

A PROJECT REPORT
On
FOOD DELIVERY WEBSITE

Submitted by

Student 1 (10800221001)
Student 2 (10800221062)
Student 3 (10800221068)
Student 4 (10800221089)

Under the Guidance of
Dr. Anup Kumar Mukhopadhyay
(Head of Department)



Information Technology
Asansol Engineering College
Asansol

Affiliated to
MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

05/2023



ASANSOL ENGINEERING COLLEGE

Kanyapur, Vivekananda Sarani, Asansol, Paschim Bardhaman, West Bengal - 713305
Phone: 225-3057, 225-2108 Telefax: (0341) 225-6334
E-mail: principal.aecwb@gmail.com Website: www.aecwb.edu.in

CERTIFICATE

Certified that this project report on “**food delivery website**” is the bonafide work of “**Student 1 (10800221001), Student 2 (10800221062), Student 3 (10800221068), Student 4 (10800221089)**” who carried out the project work under my supervision.

Dr. Anup Kumar Mukhopadhyay
(Head of Department)
(Information Technology)

Dr. Anup Kumar Mukhopadhyay
(Head of Department)
(Information Technology)

**Information Technology
Asansol Engineering College
Asansol**

ACKNOWLEDGEMENT

It is our great privilege to express my profound and sincere gratitude to our **Project Supervisor Dr. Anup Kumar Mukhopadhyay, Head of Department** for providing me with very cooperative and precious guidance at every stage of the present project work being carried out under his/her supervision. His valuable advice and instructions in carrying out the present study have been a very rewarding and pleasurable experience that has greatly benefitted us throughout our work.

We would also like to pay our heartiest thanks and gratitude to **Dr. Anup Kumar Mukhopadhyay, HoD**, and all the faculty members of the **Information technology**, Asansol Engineering College for various suggestions being provided in attaining success in our work.

Finally, we would like to express our deep sense of gratitude to our parents for their constant motivation and support throughout our work.

ANIL KUMAR (Co-Ordinate)

.....
(Student 1)

GAURAV KUMAR (Co-Ordinate)

.....
(Student 2)

ARUN KUMAR CHOURASIA (Lead - Frontend)

.....
(Student 3)

SOUMYADEEP BANERJEE (Lead - Backend)

.....
(Student 4)

Date: 25/05/2023

Place: Asansol

2nd Year

Department of Information Technology

CONTENT

Table of Contents

Certificate	ii
Acknowledgement	iii
Content	iv
List of Figures	v
1. Project Synopsis	vi-vii
1.1 Food Delivery Website	vi
1.2 The Server Module	vi
1.3 The Client Module	vii
2. Introduction	viii-x
2.1 Concept of Food Delivery Website	viii
2.2 Origin and need of Food Delivery Website in real life	ix
2.3 Idea Behind Food Delivery Website Development	x
3. Project Details	xi-li
3.1 Abstract	xi
3.2 Proposed Methodology	xii
3.3 Theories and Definitions	xiii-xiv
3.4 Outcomes Of The Project	xv
3.5 Working Flow Of The Project	xvi-li
4. Conclusion and Recommendations	lii
5. Reference	liii

LIST OF FIGURES

Table of Figure References

Figure 1	Flowchart of Food-Delivery Website	xvii
Figure 2	Output of the Food-Delivery Website “Server Screen”	1
Figure 3	Output of the Food-Delivery Website “Client Screen”	li

1. PROJECT SYNOPSIS

The Project Synopsis gives us a brief description on the project and it's working model. It gives us an overview of how the project has been handled by us through peaks and trenches making this work of ours a successful document to be showcased upon.

1.1 Food Delivery Website :

The Food Delivery Website is a project on MERN Stack Development using MongoDB, Express.js, React.js, Node.js . The aim of this project is to express how we can implement a simple food delivery website between a server and a client? The application is a desktop based application and is implemented using MongoDB as (NoSQL Database), Express.js (back-end framework of Node.js), React.js (front-end Javascript library), Node.js (back-end Javascript runtime environment) .

Application consists of two modules : The Server Module and the Client Module.

1.2 The Server Module :

The server module of a food delivery website is a crucial component responsible for managing the back-end operations and functionalities of the website. It serves as the central control system that handles requests from clients (web browsers or mobile apps), processes these requests, communicates with databases, manages user authentication, and orchestrates the various components of the application.

Here are some key aspects and responsibilities of the server module specifically tailored to a food delivery website:

1. Request Handling: The server module receives incoming HTTP requests from clients, which can include actions like browsing menus, placing orders, updating user profiles, and tracking deliveries.
2. Routing: Implement a routing system that directs incoming requests to the appropriate handlers or controllers based on the URL and HTTP method. For example, requests to `"/menu"` may be routed to a menu controller, while requests to `"/orders"` go to an order controller.
3. Business Logic: The server module houses the core business logic of the food delivery website. This logic includes processes such as menu management, order processing, restaurant selection, payment handling, and delivery tracking.
4. Database Interaction: Communicate with the database to perform various operations such as retrieving restaurant menus, storing user profiles, maintaining order history, and managing inventory levels. Ensure that the server uses secure database queries and transactions.

In summary, the server module of a food delivery website is the backbone of the application, responsible for processing requests, executing business logic, ensuring security, and providing the necessary functionality for users, restaurants, and delivery personnel. The choice of programming language, framework, and server hosting infrastructure should align with the specific requirements of your project.

1.3 The Client Module :

The client module of a food delivery website is the part of the application that runs on the user's device, typically a web browser or a mobile app. Its primary function is to interact with the server module and provide users with an interface to browse menus, place orders, track deliveries, and manage their accounts.

Here are some key aspects and responsibilities of the client module:

1. User Interface (UI):

- Develop a user-friendly and responsive user interface that allows users to easily navigate the website or app.
- Design an appealing layout for browsing restaurant menus, viewing item details, and managing orders.

2. User Experience (UX):

- Ensure a smooth and intuitive user experience throughout the application, from registration and menu selection to checkout and order tracking.
- Optimize performance to minimize loading times and provide real-time updates on order status.

3. Authentication and Authorization:

- Implement a secure login and registration process, allowing users to create accounts, log in, and recover passwords if necessary.
- Display personalized content and features based on user roles and permissions.

