1.INTRODUCTION

The **Food and Delivery App** is an all-in-one solution designed to enhance the convenience of food ordering and delivery for both customers and restaurants. This web-based platform allows users to explore a wide variety of menus, place orders effortlessly, and enjoy seamless delivery to their doorstep. The app is packed with features aimed at providing an exceptional user experience, including promo code discounts, multiple payment options like cash on delivery and net banking, and access to a selection of delicious food from local and international cuisines.

With the ability to browse menus and customize orders, users can easily discover new meals and enjoy personalized dining experiences. The integration of promo codes offers additional savings, while flexible payment methods ensure convenience for every type of customer. By leveraging modern technologies, the app delivers a fast, user-friendly, and reliable service, making it easier than ever for customers to satisfy their cravings while helping restaurants manage their orders efficiently.

1.1 Modules Description

1.1.1. User Module:

Home Page:

The Home Page serves as the central **platform** for users, showcasing the latest food offerings, featured restaurants, and exclusive deals. It provides easy navigation for exploring menus, discovering new dishes, and accessing ongoing promotions, creating a seamless experience for users to quickly find and order their favorite meals.

Menu:

The Menu section allows users to browse an extensive selection of dishes from various restaurants, featuring detailed descriptions and customization options. Customers can easily explore their favorites and discover new culinary delights to order at their convenience.

About:

The Food and Delivery App aims to connect food lovers with a diverse range of restaurants, making it easy to explore menus and enjoy delicious meals at home. Our mission is to provide a seamless ordering experience while delivering the latest food trends and exclusive deal.

Mobile App:

The Food and Delivery App is available on mobile devices, providing users with the convenience of ordering food on the go. With a user-friendly interface, customers can explore menus, place orders, and track deliveries anytime, anywhere.

Contact Us:

The Contact Us section provides users with various ways to reach our support team for inquiries, feedback, or assistance. Whether through email, phone, or a dedicated chat feature, we are here to ensure a smooth and enjoyable experience.

Sign up :
Functionality for new users to create accounts and join the platform.

Sign in:
Users can log in using their email and password .

1.1.2Admin Module:

Profile:
The Admin Profile provides a personalized space for administrators to manage their orders , update and remove products.

AddItems:
The Add Items feature allows administrators to easily include new food offerings to the menu. Admins can input detailed information such as item name, description, price, and images, ensuring that the menu is always up-to-date with the latest culinary options for customers to explore.

ListItems:

The List Items feature allows administrators to view and manage all food offerings on the menu. Admins can easily browse, edit details, or remove items to keep the menu accurate and current.

Orders:

The Orders feature enables administrators to track and manage customer orders in real-time. Admins can view order details, update statuses, and ensure timely processing and delivery, enhancing the overall customer experience

Description of Web Technologies

JAVASCRIPT:

JavaScript, often abbreviated as JS, is a programming language and core technology of the World Wide Web alongside HTML and CSS. As of 2024, 98.8% of websites use JavaScript on the client side for webpage behaviour, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on users' devices. JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard.[11] It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programminginterfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

REACT:

ReactJS, also known as React, is a popular JavaScript library for building user interfaces. It is also reffered to as a front-end Javascript library.It was developed by Facebook and is widely used for creating dynamic and interactive web applications.React is a declarative, component-based library that allows developers to build reusable UI components and it follows the Virtual DOM (Document Object Model) approach, which optimizes rendering performance by minimizing DOM updates. React is fast and works well with other tools and libraries. React operates by creating an in-memory virtual DOM rather than directly manipulating the browser's DOM. It performs necessary manipulations within this virtual representation before applying changes to the actual browser DOM.

NODE.JS:

EXPRESS.JS

Express.js is a minimal and flexible web application framework for Node.js. It simplifies building web servers and APIs by providing powerful tools for routing, handling HTTP requests and responses, and integrating middleware, making it one of the most popular choices for back-end development in the Node.js ecosystem.

1. System Environment

Operating System:

An operating system (OS) is system software that manages computer hardware and software resources, and provides common services for computer programs. This Project can develop anddeployed in macOS, Linux, Windows with some changes.

Web Server:

Since React is a JavaScript library, it's often hosted on servers that support Node. js, which is a JavaScript runtime that allows developers to build server-side applications in JavaScript. Hosting providers that support Node. you don't need a traditional web server like Apache or Nginx to serve your React application.

Programming languages and frameworks:

The following languages and frameworks are used: HTML, CSS used to design web page structure and presentation. JavaScript, React used for behaviour, and React use state to manage the components and data.

IDE (integrated development environment):

The IDE used in this project is Visual Studio Code. Visual Studio Code, also commonly referred to as VS Code, is a source-code editor developed by Microsoft for Windows, Linux, and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add functionality.

Version control System:

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. In this project we used git as Version control system. Git is one of version control system.

1.1 Hardware Configuration

The required hardware configuration for this project is as follows:

Processor: 12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz

SSD: 512 GB

RAM: 8 GB

System Type: 64-bit operating system, x64-based processor

1.2 Software Configuration

The required software configuration for this project and also used in this project are as follows:

Operating System: Windows 10,11

Front End : React.js

Back End : Node.js,Express.js

Database : MongoDB Atlas

1.3 Software Features

System specification:

About windows

Microsoft Windows, computer operating system (OS) developed by Microsoft Corporation to run personal computers (PCs). Featuring the first graphical user interface (GUI) for IBM-compatible PCs, the Windows OS soon dominated the PC market. Approximately 90 percent of PCs run someversion of Windows.

- **Windows 1.0 (1985)**: The first version of Microsoft Windows, introducing a graphical userinterface for IBM-compatible PCs.
- **Windows 2.0 (1987)**: Introduced overlapping windows and improved graphical capabilities.
- **Windows 3.0 (1990)**: Added features like Program Manager, File Manager, and improvedgraphics.
- **Windows 95 (1995)**: A major release with significant GUI updates, support for long filenames, Plug and Play hardware, and the Start menu.
- **Windows 98 (1998):** Improved USB support, Internet Explorer 4 integration, and enhancedhardware support.
- **Windows ME (Millennium Edition) (2000):** Focused on multimedia and home networking, but was considered unstable by many users.
- **Windows XP (2001):** A major release known for its stability, improved GUI, and support for multimedia and networking.
- **Windows Vista (2006):** Introduced Aero visual style, enhanced security features, and improved search functionality.
- **Windows 7 (2009):** Addressed criticisms of Vista, with improved performance, redesignedtaskbar, and enhanced touch support.
- **Windows 8 (2012):** Introduced the Metro UI with touch-centric features and apps, but received mixed reviews due to its departure from traditional desktop interface.
- **Windows 8.1 (2013):** An update to Windows 8, reintroduced the Start button and made otherimprovements.
- **Windows 10 (2015):** The latest major version as of my last update, focusing on a unified user experience across different devices, with features like Cortana, virtual desktops, and theWindows Store.
- **Windows 11 (2021):** Released as a successor to Windows 10, with a redesigned Start menu,improved window management, and other enhancements.

2. SYSTEM DESIGN

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating user requirements into a detailed blueprint that guides the implementation phase. The goal is to create a well-organized and efficient structure that meets the intended purpose while considering factors like scalability, maintainability, and performance.

2.1 Input Design

In an information system, input is the raw data that is processed to produce output. The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

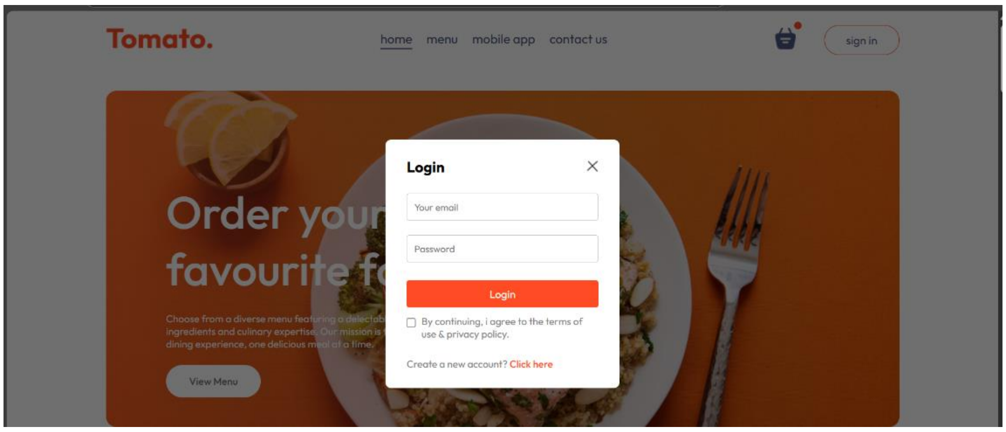
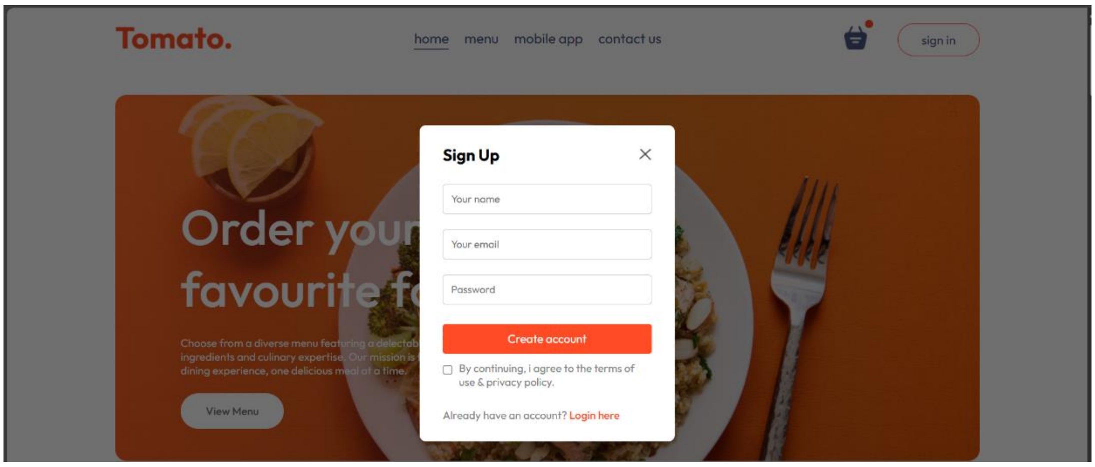


