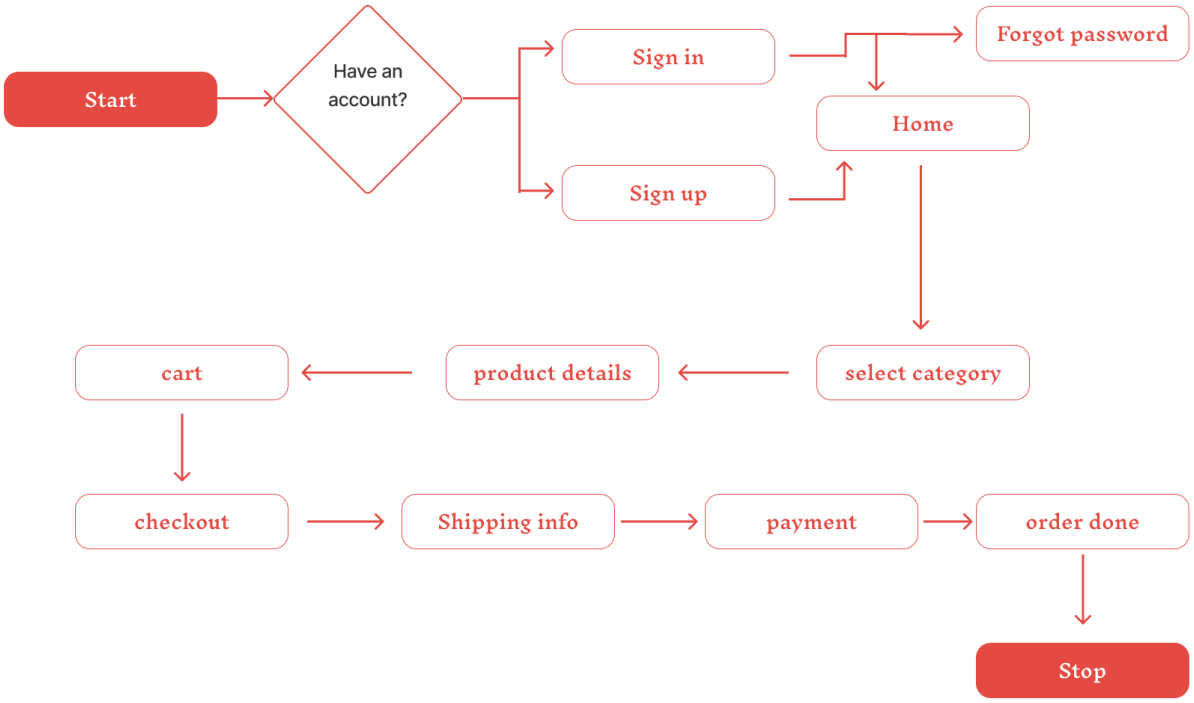
- React components are used to build dynamic and responsive user interfaces.
- React Router is used to manage navigation between different pages (home, food details, cart, library, etc.).
- State management with React Context or Redux is used for managing the cart, user data, and other application states.

9. Security Features:

- JSON Web Tokens are used to secure the routes, ensuring only authenticated users can make purchases, access their libraries, or modify their account settings.

- User passwords are hashed with **Bcrypt.js** to ensure they are stored securely in the database.

Application Flow:



Conclusion:

The SB food delivery app using MERN Stack project effectively demonstrates the applicationof modern web technologies to create a user-friendly, secure, and scalable online platform for purchasing and managing resturants. By utilizing the MERN stack— MongoDB, Express.js, React.js, and Node.js—the system ensures seamless integration between the front-end and back-end, providing an efficient and interactiveexperience for both users and administrators.

Seamless User Experience:

The food delivery provides an intuitive and responsive interface using React.js, allowing users to easily browse, select, and purchase foods. The system's user-friendly design enhances overall customer satisfaction.

Efficient and Scalable Data Management:

The use of MongoDB enables flexible and scalable management of ordering the food, ensuring the system can handle an increasing number of users and order efficiently.

Admin Control and Management:

The optional admin panel allows store administrators to manage orders, useractivities, and transactions, providing full control over the platform.

Cost-Effective and Future-Proof:

The digital nature of the store eliminates physical costs and offers immediate accessto foods. The system is scalable, allowing for future features like personalized recommendations, mobile app Innovative

Features for Future Enhancements:

AI-Driven Recommendations:

- Machine learning algorithms to suggest books based on user reading habits and preferences.

Subscription Plans:

- Introducing subscription-based models for unlimited access to a curated library of books.

Community Engagement:

- Features like discussion forums, book clubs, and live author interactions.

Integration with Audiobooks:

- Expanding the platform to include audiobook purchases and streaming.

Advanced Analytics:

- Insights for users on their reading patterns and popular trends.

integration, and subscription service.