4. Menu Browsing:

- Provide an easy-to-navigate menu browsing experience that allows users to filter and search for restaurants and dishes based on preferences like cuisine, price, and dietary restrictions.
- Display detailed information about menu items, including descriptions, prices, ingredients, and customer reviews.

5. Order Placement:

- Enable users to add items to their cart, customize orders (e.g., specifying toppings or special instructions), and proceed to checkout.
- Implement a seamless and secure payment process, supporting various payment methods such as credit/debit cards, digital wallets, and cash on delivery.

6. Order Tracking:

- Offer real-time order tracking functionality that allows users to monitor the progress of their orders, including preparation, delivery, and estimated arrival time.
- Provide notifications or alerts to keep users informed about order updates.

7. Account Management:

- Allow users to manage their profiles, including updating personal information, delivery addresses, and payment methods.
- Maintain order history and provide access to previous orders for easy reordering.

In summary, the client module of a food delivery website or app is the user-facing part of the application that enables users to interact with the platform. It plays a critical role in delivering a positive user experience and should be designed with usability, performance, and security in mind.

2. INTRODUCTION

In today's fast-paced world, the convenience of having delicious, restaurant-quality meals delivered right to your doorstep has become a culinary revolution. Welcome to our Food Delivery Website, your gateway to a world of culinary delights, where the flavors of your favorite restaurants are just a click away.

2.1 Concept of Food Delivery Website :

The concept of a Food Delivery Website revolves around providing a convenient and efficient platform for users to order food from various restaurants and have it delivered to their desired location. This concept has gained immense popularity in recent years, driven by changing lifestyles, increased reliance on technology, and a growing demand for diverse dining options.

Here's a breakdown of the key elements that define the concept of a Food Delivery Website:

1. **Online Ordering:** Food delivery websites offer users the ability to browse through a wide selection of restaurants and cuisines, view menus, and place orders online. This eliminates the need for traditional phone calls and paper menus, making the ordering process quick and user-friendly.
2. **Restaurant Partnerships:** These platforms collaborate with a network of restaurants, ranging from local eateries to well-known chains. Each restaurant has its own dedicated page showcasing its menu items, prices, and customer reviews.
3. **Diverse Cuisine Options:** Food delivery websites typically offer a diverse range of cuisines, catering to various tastes and preferences. Users can explore everything from pizza and burgers to international cuisines like sushi, Thai, Indian, and more.
4. **Convenience:** The primary goal of these websites is to provide utmost convenience to users. They can place orders from the comfort of their homes or workplaces, saving time and effort in the process.
5. **Customization:** Users often have the option to customize their orders by specifying preferences, such as choosing toppings, selecting spice levels, or adding special instructions.
6. **Real-Time Tracking:** Many food delivery websites offer real-time order tracking, allowing users to monitor the progress of their orders from preparation to delivery. This feature enhances transparency and keeps users informed about the status of their food.
7. **Secure Payment Options:** These platforms provide secure payment processing, supporting various payment methods, including credit/debit cards, digital wallets, and cash on delivery.
8. **Reviews and Ratings:** Users can leave reviews and ratings for restaurants and individual dishes, helping others make informed decisions about their orders.

In essence, a Food Delivery Website brings together the culinary delights of numerous restaurants, user-friendly technology, and a commitment to convenience and quality. It serves as a bridge between restaurants and customers, making it easier than ever for people to enjoy their favorite meals in the comfort of their homes or offices.

2.2 Origin and need of Food Delivery Website in real life :

The origin and need for food delivery websites in real life can be traced back to the evolving lifestyles, technological advancements, and changing consumer preferences. Here's a look at the factors that contributed to the emergence and demand for food delivery websites:

1. Changing Lifestyles:

- **Fast-Paced Lives:** Modern life is often characterized by busy schedules, long working hours, and limited time for meal preparation.
- **Commuting:** Lengthy commutes to work can leave little time and energy for cooking, leading people to seek convenient dining options.
- **Urbanization:** The growth of urban areas has led to an increase in single-person households, where cooking for one may not be efficient.

2. Technology Adoption:

- **Internet Penetration:** Widespread internet access and smartphone usage have made it easy for people to connect with online platforms.
- **Convenience Apps:** The popularity of mobile apps for various services has trained consumers to expect convenient digital solutions for everyday tasks.

3. Variety and Choice:

- **Diverse Cuisine:** Consumers today have a more sophisticated palate and are eager to explore a wide range of cuisines, from local to international.
- **Restaurant Partnerships:** Food delivery websites partner with a multitude of restaurants, providing users with a vast array of dining choices.

4. Time Efficiency:

- **Time-Saving:** Ordering food online eliminates the need to travel to a restaurant, place orders in person, or wait for food to be prepared.
- **Multitasking:** People can place food orders while working, studying, or attending to other responsibilities.

5. Changing Food Culture:

- **Food as an Experience:** Dining has become more experiential, with consumers valuing the convenience of enjoying restaurant-quality meals at home.
- **Social Sharing:** The rise of social media has encouraged people to share their dining experiences, including food deliveries.

6. Work-Life Balance:

- **Work-Life Integration:** Maintaining a work-life balance is a priority for many. Ordering food online can be a convenient solution on busy workdays.

In summary, the need for food delivery websites emerged from the fast-paced, technology-driven, and convenience-focused lifestyles of today's consumers. These platforms offer a solution to the time constraints and desire for culinary variety that many people face. Additionally, they play a crucial role in providing safe dining options during challenging times, such as the COVID-19 pandemic. As a result, food delivery websites have become an integral part of modern dining culture, offering a convenient and diverse array of culinary experiences to users.

2.3 Idea Behind Food Delivery Website Development :

Certainly, the development of a food delivery website is grounded in several key ideas and concepts that shape its purpose and functionality. Here are three central ideas behind the development of food delivery websites:

1. Convenience and Accessibility:

- Idea: The primary idea behind food delivery website development is to provide unparalleled convenience and accessibility to users. It aims to make ordering food as effortless as possible, eliminating the need for users to cook at home or dine out.

- Implementation: This concept is implemented by offering an easy-to-use platform that allows users to browse menus, place orders, and track deliveries from the comfort of their homes or workplaces. It also involves providing a diverse range of restaurant options to cater to different tastes and dietary preferences.