Figure 3.1.1 Sign in

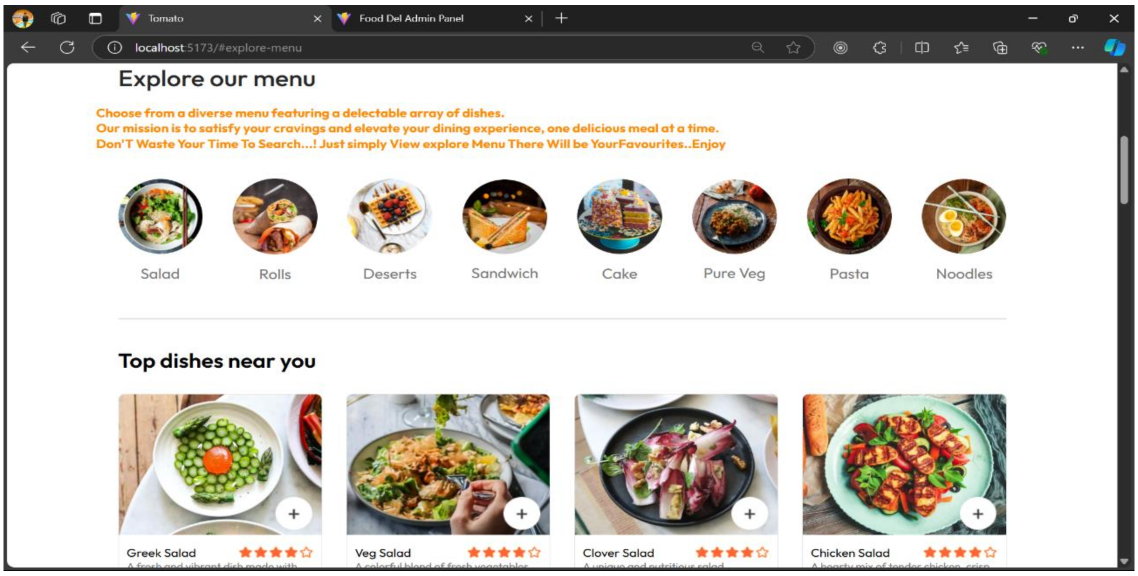
If the user already has an has an account, then they can sign in using email and password.



Choose from a diverse menu featuring a delectab ingredients and culinary expertise. Our mission is dining experience, one delicious meal at a time.

Figure 3.1.2 Sign up

If the user is visiting our website for the first time they need to sign up witha user name and email password and sign in.



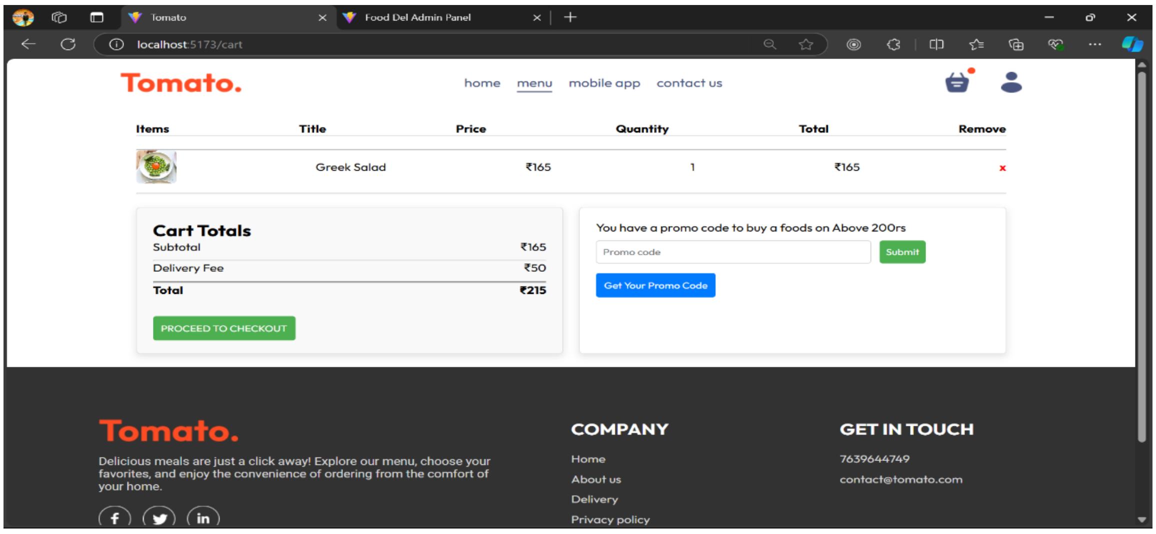
Explore our menu

Choose from a diverse menu featuring a delectable array of dishes.
Our mission is to satisfy your cravings and elevate your dining experience, one delicious meal at a time.
Don'T Waste Your Time To Search...! Just simply View explore Menu There Will be YourFavourites..Enjoy

Top dishes near you

Figure 3.1.3 Menu Page

User can use the menu option to explore the food



Items	Title	Price	Quantity	Total	Remove
	Greek Salad	₹165	1	₹165	x

Cart Totals

Subtotal	₹165
Delivery Fee	₹50
Total	₹215

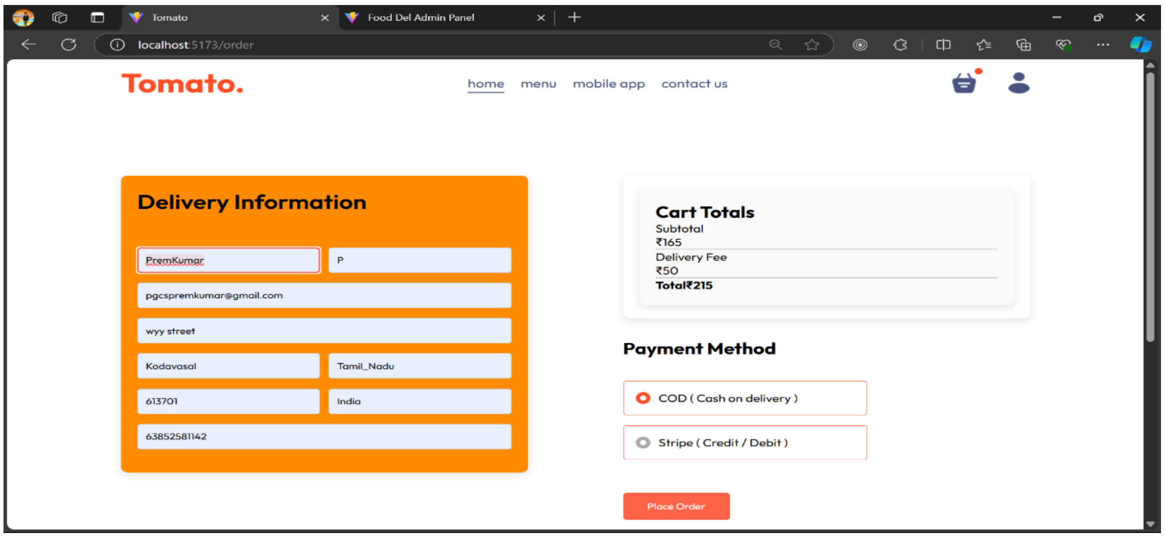
You have a promo code to buy a foods on Above 200rs

Delicious meals are just a click away! Explore our menu, choose your favorites, and enjoy the convenience of ordering from the comfort of your home.

7639644749
contact@tomato.com

Figure 3.1.4 Cart Page

If the user add the food ,it will added into cartpage.



Delivery Information

PremKumar	P
pgcspremkumar@gmail.com	
wyy street	
Kodavasal	Tamil_Nadu
613701	India
63852581142	

Cart Totals
Subtotal
₹165
Delivery Fee
₹50
Total₹215

Payment Method

[✓ COD (Cash on delivery)]
Stripe (Credit / Debit)

Figure 3.1.5 PlaceOrder Page

If the user can place order in the page.

2.2 Output Design

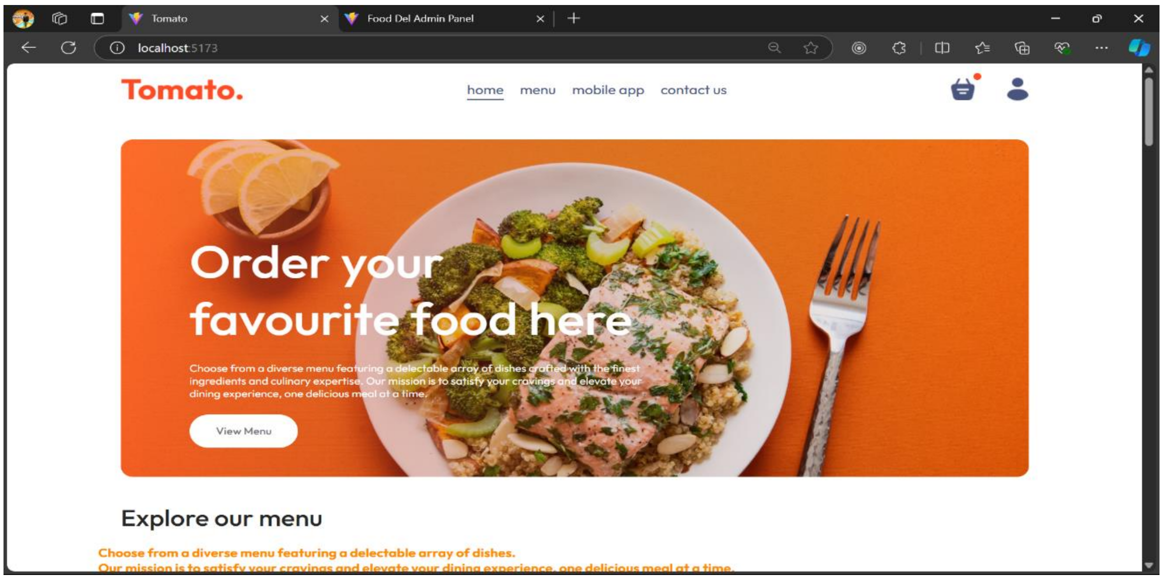
A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

2. Select methods for presenting information.

3. Create document, report, or other formats that contain information produced by the system. The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

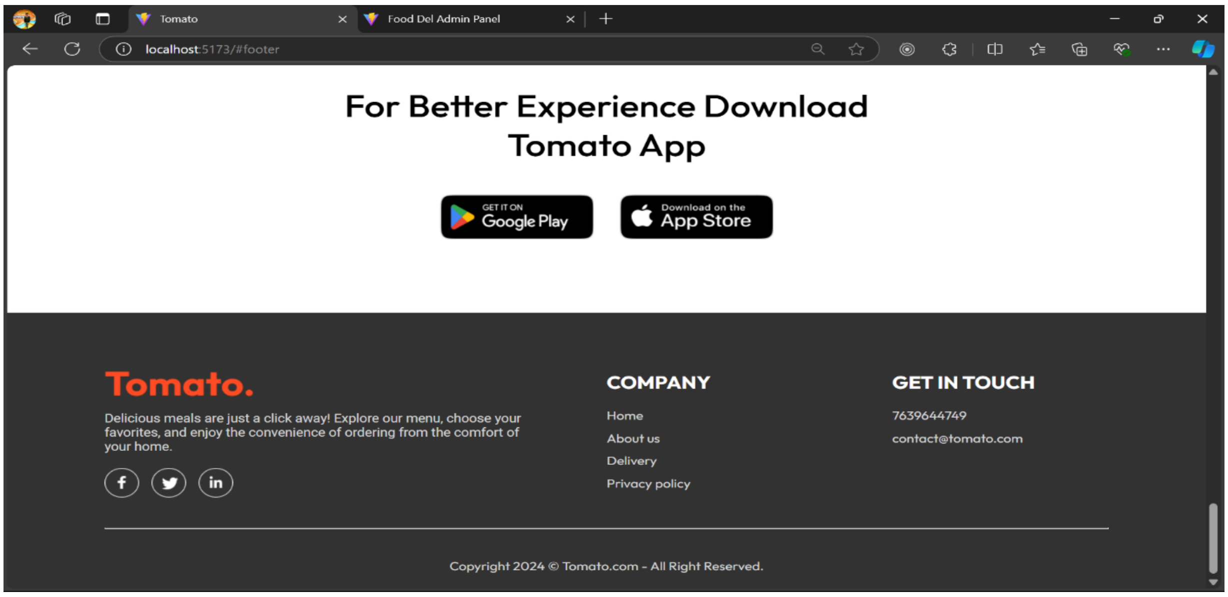


Choose from a diverse menu featuring a delectable array of dishes crafted with the finest ingredients and culinary expertise. Our mission is to satisfy your cravings and elevate your dining experience, one delicious meal at a time.

Explore our menu

Choose from a diverse menu featuring a delectable array of dishes.
Our mission is to satisfy your cravings and elevate your dining experience, one delicious meal at a time.

Figure 3.2.1 Home


For Better Experience Download Tomato App

GET IT ON Google Play | Download on the App Store

Tomato.

Delicious meals are just a click away! Explore our menu, choose your favorites, and enjoy the convenience of ordering from the comfort of your home.

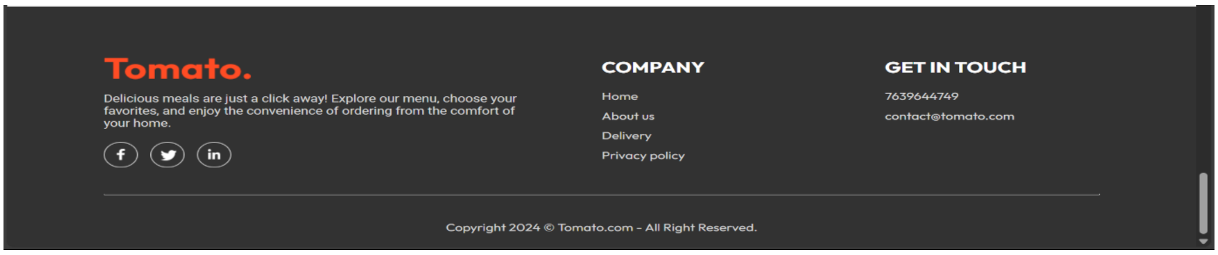
COMPANY
Home
About us
Delivery
Privacy policy

GET IN TOUCH
7639644749
contact@tomato.com

Copyright 2024 © Tomato.com - All Right Reserved.

Figure 3.2.2 Mobile-App & contact us page

In this page user can install mobileapp through the link and any issue the contact through email or phone number.


Tomato.

Delicious meals are just a click away! Explore our menu, choose your favorites, and enjoy the convenience of ordering from the comfort of your home.

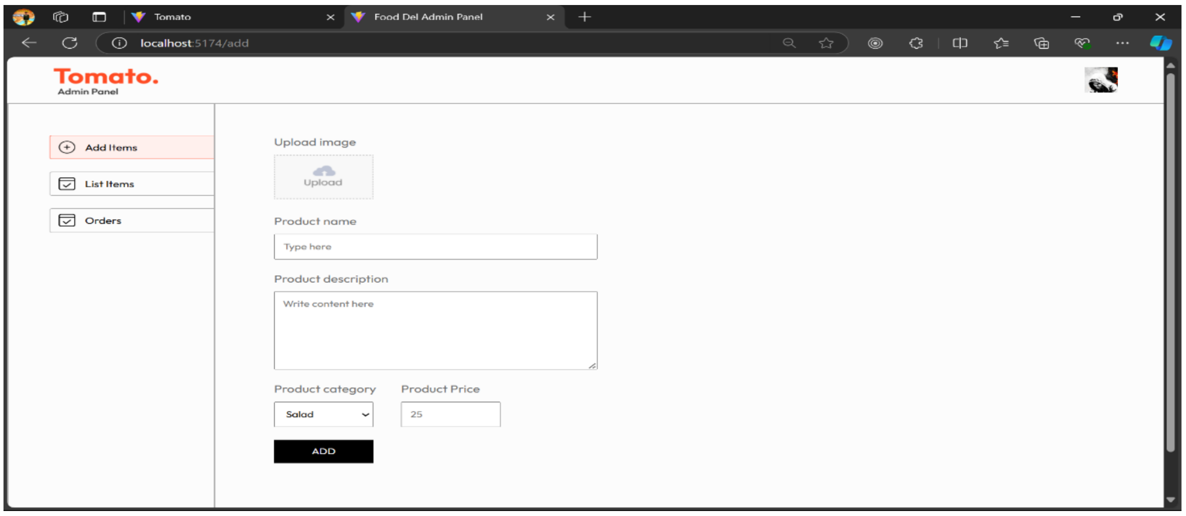
COMPANY
Home
About us
Delivery
Privacy policy

GET IN TOUCH
7639644749
contact@tomato.com

Copyright 2024 © Tomato.com - All Right Reserved.