2. Time-Saving and Efficiency:

- Idea: Food delivery websites are designed to save users time and enhance efficiency in the dining experience. They aim to streamline the process of ordering food, reducing waiting times and minimizing the effort required.

- Implementation: This concept is put into practice through features such as quick and intuitive ordering interfaces, real-time order tracking, and secure online payment options. By offering a faster and more efficient way to get food, these websites align with the modern, fast-paced lifestyle of many individuals.

3. Diverse Culinary Exploration:

- Idea: Food delivery websites encourage culinary exploration and diversity. They aim to expose users to a wide variety of cuisines, flavors, and restaurant choices beyond their immediate geographic location.

- Implementation: To bring this idea to life, food delivery websites partner with numerous restaurants, from local eateries to renowned establishments. They provide a platform for users to explore and experiment with different foods, making it easy to discover new dishes and culinary traditions. User reviews and ratings also play a role in helping others make informed choices and expand their gastronomic horizons.

In summary, food delivery website development is centered on the core ideas of convenience, time-saving efficiency, and culinary diversity. These platforms are designed to meet the evolving needs and preferences of consumers by offering a seamless and accessible way to order a wide range of delicious meals from a variety of restaurants, all while saving time and effort in the process.

3. PROJECT DETAILS

For A Successful Project Report to be build , a project must contain some of the specifications to make the report more informative and more representative to the client whom we deal with.

The following Project Specifications are the major Distinctions which make our Project a more desirable one.

- The Project is more **SPECIFIC** that is it is Specific to the Topics and does not contain any useless or uninformative data that makes the project hard to sustain.
- The Project is **MEASURABLE** that is our project yields a clearly defined project that is measurable in terms of both achievements and benefits.
- Our Project is **ACHIEVABLE** that is although we talk more about ambitions, we need to know that our project should meet our expectations and can be achieved.
- The Project is **RELEVANT** that means the project must guarantee to be the most efficient in overall aspects of its production, otherwise it becomes irrelevant to its entity.
- The Project Must be **TIME BOUND** which suggests that our project comes within a time JFrame for its completion including its Planning Development and Execution.

The above mentioned characteristics are jot down of our project report and helps in identification and knowledge of our project.

3.1 ABSTRACT :

The Food Delivery Website is a user-friendly online platform designed to provide unmatched convenience in accessing a diverse range of restaurant-quality meals. With a focus on saving time and effort, our website simplifies the ordering process, allowing users to explore menus, customize orders, and track deliveries in real-time. We prioritize safety and quality, ensuring that partner restaurants adhere to rigorous hygiene standards. The platform offers an array of culinary experiences, from local gems to renowned establishments, making it easy to satisfy diverse tastes and preferences. Loyalty programs and promotions reward users for their trust and loyalty. Join us in redefining how you enjoy food, one delicious meal at a time. Discover the ultimate dining convenience with the Food Delivery Website. Our platform is designed for efficiency, streamlining the ordering process and eliminating wait times. Explore a world of flavors from around the globe, from local favorites to international cuisine, all from the comfort of your home. Real-time order tracking keeps you informed, ensuring your meal arrives exactly when you want it. We prioritize your safety, with partner restaurants upholding strict hygiene standards. Enjoy cost-saving benefits through loyalty programs and exclusive promotions. Join us in redefining the way you dine, making every meal a delightful experience. With the Food Delivery Website, your culinary cravings are just a click away. Our user-friendly interface makes ordering a breeze, so you can enjoy restaurant-quality meals without leaving your doorstep. Whether it's a busy workday or a relaxing weekend, our platform offers a seamless dining experience. Real-time order tracking ensures you're always in the know, from kitchen to delivery.

3.2 PROPOSED METHODOLOGY :

Developing a food delivery website using the MERN (MongoDB, Express.js, React, Node.js) stack involves several key steps and methodologies.

Here's a proposed methodology for building such a website:

1. Project Planning and Requirements Gathering:

- Define the scope of your project and gather detailed requirements.
- Identify target audience, features, and objectives.

2. Technology Selection:

- Choose the MERN stack for your project, including MongoDB for the database, Express.js for the backend, React for the frontend, and Node.js as the runtime environment.

3. Architecture Design:

- Create a high-level architectural design for your food delivery website, including data flow, user interfaces, and database structure.
- Define the API endpoints that will facilitate communication between the frontend and backend.

4. Database Design:

- Design the database schema to store user profiles, restaurant data, menu items, orders, and other relevant information.
- Ensure data integrity and security.

5. Backend Development:

- Develop the server-side application using Node.js and Express.js.
- Implement user authentication and authorization mechanisms for user accounts.
- Create API endpoints for handling restaurant listings, menu items, user profiles, and order processing.
- Integrate payment gateways for secure online transactions.
- Implement real-time order tracking using WebSockets or similar technologies.

6. Frontend Development:

- Develop the user interface using React.js, ensuring a responsive and user-friendly design.
- Create components for browsing restaurant listings, viewing menus, customizing orders, and managing user profiles.
- Implement real-time order tracking and notifications.
- Optimize the frontend for performance and user experience.

7. Testing and Quality Assurance:

- Conduct thorough testing of both the frontend and backend, including unit testing, integration testing, and user acceptance testing.
- Identify and fix bugs and issues.

By following this proposed methodology, we can efficiently develop a food delivery website using the MERN stack, providing a seamless and user-friendly platform for ordering and enjoying meals from a variety of restaurants.

3.3 THEORIES AND DEFINITIONS :

THEORIES :

There are several theories and concepts that underlie the development and operation of food delivery websites. These theories help shape the strategies and practices employed in the food delivery industry. Here are some key theories and concepts related to food delivery websites:

1. Convenience Theory:

- This theory suggests that the primary appeal of food delivery websites is the convenience they offer to consumers. It acknowledges that people value ease of use and time-saving solutions, especially in today's fast-paced world. Food delivery websites align with this theory by streamlining the process of ordering and delivering food to customers' doors.

2. Choice Theory:

- Choice theory posits that consumers value having a variety of options and the ability to make choices that align with their preferences. Food delivery websites provide extensive menus from various restaurants, enabling users to select from a wide range of cuisines and dishes.