Figure 3.2.3 Footer

In the footer we have our Facebook, Twitter,LinkedIn and email links for contact. And also, it has a privacy policy and terms and conditions of the App.


Tomato | Food Del Admin Panel

localhost:5174/add

Tomato.
Admin Panel

Add Items
List Items
Orders

Upload image
Upload

Product name
Type here

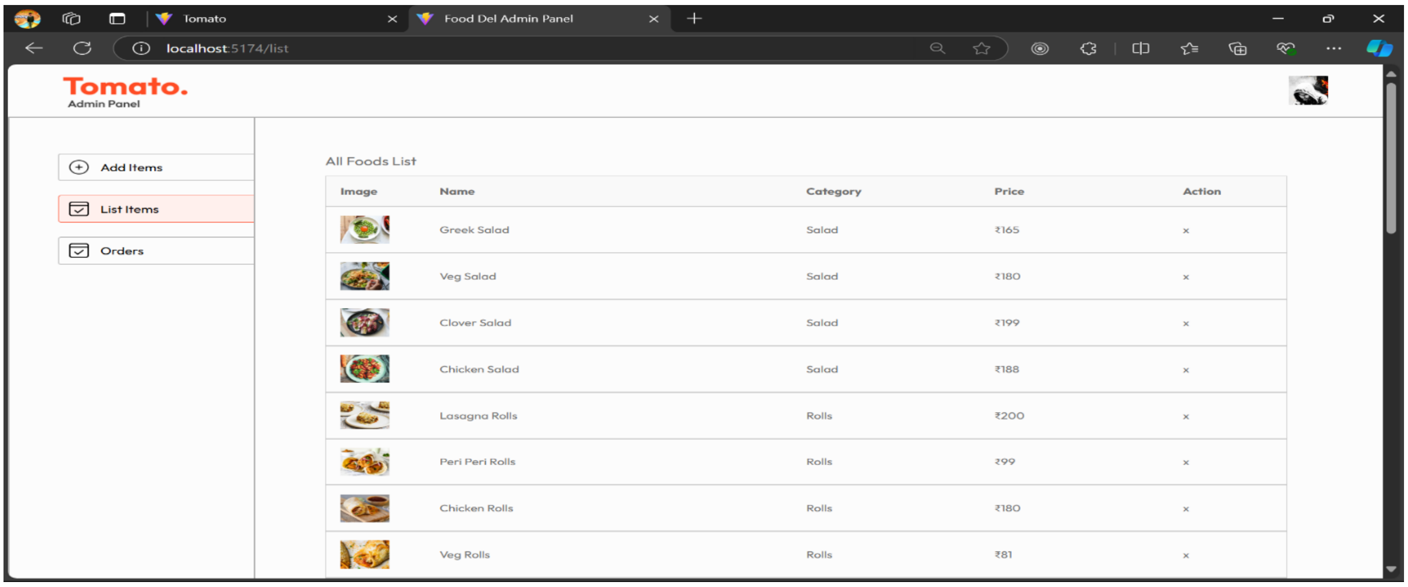
Product description
Write content here

Product category: Salad
Product Price: 25

ADD

Figure 3.2.4 Admin Panel &

Add Item Page


Tomato | Food Del Admin Panel

localhost:5174/list

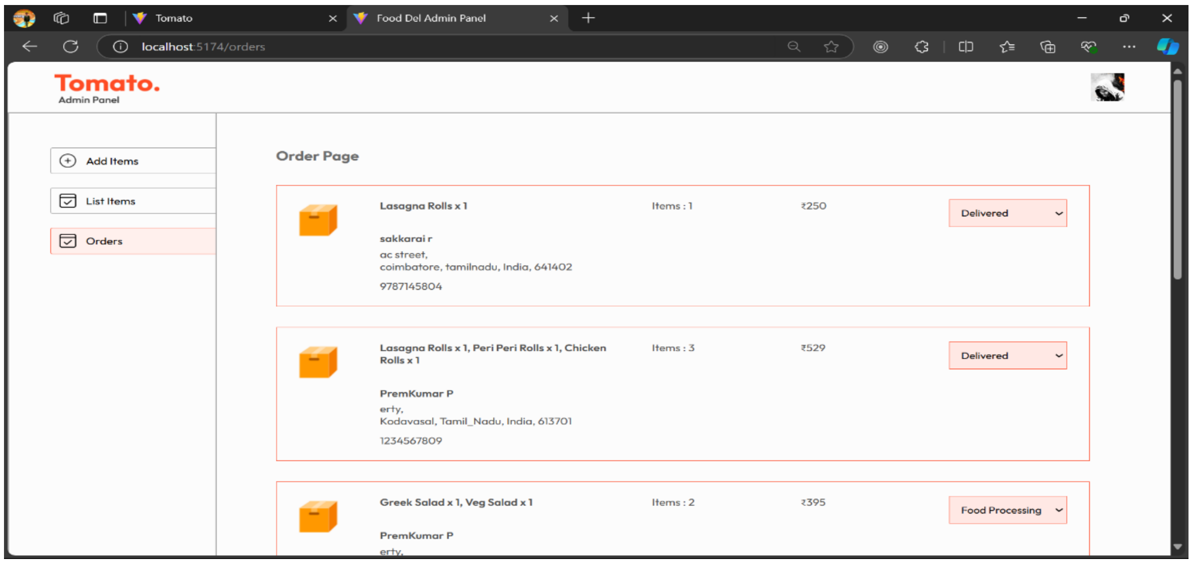
Tomato.
Admin Panel

Add Items
List Items
Orders

All Foods List

Image	Name	Category	Price	Action
	Greek Salad	Salad	₹165	x
	Veg Salad	Salad	₹180	x
	Clover Salad	Salad	₹199	x
	Chicken Salad	Salad	₹188	x
	Lasagna Rolls	Rolls	₹200	x
	Peri Peri Rolls	Rolls	₹99	x
	Chicken Rolls	Rolls	₹180	x
	Veg Rolls	Rolls	₹81	x

Figure 3.2.5 Admin List Items page


Tomato
Food Del Admin Panel
localhost:5174/orders
Tomato.
Admin Panel
Add Items
List Items
Orders
Order Page

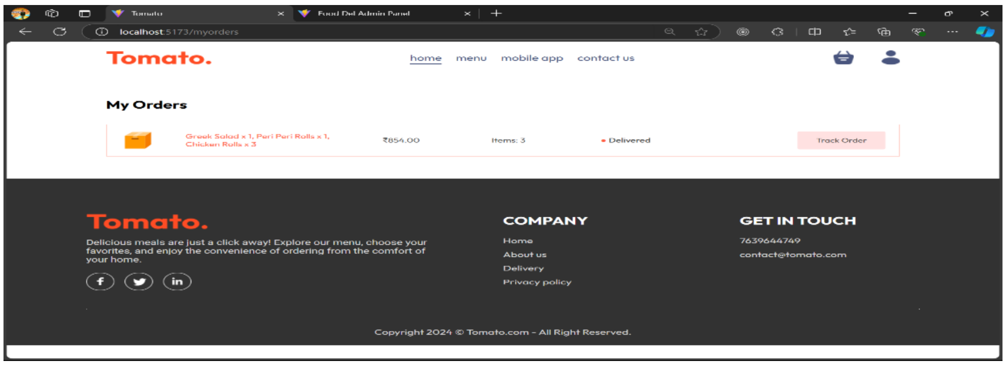
Lasagna Rolls x 1 | Items : 1 | ₹250 | Delivered
sakkarai r
ac street,
coimbatore, tamilnadu, India, 641402
9787145804

Lasagna Rolls x 1, Peri Peri Rolls x 1, Chicken Rolls x 1 | Items : 3 | ₹529 | Delivered
PremKumar P
erty,
Kodavasal, Tamil_Nadu, India, 613701
1234567809

Greek Salad x 1, Veg Salad x 1 | Items : 2 | ₹395 | Food Processing
PremKumar P
erty,

Figure 3.2.6 Admin Orders

Only admins have access the order status of user.



My Orders

Greek Salad x 1, Peri Peri Rolls x 1, Chicken Rolls x 3	₹854.00	Items: 3	● Delivered	Track Order

Tomato.

Delicious meals are just a click away! Explore our menu, choose your favorites, and enjoy the convenience of ordering from the comfort of your home.

COMPANY

Home
About us
Delivery
Privacy policy

GET IN TOUCH

7639644749
contact@tomato.com

Copyright 2024 © Tomato.com - All Right Reserved.

Figure 3.2.7 users

Now admin update the order status for user, refer Figure 3.2.6 Admin Orders .

3. Database Design

This database design is structured around three primary entities: User, Post, and Comment. Each entity is represented by a MongoDB collection with its own schema, and these collections relate to each other in specificways. Below is a detailed explanation of the design and the relationships between these entities.

3.3.1 User:

The User collection holds information about each person who uses the Food and Delivery App. This includes regular customers and administrators (admins). Here are the key attributes and their explanations:

Fields:

- username: A unique identifier for the user, ensuring that no two users can have the same username.
- email: Each user must provide a unique email address for login and communication purposes.
- password: A password is required to secure user accounts. Passwords should be hashed beforestoring in the database.
- Role: This field specifies whether the user is a regular customer or an administrator, allowing for role-based access control.
- **Timestamps**: createdAt and updatedAt fields, automatically managed by Mongoose, store whenthe user was created and last updated.

Relationships:

- **One-to-Many with Orders**: Each user can place multiple orders, but each order is tied to only one user.
- **One-to-Many with Menu Items:** If desired, you can implement a relationship where admins can manage multiple menu items, but each menu item is tied to only one admin.

3.3.2 Menu Item:

The Menu Item collection stores details about food offerings available in the app.

Fields:

- **name:** The name of the food item.
- **description:** A detailed description of the food item.
- **price:** The cost of the food item
- **image:** A link or path to an image of the food item.
- **Timestamps**: CreatedAt and updatedAt fields for tracking when the item was added or modified.

Relationships:

- **One-to-Many with Orders:** Each menu item can be part of multiple orders, while each order can consist of multiple menu items.

3.3.3 Order:

The Order collection holds information about each customer order made through the app.

Fields:

- **userId:** A reference to the User who placed the order.
- **menuItems:** An array containing the IDs of the Menu Items included in the order.
- **totalAmount:** The total cost of the order.
- **status:** The current status of the order (e.g., pending, in progress, completed).
- **timestamps:** CreatedAt and updatedAt fields for tracking when the order was placed and modified.

Relationships:

- **Many-to-One with Users**: Each order is associated with one user.
- **Many-to-One with Menu Items**: An order can include multiple menu items, and each menu item can be part of multiple orders.

Key Relationships:

User to Order (One-to-Many):

A single user can place multiple orders, but each order is associated with only one user. This is represented by the [`userId`] in the Order schema, which refers to a specific user.

Order To Menu_item (Many-to-Many):

Each order can contain multiple menu items, and each menu item can be part of multiple orders. This relationship is represented through an array of [`menuItemIds`] in the Order schema, linking to the specific Menu Items included in the order.

User to Menu_item (One-to-Many):

If implemented, each admin user can manage multiple menu items, but each menu item is tied to only one admin. This relationship can be represented by the [`admin`] in the Menu Item schema, linking it back to its creator.

Potential Enhancements:

- **Order Tracking**: Implement a tracking feature to allow users to see the status of their orders in real-time, enhancing transparency and customer satisfaction.
- **Promo Code**: Add a field in the Order schema for promo codes, enabling users to apply discounts at checkout and encouraging customer engagement.
- **Reviews And Ratings**: Introduce a review and rating system for menu items, allowing users to leave feedback. This can be represented by a separate Review schema linked to both the User and Menu Item.

This database design provides a solid foundation for your food and delivery app, supporting essential functionalities such as user registration, order management, and menu item exploration while offering ample opportunities for future enhancements and features.

3.4 Code Design

Code design involves structuring software code for efficiency, maintainability, and scalability by making architectural decisions and employing design patterns. It aims to create code that is modular, reusable, and easy to understand, ensuring readability and facilitating future modifications. Effective code design enhances software quality, promotes code reusability, and supports the evolution of software systems over time.

App.js

```
import React, { useState } from 'react';
import { Route, Routes } from 'react-router-dom';
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';



// Importing components and pages
import Home from './pages/Home/Home';
import Footer from './components/Footer/Footer';
import Navbar from './components/Navbar/Navbar';
import Cart from './pages/Cart/Cart';
import LoginPopup from './components/LoginPopup/LoginPopup';
import PlaceOrder from './pages/PlaceOrder/PlaceOrder';
import MyOrders from './pages/MyOrders/MyOrders';
import Verify from './pages/Verify/Verify';

const App = () => {
  const [showLogin, setShowLogin] = useState(false);

  return (
    <>
      <ToastContainer />
      {showLogin && <LoginPopup setShowLogin={setShowLogin} />}
      <div className='app'>
        <Navbar setShowLogin={setShowLogin} />
        <Routes>
          <Route path='/' element={<Home />} />
          <Route path='/cart' element={<Cart />} />
          <Route path='/order' element={<PlaceOrder />} />
          <Route path='/myorders' element={<MyOrders />} />
          <Route path='/verify' element={<Verify />} />
        </Routes>
      </div>
      <Footer />
    </>
  );
}

export default App;
```

Home.js

```
import React, { useState } from 'react'
import Header from '../../components/Header/Header'
import ExploreMenu from '../../components/ExploreMenu/ExploreMenu'
import FoodDisplay from '../../components/FoodDisplay/FoodDisplay'
import AppDownload from '../../components/AppDownload/AppDownload'

const Home = () => {

  const [category,setCategory] = useState("All")

  return (
    <>
      <Header/>
      <ExploreMenu setCategory={setCategory} category={category}/>
      <FoodDisplay category={category}/>
      <AppDownload/>
    </>
  )
}

export default Home
```

Admin Order.js

```
import React, { useEffect, useState } from 'react'
import './Orders.css'
import { toast } from 'react-toastify';
import axios from 'axios';
import { assets, url, currency } from '../../assets/assets';

const Order = () => {

  const [orders, setOrders] = useState([]);

  const fetchAllOrders = async () => {
    const response = await axios.get(`${url}/api/order/list`)
    if (response.data.success) {
      setOrders(response.data.data.reverse());
    }
    else {
      toast.error("Error")
    }
  }

  const statusHandler = async (event, orderId) => {
    console.log(event, orderId);
    const response = await axios.post(`${url}/api/order/status`, {
      orderId,
      status: event.target.value
    })
    if (response.data.success) {
      await fetchAllOrders();
    }
  }



  useEffect(() => {
    fetchAllOrders();
  }, [])

  return (
    <div className='order add'>
      <h3>Order Page</h3>
      <div className="order-list">
        {orders.map((order, index) => (
          <div key={index} className='order-item'>
            <img src={assets.parcel_icon} alt="" />
            <div>
              <p className='order-item-food'>
                {order.items.map((item, index) => {
                  if (index === order.items.length - 1) {
                    return item.name + " x " + item.quantity
                  }
                  else {
                    return item.name + " x " + item.quantity + ", "
                  }
                })}
              </p>
              <p className='order-item-name'>{order.address.firstName + " " + order.address.lastName}</p>
              <div className='order-item-address'>
                <p>{order.address.street + ","}</p>
                <p>{order.address.city + ", " + order.address.state + ", " + order.address.country + ", " +
order.address.zipcode}</p>
              </div>
              <p className='order-item-phone'>{order.address.phone}</p>
            </div>
            <p>Items : {order.items.length}</p>
            <p>{currency}{order.amount}</p>
            <select onChange={(e) => statusHandler(e, order._id)} value={order.status} name="" id="">
              <option value="Food Processing">Food Processing</option>
              <option value="Out for delivery">Out for delivery</option>
              <option value="Delivered">Delivered</option>
            </select>
          </div>
        ))}
      </div>
    </div>
  )
```

```
}

export default Order
```