3. Transaction Cost Theory:

- Transaction cost theory suggests that individuals seek to minimize transaction costs, including time and effort, when engaging in economic activities. Food delivery websites reduce transaction costs associated with dining out or cooking by simplifying the process of ordering and delivering food.

4. Technology Adoption Theory:

- This theory explains how people adopt and use new technologies based on perceived usefulness and ease of use. Food delivery websites leverage technology to make ordering food more accessible and user-friendly, encouraging adoption among consumers.

5. Network Effects Theory:

- Network effects theory posits that the value of a platform or service increases as more users join it. Food delivery websites benefit from network effects, as a larger user base attracts more restaurants and vice versa. This theory drives competition and growth in the industry.

6. Psychology of Food Choice:

- Understanding the psychology of food choice is essential for food delivery websites. Concepts like taste preferences, cultural influences, and emotional connections to food play a significant role in menu design, marketing, and user experience.

7. Economic Theories of Pricing:

- Economic theories, such as supply and demand, pricing strategies, and cost analysis, guide how food delivery websites set prices for menu items, delivery fees, and service charges.

These theories and concepts provide a foundation for the development, operation, and continuous improvement of food delivery websites. They help businesses understand user behaviour, market dynamics, and the evolving preferences of consumers, ultimately shaping the strategies and services offered in the food delivery industry.

DEFINITIONS :

A food delivery website is an online platform or website that allows users to browse menus, place orders for food items, and have those orders delivered to their specified location, typically their homes or workplaces. These websites serve as intermediaries between users and a network of partner restaurants, facilitating the entire food ordering and delivery process.

Key components of a food delivery website typically include:

1. Restaurant Listings: A directory of partner restaurants, each with its own dedicated page displaying menus, prices, and customer reviews.
2. Menu Browsing: The ability for users to browse restaurant menus, view item details, and select dishes they want to order.
3. Order Placement: Tools for users to add items to a virtual shopping cart, customize orders (e.g., specify toppings or special instructions), and complete the checkout process.
4. Payment Processing: Secure online payment options for users to pay for their orders, including credit/debit cards, digital wallets, and cash on delivery.
5. Order Tracking: Real-time tracking features that allow users to monitor the status of their orders, from preparation to delivery, often with estimated arrival times.
6. User Authentication: User registration and login functionalities, enabling users to create accounts, save preferences, and track order history.
7. Reviews and Ratings: The ability for users to leave reviews and ratings for restaurants and individual dishes, helping others make informed choices.
8. Delivery Coordination: Integration with delivery drivers or services to ensure efficient and timely food deliveries.
9. Promotions and Loyalty Programs: Offers, discounts, and loyalty programs to incentivize repeat business and reward loyal customers.
10. Customer Support: Channels for users to seek assistance, provide feedback, and resolve issues related to their orders or the platform.

Food delivery websites have gained immense popularity due to their convenience, variety of dining options, and ability to cater to the changing lifestyles of consumers. They play a pivotal role in connecting users with a wide array of culinary experiences, making it easier than ever to enjoy restaurant-quality meals in the comfort of one's own location.

3.4 OUTCOMES OF THE PROJECT :

The outcomes of a food delivery website can have a significant impact on various stakeholders, including users, restaurants, and the website operators. These outcomes are often both positive and negative and can vary depending on how well the platform is executed. Here are some of the key outcomes associated with a food delivery website:

****For Users:**

1. Convenience: Users benefit from the convenience of ordering food from a variety of restaurants without leaving their homes or workplaces.
2. Time Savings: Food delivery websites save users time that would otherwise be spent on cooking or dining out. This is especially valuable for individuals with busy schedules.
3. Diverse Dining Options: Users have access to a wide range of culinary options, from local favourites to international cuisines, allowing them to explore and experiment with different flavors.
4. Cost Savings: Loyalty programs, discounts, and promotions offered by food delivery websites can provide cost savings for users who order frequently.

****For Restaurants:**

1. Increased Visibility: Partnering with a food delivery website can increase a restaurant's visibility and attract a broader customer base.
2. Additional Revenue Stream: Restaurants can tap into a new revenue stream by accepting online orders through the platform.
3. Efficiency: Food delivery websites often provide tools for efficient order management and delivery coordination, improving the overall operational efficiency of restaurants.

****For Website Operators:**

1. Revenue Generation: Operators of food delivery websites can generate revenue through various means, including commission fees from restaurants, delivery charges, and premium subscription plans for users.
2. User Growth: Successful websites can experience significant user growth, leading to a larger customer base and potential for increased revenue.
3. Data Insights: Operators can gather valuable data on user behaviour, preferences, and ordering trends, which can inform business decisions and marketing strategies.

In summary, the outcomes of a food delivery website can be positive for users, restaurants, and operators alike, offering convenience, revenue opportunities, and access to a diverse range of dining options. However, challenges such as competition and logistical complexities also come into play, making it important for operators to continually adapt and improve their platforms.

3.5 WORKING FLOW OF THE PROJECT :

The working flow of a food delivery website developed using the MERN (MongoDB, Express.js, React, Node.js) stack involves several interconnected components and processes.

Here's a step-by-step working flow:

1. User Registration and Login:

- Users begin by registering on the website or logging in if they already have an account.
- User authentication and authorization mechanisms ensure that only registered users can place orders and access certain features.

2. Menu Exploration and Selection:

- Users can explore restaurant menus, view item details, and select dishes they want to order.
- They can customize their orders, specifying options like toppings or special instructions.

3. Adding to Cart and Order Review:

- Users add selected items to a virtual shopping cart.
- They can review their order, make modifications, and see the total cost.

4. Order Placement and Payment:

- Users proceed to checkout, where they provide delivery details, such as the address and contact information.
- Secure online payment options, including credit/debit cards and digital wallets, are available for users to complete the transaction.
- Some users may choose the "cash on delivery" option, where payment is made upon delivery.

5. Order Confirmation:

- Upon successful payment, users receive an order confirmation with details of the order and estimated delivery time.

6. Order Processing:

- The order is transmitted to the restaurant's backend system.
- The restaurant begins preparing the food.