Cart.js

```
import React, { useContext, useState, useEffect } from 'react';
import { StoreContext } from '../../Context/StoreContext';
import { useNavigate } from 'react-router-dom';

const Cart = () => {
  const { cartItems, food_list, removeFromCart, getTotalCartAmount, url, currency, deliveryCharge } =
useContext(StoreContext);
  const navigate = useNavigate();

  const [promoCode, setPromoCode] = useState('');
  const [appliedPromoCode, setAppliedPromoCode] = useState(null);
  const [discount, setDiscount] = useState(0);
  const [hovered, setHovered] = useState(false); // State to manage hover effect

  const totalCartAmount = getTotalCartAmount();
  const minimumAmountForPromo = 200; // Minimum amount to be eligible for promo code
  const discountPercentage = 25; // Discount percentage for the promo code

  // Function to generate a random promo code
  const generatePromoCode = () => {
    if (totalCartAmount > minimumAmountForPromo) {
      const code = `PROMO${Math.floor(1000 + Math.random() * 9000)}`; // Random 4-digit code
      setPromoCode(code);
      alert(`You are eligible for a promo code! Use code: ${code} for a 25% discount.`);
    } else {
      alert('Your cart total must be more than Rs. 200 to receive a promo code.');
    }
  };
```

```
// Function to apply promo code and calculate discount
const applyPromoCode = () => {
  if (promoCode && appliedPromoCode === null) {
    setDiscount((totalCartAmount * discountPercentage) / 100);
    setAppliedPromoCode(promoCode);
    alert('Promo code applied! You have received a 25% discount.');
  } else {
    alert('Invalid promo code or promo code already applied.');
  }
};

// Effect to clear promo code if cart total is below the minimum amount
useEffect(() => {
  if (totalCartAmount < minimumAmountForPromo) {
    setPromoCode(''); // Clear promo code
    setAppliedPromoCode(null); // Reset applied promo code
    setDiscount(0); // Reset discount
  }
}, [totalCartAmount, minimumAmountForPromo]);

return (
  <div className='cart' style={styles.cart}>
    <div className="cart-items" style={styles.cartItems}>
      <div className="cart-items-title" style={styles.cartItemsTitle}>
        <p>Items</p> <p>Title</p> <p>Price</p> <p>Quantity</p> <p>Total</p> <p>Remove</p>
      </div>
      <br />
      <hr />
      {food_list.map((item, index) => {
        if (cartItems[item._id] > 0) {
          return (
            <div key={index}>
              <div className="cart-items-title cart-items-item" style={styles.cartItemsItem}>
                <img src={url + "/images/" + item.image} alt="" style={styles.cartItemImage} />
                <p>{item.name}</p>
                <p>{currency}{item.price}</p>
                <div>{cartItems[item._id]}</div>
                <p>{currency}{item.price * cartItems[item._id]}</p>
                <p style={styles.cartItemsRemoveIcon} onClick={() => removeFromCart(item._id)}>x</p>
              </div>
              <hr />
            </div>
          );
        }
      })}
    </div>

    <div className="cart-bottom" style={styles.cartBottom}>
      <div
        className="cart-total"
        style={{
          ...styles.cartTotal,
          backgroundColor: hovered ? '#FF8C00' : '#f9f9f9' // Change background based on hovered state
        }}
        onMouseEnter={() => setHovered(true)}
        onMouseLeave={() => setHovered(false)} // Toggle hover state
      >
        <h2>Cart Totals</h2>
        <div>
          <div className="cart-total-details" style={styles.cartTotalDetails}>
            <p>Subtotal</p>
            <p>{currency}{totalCartAmount}</p>
          </div>
          <hr />
          <div className="cart-total-details" style={styles.cartTotalDetails}>
            <p>Delivery Fee</p>
            <p>{currency}{totalCartAmount === 0 ? 0 : deliveryCharge}</p>
          </div>
          <hr />
          {discount > 0 && (
            <div className="cart-total-details" style={styles.cartTotalDetails}>
              <p>Discount (25%)</p>
              <p>-{currency}{discount}</p>
            </div>
          )}
          <hr />
          <div className="cart-total-details" style={styles.cartTotalDetails}>
            <b>Total</b>
            <b>{currency}{totalCartAmount === 0 ? 0 : totalCartAmount + deliveryCharge - discount}</b>
          </div>
```

```
          </div>
          <button onClick={() => navigate('/order')} style={styles.proceedButton}>PROCEED TO CHECKOUT</button>
        </div>

        <div className="cart-promocode"
          style={{
            ...styles.cartTotal, // Apply the same styles as the cart total
            marginLeft: '20px' // Add margin for spacing
          }}
        >
          <div>
            <p>
            You have a promo code to buy  a foods on Above 200rs</p>
            <div className='cart-promocode-input' style={styles.cartPromocodeInput}>
              <input
                type="text"
                value={promoCode}
                onChange={(e) => setPromoCode(e.target.value)}
                placeholder='Promo code'
                style={styles.input}
              />
              <button onClick={applyPromoCode} style={styles.submitButton}>Submit</button>
            </div>
            <button onClick={generatePromoCode} style={styles.generatePromoButton}>
              Get Your Promo Code
            </button>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Cart;

const styles = {
  cart: {
    padding: '20px',
    animation: 'fadeIn 0.5s', // Add fadeIn animation
  },
cartItems: {
  marginBottom: '20px',
},
cartItemsTitle: {
  display: 'flex',
  justifyContent: 'space-between',
  fontWeight: 'bold',
  borderBottom: '1px solid #ddd',
},
cartItemsItem: {
  display: 'flex',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: '15px',
  transition: 'transform 0.3s',
},
cartItemImage: {
  width: '50px',
  height: '50px',
  objectFit: 'cover',
  borderRadius: '5px',
},
cartItemsRemoveIcon: {
  cursor: 'pointer',
  color: 'red',
  fontWeight: 'bold',
},
cartBottom: {
  display: 'flex',
  justifyContent: 'space-between',
},
cartTotal: {
  width: '50%',
  padding: '20px',
  border: '1px solid #ddd',
  borderRadius: '5px',
  boxShadow: '0 4px 10px rgba(0,0,0,0.1)',
  transition: 'transform 0.3s, background-color 0.3s', // Added transition for background color
},
cartTotalDetails: {
  display: 'flex',
  justifyContent: 'space-between',
  marginBottom: '10px',
},
proceedButton: {
  padding: '10px',
  backgroundColor: '#4CAF50',
  color: 'white',
  border: 'none',
  cursor: 'pointer',
  borderRadius: '4px',
  marginTop: '20px',
  transition: 'background-color 0.3s',
},
cartPromocodeInput: {
  display: 'flex',
  gap: '10px',
  marginTop: '10px',
},
input: {
  padding: '8px',
  width: '70%',
  border: '1px solid #ccc',
  borderRadius: '4px',
},
submitButton: {
  padding: '8px',
  backgroundColor: '#4CAF50',
  color: 'white',
  border: 'none',
  cursor: 'pointer',
  borderRadius: '4px',
  transition: 'background-color 0.3s',
},
generatePromoButton: {
  marginTop: '15px',
  backgroundColor: '#007BFF',
  color: 'white',
  padding: '10px',
  border: 'none',
    borderRadius: '4px',
    cursor: 'pointer',
    transition: 'background-color 0.3s',
  },
};

// Add keyframes for animations
const fadeIn = `
@keyframes fadeIn {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}`;

// Inject the styles into the document
const styleSheet = document.createElement("style");
styleSheet.type = "text/css";
styleSheet.innerText = fadeIn;
document.head.appendChild(styleSheet);
```