7. Real-Time Order Tracking:

- Users can track the status of their order in real-time, from the restaurant's kitchen to the delivery process.
- Updates on order preparation, dispatch, and estimated arrival time are provided.

8. Delivery Coordination:

- The website's backend system coordinates with delivery personnel or services to ensure timely and efficient food delivery.
- Drivers receive order details and navigation instructions.

This working flow illustrates the seamless process from user registration and menu browsing to order placement, delivery, and user feedback. The MERN stack provides the technology foundation to ensure that each step operates efficiently and effectively, delivering a smooth user experience.

FLOWCHART

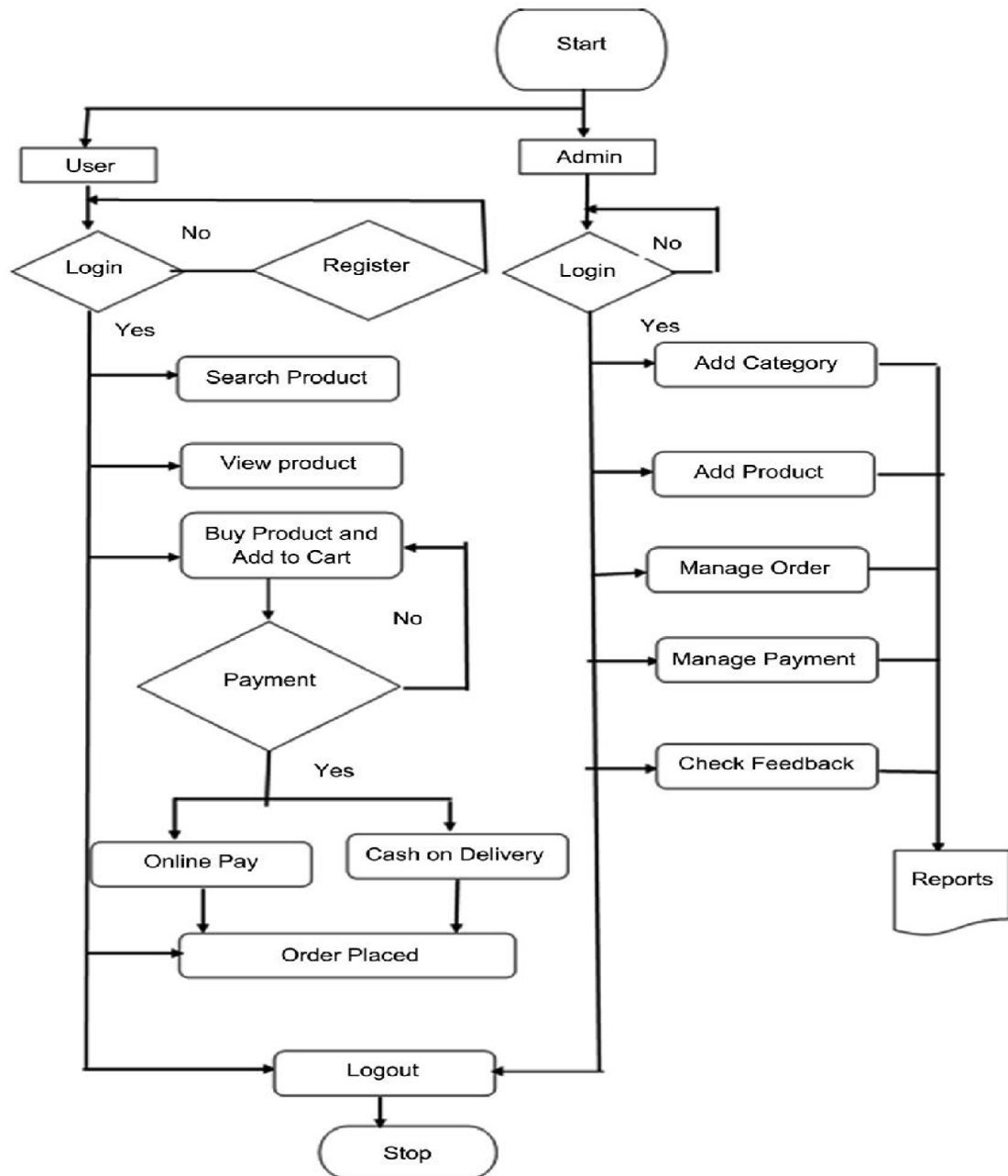


Figure 1 : Flowchart of Food-Delivery Website

SOURCE CODE

#Mern – Pizza Food Delivery Website :

1. Source Code for Backend / Server page :

- Server.js

```
const express = require("express");

const Pizza = require('./models/pizzaModel')

const app = express();
const db = require("./db.js")
app.use(express.json());

const pizzasRoute = require('./routes/pizzasRoute')
const userRoute = require('./routes/userRoute')
const ordersRoute = require('./routes/ordersRoute')

app.use('/api/pizzas/', pizzasRoute)
app.use('/api/users/' , userRoute)
app.get("/", (req, res) => {
    res.send("Server working" + port);
});

app.get("/getpizzas", (req, res) => {

    Pizza.find({} , (err , docs)=>{

        if(err){
            console.log(err);
        }
        else{
            res.send(docs)
        }

    })

});

const port = process.env.PORT || 8000;

app.listen(port, () => `Server running on port port`);
```

- db.js

```
const mongoose = require("mongoose");

var mongoURL =
'mongodb+srv://Database:Database121@cluster0.iozkoal.mongodb.net/mern-pizza'

mongoose.connect(mongoURL , {useUnifiedTopology:true , useNewUrlParser:true})

var db = mongoose.connection

db.on('connected' , ()=>{
  console.log('Mongo DB Connection Successfull');
})

db.on('error' , ()=>{
  console.log('Mongo DB Connection failed');
})

module.exports = mongoose
```

2. Source Code for Frontend / Client page :

- index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
      https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
    awesome/5.15.3/css/all.min.css" integrity="sha512-
    gMjQeDaELJ0ryCI+FtItusU9MkAifCZcGq789FrzkiM49D8lbDhoaUaIX4ASU187wofMNlgBJ4ckbrX
    M9sE6Pg==" crossorigin="anonymous">
```

```

    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-
free@5.15.3/css/fontawesome.min.css" integrity="sha384-
QYIZto+st3yW+o8+50HfT6S482Zsvz2Wf0zpFSXMF9zqeLcFV0/wlZpMtyFcZALm"
crossorigin="anonymous">
    <!--
    Notice the use of %PUBLIC_URL% in the tags above.
    It will be replaced with the URL of the 'public' folder during the build.
    Only files inside the 'public' folder can be referenced from the HTML.