PlaceOrder.js

```
import React, { useContext, useEffect, useState } from 'react';
import './PlaceOrder.css';
import { StoreContext } from '../../Context/StoreContext';
import { assets } from '../../assets/assets';
import { useNavigate } from 'react-router-dom';
import { toast } from 'react-toastify';
import axios from 'axios';

const PlaceOrder = () => {
    const [payment, setPayment] = useState("cod");
    const [data, setData] = useState({
        firstName: "",
        lastName: "",
        email: "",
        street: "",
        city: "",
        state: "",
        zipcode: "",
        country: "",
        phone: ""
    });
    const [isTouching, setIsTouching] = useState(false); // State to manage touch effect

    const { getTotalCartAmount, token, food_list, cartItems, url, setCartItems, currency, deliveryCharge } =
useContext(StoreContext);
    const navigate = useNavigate();

    const onChangeHandler = (event) => {
        const name = event.target.name;
        const value = event.target.value;
        setData(data => ({ ...data, [name]: value }));
    };

    const validateForm = () => {
        // Basic validations
        if (!data.firstName || !data.lastName || !data.email || !data.street ||
            !data.city || !data.state || !data.zipcode || !data.country || !data.phone) {
            toast.error("All fields are required.");
            return false;
        }

        // Email validation
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(data.email)) {
            toast.error("Invalid email format.");
            return false;
        }

        // Phone number validation (must be exactly 10 digits)
        const phoneRegex = /^[0-9]{10}$/; // Updated to require exactly 10 digits
        if (!phoneRegex.test(data.phone)) {
            toast.error("Phone number must be exactly 10 digits long.");
            return false;
        }
        // Zip code validation (example: 5 digits)
        const zipRegex = /^[0-9]{6}$/; // Adjust according to your requirements
        if (!zipRegex.test(data.zipcode)) {
            toast.error("Zip code must be 6 digits.");
            return false;
        }

        return true;
    };

    const placeOrder = async (e) => {
        e.preventDefault();
        if (!validateForm()) return; // Stop if validation fails

        let orderItems = [];
        food_list.map((item) => {
            if (cartItems[item._id] > 0) {
                let itemInfo = item;
                itemInfo["quantity"] = cartItems[item._id];
                orderItems.push(itemInfo);
            }
        });

        let orderData = {
            address: data,
            items: orderItems,
            amount: getTotalCartAmount() + deliveryCharge,
        };

        if (payment === "stripe") {
            let response = await axios.post(url + "/api/order/place", orderData, { headers: { token } });
            if (response.data.success) {
                const { session_url } = response.data;
                window.location.replace(session_url);
            } else {
                toast.error("Something Went Wrong");
            }
        } else {
            let response = await axios.post(url + "/api/order/placecod", orderData, { headers: { token } });
            if (response.data.success) {
                navigate("/myorders");
                toast.success(response.data.message);
                setCartItems({});
            } else {
                toast.error("Something Went Wrong");
            }
        }
    };

    useEffect(() => {
        if (!token) {
            toast.error("To place an order, sign in first");
            navigate('/cart');
        } else if (getTotalCartAmount() === 0) {
            navigate('/cart');
        }
    }, [token, getTotalCartAmount, navigate]);

    // Inline styles for the delivery info box
    const deliveryInfoStyle = {
        position: 'relative',
        padding: '20px',
        borderRadius: '8px',
        backgroundColor: '#fff',
        boxShadow: '0 4px 10px rgba(0, 0, 0, 0.1)',
        transition: 'background-color 0.3s ease',
    };

    const deliveryInfoHoverStyle = {
        backgroundColor: 'darkorange',
    };

    // Styles for cart total details with touch effect
    const cartTotalDetailsStyle = {
        color: isTouching ? 'darkorange' : 'black',
        transition: 'color 0.3s ease',
        cursor: 'pointer', // Add cursor pointer for touch effect
    };

    return (
        <form onSubmit={placeOrder} className='place-order'>
          <div className="place-order-left">
            <div
              className="delivery-info"
              style={deliveryInfoStyle}
              onMouseEnter={(e) => e.currentTarget.style.backgroundColor = deliveryInfoHoverStyle.backgroundColor}
              onMouseLeave={(e) => e.currentTarget.style.backgroundColor = '#fff'}
            >
              <p className='title'>Delivery Information</p>
              <div className="multi-field">
                <input type="text" name='firstName' onChange={onChangeHandler} value={data.firstName}
    placeholder='First name' required />
                <input type="text" name='lastName' onChange={onChangeHandler} value={data.lastName}
    placeholder='Last name' required />
              </div>
              <input type="email" name='email' onChange={onChangeHandler} value={data.email} placeholder='Email
    address' required />
              <input type="text" name='street' onChange={onChangeHandler} value={data.street} placeholder='Street'
    required />
              <div className="multi-field">
                <input type="text" name='city' onChange={onChangeHandler} value={data.city} placeholder='City'
    required />
                <input type="text" name='state' onChange={onChangeHandler} value={data.state} placeholder='State'
    required />
              </div>
              <div className="multi-field">
                <input type="text" name='zipcode' onChange={onChangeHandler} value={data.zipcode} placeholder='Zip
    code' required />
                <input type="text" name='country' onChange={onChangeHandler} value={data.country}
    placeholder='Country' required />
              </div>
              <input type="text" name='phone' onChange={onChangeHandler} value={data.phone} placeholder='Phone'
    required />
            </div>
          </div>
          <div className="place-order-right">
          <div
              className="cart-total"
              style={deliveryInfoStyle}
              onMouseEnter={(e) => e.currentTarget.style.backgroundColor = deliveryInfoHoverStyle.backgroundColor}
              onMouseLeave={(e) => e.currentTarget.style.backgroundColor = '#fff'}
            >

            <div className="cart-total" style={{
              backgroundColor: '#f9f9f9',
              borderRadius: '8px',
              padding: '20px',
              boxShadow: '0 4px 10px rgba(0, 0, 0, 0.1)',
              transition: 'transform 0.3s ease, box-shadow 0.3s ease'
            }} onMouseEnter={(e) => {
              e.currentTarget.style.transform = 'scale(1.05)';
              e.currentTarget.style.boxShadow = '0 8px 20px rgba(0, 0, 0, 0.2)';
            }} onMouseLeave={(e) => {
              e.currentTarget.style.transform = 'scale(1)';
              e.currentTarget.style.boxShadow = '0 4px 10px rgba(0, 0, 0, 0.1)';
            }}>

              <h2>Cart Totals</h2>
              <div>
                <div className="cart-total-details" style={cartTotalDetailsStyle}>
                  <p>Subtotal</p><p>{currency}{getTotalCartAmount()}</p>
                </div>
                <hr />
                <div className="cart-total-details"
                   style={cartTotalDetailsStyle}
                   onTouchStart={() => setIsTouching(true)} // Set isTouching to true on touch
                   onTouchEnd={() => setIsTouching(false)} // Reset isTouching to false on touch end
                   onMouseEnter={() => setIsTouching(true)} // Optional: Keep it for mouse hover effect
                   onMouseLeave={() => setIsTouching(false)} // Optional: Reset for mouse leave
                >

                  <p>Delivery Fee</p>
                  <p>{currency}{getTotalCartAmount() === 0 ? 0 : deliveryCharge}</p>
                </div>
                <hr />
                <div className="cart-total-details" style={cartTotalDetailsStyle}>
                  <b>Total</b><b>{currency}{getTotalCartAmount() === 0 ? 0 : getTotalCartAmount() +
    deliveryCharge}</b>
                </div>
              </div>
```

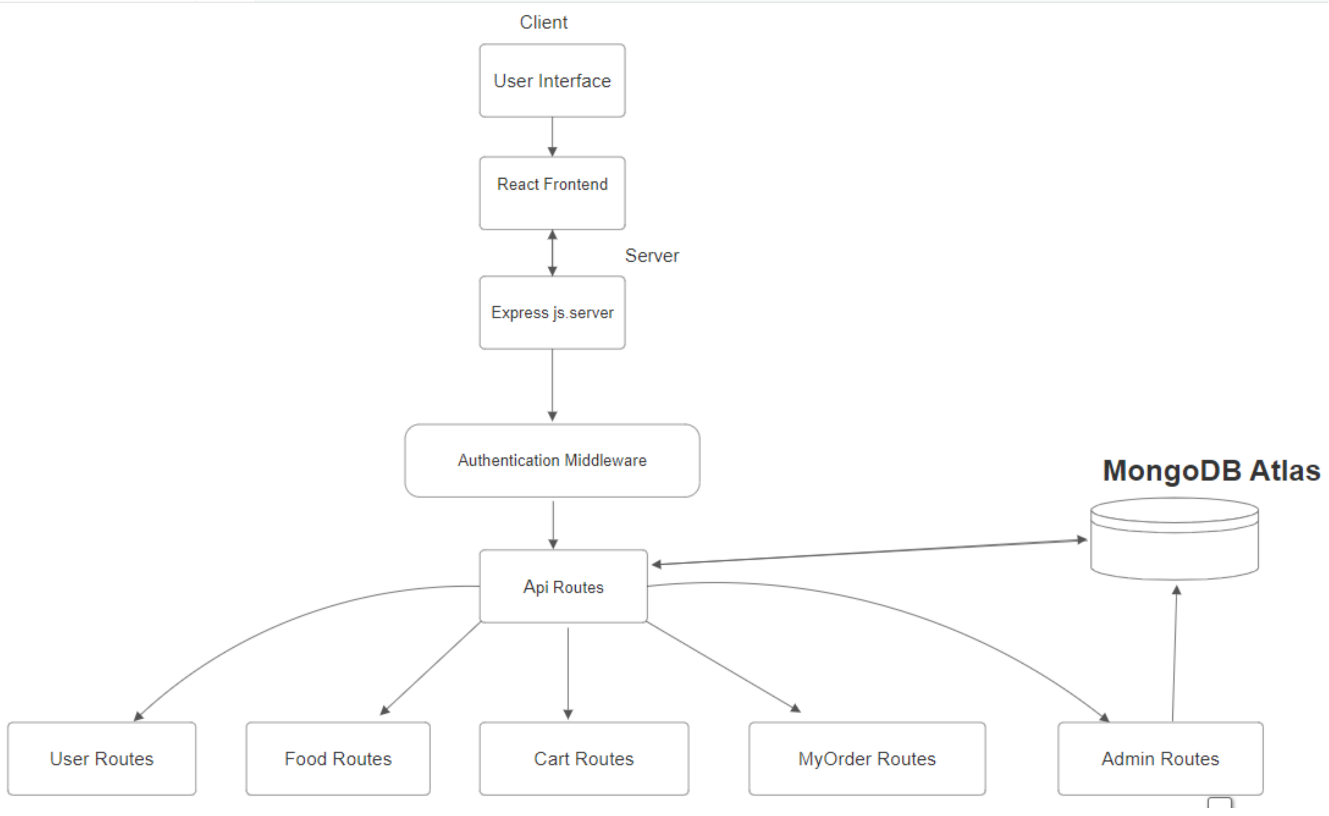
```
                </div>
                </div>
                <div className="payment">
                    <h2>Payment Method</h2>
                    <div onClick={() => setPayment("cod")} className="payment-option">
                        <img src={payment === "cod" ? assets.checked : assets.un_checked} alt="" />
                        <p>COD ( Cash on delivery )</p>
                    </div>
                    <div onClick={() => setPayment("stripe")} className="payment-option">
                        <img src={payment === "stripe" ? assets.checked : assets.un_checked} alt="" />
                        <p>Stripe ( Credit / Debit )</p>
                    </div>
                </div>
                <button className='place-order-submit' type='submit'>{payment === "cod" ? "Place Order" : "Proceed To
Payment"}</button>
            </div>

        </form>
    );
}

export default PlaceOrder;
```

4.FLOW DIAGRAM

A flow diagram, also known as a flowchart, is a visual representation that illustrates the sequence of steps in a process or system. It uses standardized symbols, such as ovals for starting and ending points, rectangles for processes or tasks, diamonds for decision points, and arrows to indicate the flow of control or information. Flow diagrams are valuable tools for clarifying complex processes, making them easier to understand and analyze. They are widely used in various fields, including business, engineering, and education, to improve communication, enhance problem-solving, and streamline workflows. By providing a clear overview of how tasks are interconnected, flow diagrams help identify inefficiencies and opportunities for improvement.

4.1 SYSTEM FLOW DIAGRAM

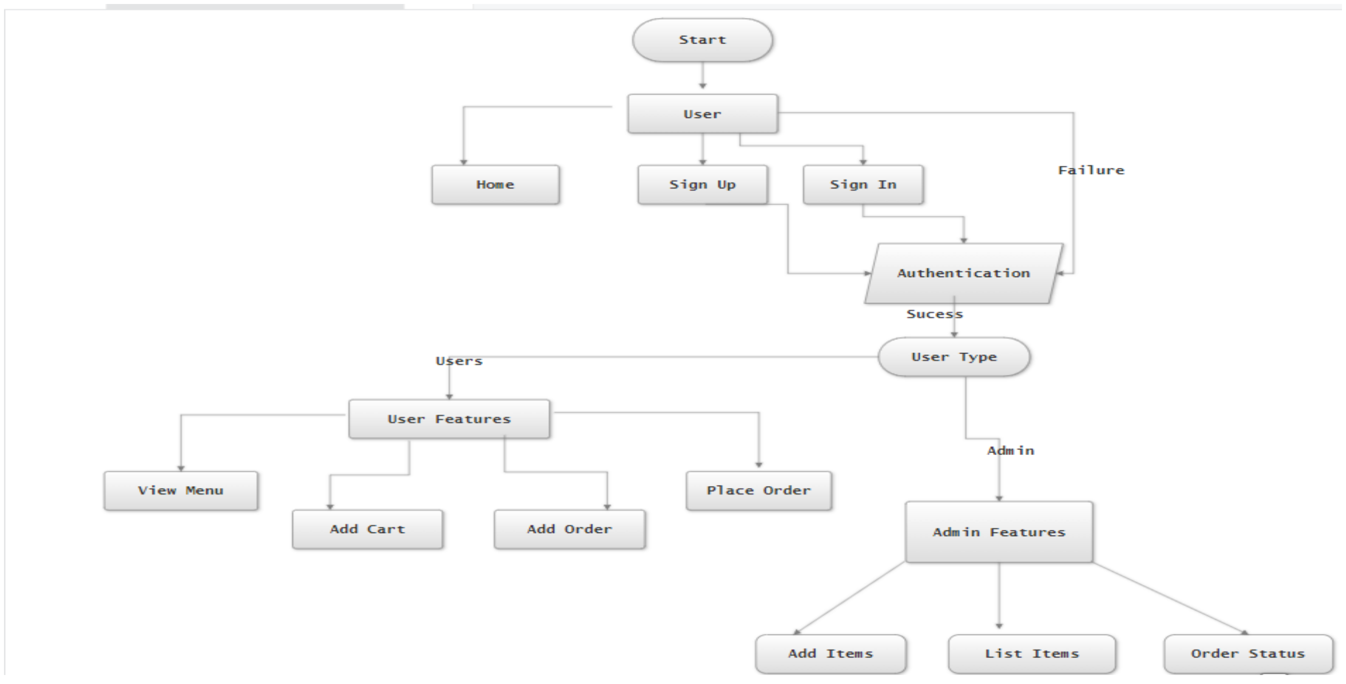
A system flow diagram is a specialized type of flowchart that visually represents the interactions and relationships between various components of a system. It focuses on the flow of information, data, or materials through the system, highlighting how inputs are transformed into outputs. Typically composed of symbols such as circles for processes, arrows for directional flow, and rectangles for external entities, system flow diagrams help in understanding the dynamics of complex systems. They are particularly useful in fields like software development, engineering, and process management, as they provide a clear and concise view of system architecture and operations. By illustrating the sequence and dependencies of processes, system flow diagrams aid in identifying potential bottlenecks, inefficiencies, and areas for improvement, facilitating better decision-making and system design.



System Flow Diagram

4.2 PROCESS FLOW DIAGRAM

A process flow diagram (PFD) is a graphical representation that details the steps and sequence of activities involved in a specific process. Typically used in engineering and manufacturing contexts, PFDs illustrate the flow of materials and information between various stages, depicting key elements such as inputs, outputs, equipment, and control points. Utilizing standardized symbols, PFDs help to simplify complex processes by breaking them down into easily understandable components. This clarity aids in process analysis, optimization, and troubleshooting, making it an essential tool for engineers, project managers, and quality assurance professionals. By providing a visual overview, process flow diagrams enhance communication among stakeholders, facilitate training, and support compliance with industry standards, ultimately contributing to more efficient and effective process management.



Process Flow Diagram

5. SYSTEM TESTING

System testing ensures that the entire system functions as expected and meets the specified requirements. It involves validating the system's behavior against predefined criteria to ensure its correctness, completeness, and reliability. The system testing of this Food Website is broken down into three types: validation testing, unit testing, and integration testing.

5.1 Validation testing

Validation testing ensures that the system meets the user's requirements and expectations. It verifies that the system fulfills its intended purpose and performs all specified functions accurately.

Key Validation Tests:

Account Creation: Ensure that the account creation form accurately verifies user-provided information such as username, email, and password, adhering to validation rules.
Name field validation: Accepts only alphabetic characters. Displays an error message if numbers or special characters are entered.

Example:

Error: "Name cannot contain numbers or symbols."

Email field validation: Validates proper email format and ensures the email is not already registered.

Example:

Error: "Please enter a valid email."

Password field validation: Enforces strong password requirements (uppercase, lowercase, digits, symbols) and displays error messages for weak passwords.

Login Functionality: Test the login functionality to ensure proper user authentication.**Invalid credentials**: Display appropriate error messages.

Example:

Error: "Incorrect email or password."

5.2 Unit testing

Unit testing focuses on testing individual components or modules in isolation to verify that each unit works as expected.

Key Unit Tests:

React Components: Test individual React components to ensure they render correctly and respond properlyto user interactions.

Form Components: Validate that forms (login, sign-up, Delivery information) are working correctly.

- Input fields for username, email, password validation.
- Button clicks triggering the right actions (e.g., submission, reset).

Navigation: Test that all navigation links (e.g., Home, About) direct users to the correct pages.

State Management: Ensure that React components maintain the correct state across interactions (e.g.,tracking logged-in status, handling input field changes).

Backend Services (Node.js): Test backend functions to ensure they return the expected results for variousscenarios.

User Authentication: Unit test the authentication service (e.g., validating credentials and token generation).
Order Management: Ensure the admin can create, edit, and delete items. Also, they update the ordertrack

5.3 Integration testing

Integration testing ensures that different modules and components work together seamlessly.

Key Integration Tests:

Frontend and Backend Interaction:

Account Creation: Test the integration between the front-end (React forms) and back-end (Node.js API) to ensure data is passed correctly and responses (such as success or error messages) are handled properly.

Login Authentication: Ensure the frontend login form correctly sends login credentials to the backend, receives authentication tokens, and allows access to restricted resources like the admin server.

Database Integration:

MongoDB Atlas: Ensure that data (users, posts, comments) is correctly stored in and retrieved from the database. For example, after creating a new post, verify that it appears in the relevant category and on the homepage.

Admin vs. User Roles: Verify that the role-based access control functions correctly:

- Admins can access dashboards, create Item, List Item, Update item and manage users order status.
- Regular users can ,review the food,view the food and order the food.

6. CONCLUSION

The Food and Delivery App is designed to provide a seamless and user-friendly platform for customers seeking to explore, order, and enjoy delicious meals from their favorite restaurants. By allowing users to browse menus, customize orders, and manage their carts, the app creates an enjoyable dining experience that meets modern consumer demands. The inclusion of features such as promo codes, cash on delivery, and secure payment options emphasizes the app's commitment to usability and customer satisfaction.

Utilizing a robust tech stack that includes React, Node.js, and MongoDB, the application ensures high performance and reliability. The architecture supports efficient data management and enables smooth interactions, fostering a positive user experience. System testing has been rigorously conducted, covering validation, unit, and integration tests to guarantee that the app meets its intended functionalities and user requirements.

As the app continues to grow, staying attuned to user feedback and technological advancements will be crucial. This proactive approach will allow the Food and Delivery App to evolve, ensuring it remains competitive and engaging for users. Future enhancements, such as personalized recommendations, improved order tracking, and advanced analytics, will further enrich user interactions and solidify the app’s position as a preferred choice for food delivery.

7. FUTURE ENHANCEMENT

Future enhancements for the Food and Delivery App could focus on improving user engagement and overall experience. One potential enhancement is the introduction of advanced user profiles that allow for greater customization, such as adding profile pictures and dietary preferences, enabling more personalized interactions.

Implementing an advanced search feature would enhance content discoverability, allowing users to filter menu items by categories, dietary restrictions, or price ranges. Additionally, integrating AI-driven recommendations based on user behavior and order history could further increase engagement by suggesting meals tailored to individual tastes.

To improve accessibility, expanding the app's capabilities to be mobile-responsive and multilingual would attract a broader audience, catering to diverse user needs. Enhancements to the analytics dashboard for restaurant partners would provide insights into customer preferences and sales trends, helping them optimize their offerings.

Strengthening security measures, such as incorporating two-factor authentication for user accounts, would ensure a safer environment for transactions and personal information. Lastly, implementing a feedback mechanism to gather user suggestions could foster community involvement and continuous improvement, making the app more dynamic and user-centric.

8. BIBLIOGRAPHY

https://reactjs.org/docs/getting-started.html

https://nodejs.org/en/docs/

https://docs.mongodb.com/

https://reactrouter.com/en/main

https://react-toastify.com/

https://lottiefiles.com/

https://youtu.be/DBMPXJJfQEA?feature=shared