    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a non-root public URL.
    Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
</head>
<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
    -->
</body>
</html>

```

- cartActions.js

```

export const addToCart=(pizza , quantity , varient)=>(dispatch , getState)=>{

    var cartItem = {
        name : pizza.name ,
        _id : pizza._id,
        image : pizza.image ,
        varient : varient ,
        quantity : Number(quantity) ,
        prices : pizza.prices ,
        price : pizza.prices[0][varient]* quantity

    }

    if(cartItem.quantity>10)

```

```

{
  alert('You cannot add more than 10 quantities')
}
else{
  if(cartItem.quantity<1)
  {
    dispatch({type:'DELETE_FROM_CART' , payload:pizza})
  }
  else{
    dispatch({type:'ADD_TO_CART' , payload : cartItem})
  }
}

dispatch({type:'ADD_TO_CART' , payload : cartItem})

const cartItems = getState().cartReducer.cartItems
localStorage.setItem('cartItems' , JSON.stringify(cartItems))
}

export const deleteFromCart=(pizza)=>(dispatch , getState)=>{

  dispatch({type:'DELETE_FROM_CART' , payload:pizza})

  localStorage.setItem('cartItems' , JSON.stringify(cartItems))
}

```

- orderActions.js

```

import axios from "axios";
export const placeOrder=(token , subtotal)=>async (dispatch , getState)=>{

  dispatch({type:'PLACE_ORDER_REQUEST'})
  const currentUser = getState().loginUserReducer.currentUser
  const cartItems = getState().cartReducer.cartItems

  try {

    const response = await axios.post('/api/orders/placeorder' , {token ,
subtotal , currentUser , cartItems})
    dispatch({type:'PLACE_ORDER_SUCCESS'})
    console.log(response);

  } catch (error) {
    dispatch({type:'PLACE_ORDER_FAILED'})
    console.log(error);
  }
}

```

```

    }
  }

export const getUserOrders=()=>async (dispatch,getState)=>{

  const currentUser = getState().loginUserReducer.currentUser
  dispatch({type: 'GET_USER_ORDERS_REQUEST'})

  try {
    const response = await axios.post('/api/orders/getuserorders' ,
{userid : currentUser._id})
    console.log(response);
    dispatch({type: 'GET_USER_ORDERS_SUCCESS' , payload : response.data})
  } catch (error) {
    dispatch({type: 'GET_USER_ORDERS_FAILED' , payload : error})
  }
}

```

- pizzaActions.js

```

import axios from "axios";
export const getAllPizzas=()=>async dispatch=>{

  dispatch({type: 'GET_PIZZAS_REQUEST'})

  try {
    const response = await axios.get('/api/pizzas/getallpizzas')
    console.log(response);
    dispatch({type: 'GET_PIZZAS_SUCCESS' , payload : response.data})
  } catch (error) {
    dispatch({type: 'GET_PIZZAS_FAILED' , payload : error})
  }
}

```

- userActions.js

```

import axios from 'axios';
export const registerUser=()=>async dispatch=>{

  dispatch({type: 'USER_REGISTER_REQUEST'})

  try {
    const response = await axios.post('/api/users/register' , user)
    console.log(response);
    dispatch({type: 'USER_REGISTER_SUCCESS'})
  } catch (error) {
    dispatch({type: 'USER_REGISTER_FAILED' , payload: error})
  }
}

```

```

    }
  }

export const loginUser=()=>async dispatch=>{

  dispatch({type:'USER_LOGIN_REQUEST'})

  try {
    const response = await axios.post('/api/users/login' , user)
    console.log(response);
    dispatch({type:'USER_LOGIN_SUCCESS' , payload: response.data})
    localStorage.setItem('currentUser' , JSON.stringify(response.data))
    window.location.href='/'
  } catch (error) {
    dispatch({type:'USER_LOGIN_FAILED' , payload: error})
  }
}

export const logoutUser=()=>dispatch=>{

  localStorage.removeItem('currentUser')
  window.location.href='/login'

}

```

- Checkout.js

```

import React from 'react';
import { useDispatch, useSelector } from 'react-redux';
import StripeCheckout from 'react-stripe-checkout';
import { placeOrder } from '../actions/orderActions';
import Error from '../components/Error';
import Loading from '../components/Loading';
import Success from '../components/Success';
export default function Checkout({subtotal}) {

  const orderstate = useSelector((state) => state.placeOrderReducer)
  const {loading , error , success} = orderstate
  const dispatch = useDispatch()
  function tokenHandler(token)
  {
    console.log(token);
    dispatch(placeOrder(token , subtotal))
  }
}

```

```

return (
  <div>

    {loading && (<Loading/>)}
    {error && (<Error error='Something went wrong'/>)}
    {success && (<Success success='Your Order Placed Successfully'/>)}
    <StripeCheckout amount={subtotal} shippingAddress token={tokenHandler}
stripekey={pk_test_51MhA3KSJhyrIFUV9TuUTRkIxU0qnIvhiJCu1pxS7ERkrczBdoapWAAgNuiuv8bCTKuYwMxviTp3gQYsnIizpjWAu00TY80dcblpk_test_51MhA3KSJhyrIFUV9TuUTRkIxU0qnIvhiJCu1pxS7ERkrczBdoapWAAgNuiuv8bCTKuYwMxviTp3gQYsnIizpjWAu00TY80dcbl}
currency='INR'>

      <button className='btn'>Pay Now</button>

    </StripeCheckout>

  </div>
)
}

```

- Error.js

```

import React from 'react';

export default function Error({error}) {
  return (
    <div>
      <div className="alert alert-danger" role="alert">
        {error}
      </div>
    </div>
  );
}

```

- Loading.js

```

import React from 'react'

export default function Loading(){
  return (
    <div className="text-center">
      <div className="spinner-border" role="status" style={{height:'100px' ,
width:'100px' , marginTop:'100px'}}>
        <span className="sr-only">Loading...</span>
      </div>
    </div>
  );
}

```



```
}
```

- Navbar.js

```
import React from "react";
import { useSelector, useDispatch } from 'react-redux';
import { logoutUser } from "../actions/userActions";
export default function Navbar() {
  const cartstate = useSelector(state => state.cartReducer);
  const userstate = useSelector(state => state.loginUserReducer);
  const { currentUser } = userstate;
  const dispatch = useDispatch()
  return (
    <div>
      <nav className="navbar navbar-expand-lg shadow-lg p-3 mb-5 bg-white rounded">
        <a className="navbar-brand" href="/">PIZZA-PRO</a>
        <button className="navbar-toggler"
          type="button"
          data-toggle="collapse"
          data-target="#navbarNav"
          aria-controls="navbarNav"
          aria-expanded="false"
          aria-label="Toggle navigation">
          <span className="navbar-toggler-icon"></span>
        </button>
        <div className="collapse navbar-collapse" id="navbarNav">
          <ul className="navbar-nav ml-auto">
            {currentUser ? (<div class="dropdown mt-2">
              <a style={{color:'black'}} className="dropdown-
toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown" aria-
haspopup="true" aria-expanded="false">
                {currentUser.name}
              </a>
              <div className="dropdown-menu" aria-
labelledby="dropdownMenuButton">
                <a className="dropdown-item"
href="/orders">Orders</a>
                <a className="dropdown-item" href="#"
onclick={()=>{dispatch(logoutUser())}}><li>Logout</li></a>
              </div>
            </div>) : (<li className="nav-item">
              <a className="nav-link" href="/login">
                Login
              </a>
            </li>)}
          </ul>
        </div>
      </nav>
    </div>
  )
}
```

```

        <li className="nav-item">
          <a className="nav-link" href="/cart">
            Cart {cartstate.cartItems.length}
          </a>
        </li>
      </ul>
    </div>
  </nav>
</div>
)
}

```

- Pizza.js

```

import React, { useState } from 'react';
import { Modal } from 'react-bootstrap';
import { useDispatch, useSelector } from 'react-redux';
import { addToCart } from '../actions/cartActions';
export default function Pizza({ pizza }) {
  const [quantity, setquantity] = useState(1)
  const [varient, setvarient] = useState('small');
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  const dispatch = useDispatch()
  function addtocart()
  {
    dispatch(addToCart(pizza , quantity , varient))
  }
  return (
    <div className="shadow-1g p-3 mb-5 bg-white rounded" key={pizza._id}>

      <div onClick={handleShow}>
        <h1>{pizza.name}</h1>
        <img src={pizza.image} className="img-fluid" style={{ height:
'200px', width: '200px' }} />
      </div>

      <div className="flex-container">

        <div className='w-100 m-1'>
          <p>Varients</p>
          <select className='form-control' value={varient}
onChange={(e) => { setvarient(e.target.value) }
}>
            {pizza.varients.map(varient => {

```

```

        return <option value={varient}>{varient}</option>
      })}
    </select>
  </div>

  <div className='w-100 m-1'>
    <p>Quantity</p>
    <select className='form-control' value={quantity}
onChange={(e) => { setquantity(e.target.value) }}>
      {[...Array(10).keys()].map((x, i) => {

        return <option value={i + 1}>{i + 1}</option>

      })}
    </select>
  </div>
</div>

<div className="flex-container">
  <div className='m-1 w-100'>
    <h1 className='mt-1'>Price : {pizza.prices[0][varient] *
quantity} Rs/-</h1>
  </div>
  <div className='m-1 w-100'>
    <button className="btn" onClick={addtocart}>ADD TO
CART</button>
  </div>
</div>

<Modal show={show} onHide={handleClose}>
  <Modal.Header closeButton>
    <Modal.Title>{pizza.name}</Modal.Title>
  </Modal.Header>

  <Modal.Body>
    <img src={pizza.image} style={{height: '400px'}}></img>
    <p>{pizza.description}</p>
  </Modal.Body>

  <Modal.Footer>
    <button className="btn"
onClick={handleClose}>CLOSE</button>
  </Modal.Footer>
</Modal>

</div>
)
}

```

- Success.js

```
import React from 'react';

export default function Success({ success }) {
  return (
    <div>
      <div className="alert alert-success" role="alert">
        {success}
      </div>
    </div>
  );
}
```

- cartReducer.js

```
export const cartReducer=(state={cartItems : []} , action)=>{

  switch (action.type)
  {
    case 'ADD_TO_CART' :

      const alreadyExists = state.cartItems.find(item=>
item._id===action.payload._id)
      if(alreadyExists)
      {
        return{
          ...state ,
          cartItems : state.cartItems.map(item=>
item._id===action.payload._id ? action.payload : item)
        }
      }
      else{
        return{

          ...state ,
          cartItems:[...state.cartItems , action.payload]

        }
      }
    case 'DELETE_FROM_CART' : return{

      ...state ,
      cartItems : state.cartItems.filter(item =>
item._id !==action.payload._id)
    }
  }
}
```

```

    }
    default : return state
  }
}

```

- orderReducer.js

```

export const placeOrderReducer =(state={}, action) =>{

  switch(action.type)
  {
    case 'PLACE_ORDER_REQUEST' : return{
      loading:true
    }
    case 'PLACE_ORDER_SUCCESS' : return{
      loading:false,
      success:true
    }
    case 'PLACE_ORDER_FAILED' : return{
      loading:false,
      error:action.payload
    }
    default : return state
  }
}

export const getUserOrdersReducer=(state={orders : []} , action)=>{

  switch(action.type)
  {
    case 'GET_USER_ORDERS_REQUEST' : return{
      loading : true,
      ...state
    }
    case 'GET_USER_ORDERS_SUCCESS' : return{
      loading : false ,
      orders : action.payload
    }
    case 'GET_USER_ORDERS_FAILED' : return{
      error : action.payload ,
      loading : false
    }
    default : return state
  }
}

```

```
}
```

- pizzaReducers.js

```
export const getAllPizzasReducer=(state={pizzas : []} , action)=>{

  switch(action.type)
  {
    case 'GET_PIZZAS_REQUEST' : return{
      loading : true,
      ...state
    }
    case 'GET_PIZZAS_SUCCESS' : return{
      loading : false ,
      pizzas : action.payload
    }
    case 'GET_PIZZAS_FAILED' : return{
      error : action.payload ,
      loading : false
    }
    default : return state
  }
}
```

- userReducer.js

```
export const registerUserReducer =(state={} ,action) =>{

  switch(action.type)
  {
    case 'USER_REGISTER_REQUEST' : return{
      loading:true
    }
    case 'USER_REGISTER_SUCCESS' : return{
      loading:false,
      success:true
    }
    case 'USER_REGISTER_FAILED' : return{
      loading:false,
      error:action.payload
    }
    default : return state
  }
}

export const loginUserReducer =(state={} ,action) =>{
```

```

switch(action.type)
{
  case 'USER_LOGIN_REQUEST' : return{
    loading:true
  }
  case 'USER_LOGIN_SUCCESS' : return{
    loading:false,
    success:true,
    currentUser : action.payload
  }
  case 'USER_LOGIN_FAILED' : return{
    loading:false,
    error:action.payload
  }
  default : return state
}
}

```

- Cartscreen.js

```

import React from 'react';
import {useSelector , useDispatch} from 'react-redux';
import {addToCart} from '../actions/cartActions';
import { deleteFromCart } from '../actions/cartActions';
export default function Cartscreen() {

  const cartstate = useSelector(state=>state.cartReducer)
  const cartItems = cartstate.cartItems
  var subtotal = cartItems.reduce((x , item)=> x+item.price , 0)
  const dispatch = useDispatch()
  return (
    <div>
      <div className="row justify-content-center">

        <div className="col-md-6">
          <h2 style={{fontSize:'40px'}}>My Cart</h2>

          {cartItems.map(item=>{
            return <div className="flex-container">

              <div className='text-left m-1 w-100'>
                <h1>{item.name} [{item.varient}]</h1>
                <h1>Price : {item.quantity} *
{item.prices[0][varient]} = {item.price}</h1>
                <h1 style={{display:'inline-block'}}>Quantity :
</h1>

```

```

        <i className="fa fa-plus" aria-hidden="true"
onClick={()=>{dispatch(addToCart(item , item.quantity+1 , item.varient))}}></i>
        <b>{item.quantity}</b>
        <i className="fa fa-minus" aria-hidden="true"
onClick={()=>{dispatch(addToCart(item , item.quantity-1 , item.varient))}}></i>
        <hr/>
    </div>

    <div className='m-1 w-100'>
        
    </div>
    <div className='m-1 w-100'>
        <i className="fa fa-trash mt-5" aria-hidden="true"
onClick={()=>{dispatch(deleteFromCart(item))}}></i>
    </div>

    </div>
    }}

</div>

<div className="col-md-4 text-right">
    <h2 style={{fontSize:'45px'}}>SubTotal : {subtotal} /-</h2>
    <Checkout subtotal={subtotal}/>
</div>

</div>
</div>
)
}

```

- Homescreen.js

```

import React, { useState, useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { getAllPizzas } from '../actions/pizzaActions';
import Error from '../components/Error';
import Pizza from '../components/Pizza';
export default function Homescreen() {
    const dispatch = useDispatch()

    const pizzasstate = useSelector(state => state.getAllPizzasReducer)

    const { pizzas, error, loading } = pizzasstate

    useEffect(() => {
        dispatch(getAllPizzas())
    })
}

```



```

    }, [])

    return (
      <div>
        <div className="row justify-content-center">
          {loading ? (<Loading />) : error ? (<Error error='Something
went wrong' />)} : (
            pizzas.map((pizza) => {
              return (<div className="col-md-3 m-3" key={pizza._id}>
                <div>
                  <Pizza pizza={pizza} />
                </div>
              </div>
            );
          })
        </div>
      </div>
    )
  }
}

```

- Loginscreen.js

```

import React, { useState, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { loginUser } from "../actions/userActions";
import Error from "../components/Error";
import Loading from "../components/Loading";
export default function Loginscreen() {

  const [email, setemail] = useState("");
  const [password, setpassword] = useState("");
  const loginstate = useSelector(state=>state.loginUserReducer);
  const {loading, error} = loginstate
  const dispatch = useDispatch()

  useEffect(() => {

    if(localStorage.getItem('currentUser'))
    {
      window.location.href='/'
    }

  }, [])

  function login(){
    const user={email, password}
  }

```

```

    dispatch(loginUser(user))
  }

  return (
    <div>
      <div className="row justify-content-center mt-5">
        <div className="col-md-5 mt-5 text-left shadow-lg p-3 mb-5 bg-white rounded">
          <h2 className="text-center m-2" style={{ fontSize: '35px' }}>Login</h2>

          {loading && (<loading />)}
          {error && (<Error error='Invalid Credentials' />)}

          <div>
            <input required type="text" placeholder="name"
className="form-control" value={name} onChange={(e)=>{setname(e.target.value)}}
/>
            <input required type="text" placeholder="email"
className="form-control" value={email}
onChange={(e)=>{setemail(e.target.value)}} />
            <input required type="text" placeholder="password"
className="form-control" value={password}
onChange={(e)=>{setpassword(e.target.value)}} />
            <input required type="text" placeholder="confirm password"
className="form-control" value={cpassword}
onChange={(e)=>{setcpassword(e.target.value)}} />
            <button onClick={login} className="btn mt-3 mb-3">LOGIN</button>
            <br/>
            <a style={{color:'black'}} href="/register" className="mt-2">Click Here to Register</a>
          </div>
        </div>
      </div>

      <div></div>
    </div>
  )
}

```

- Ordersscreen.js

```

import React , {useState, useEffect} from 'react'
import {useDispatch , useSelector} from 'react-redux'
import { getUserOrders } from '../actions/orderActions'
import Loading from '../components/Loading'

```

```

export default function Ordersscreen() {

  const dispatch = useDispatch()
  const orderstate = useSelector(state=>state.getUserOrdersReducer)
  const {orders , error , loading} = orderstate

  useEffect(() => {

    dispatch(getUserOrders())

  }, [])

  return (
    <div>
      <h2 style={{fontSize:'35px'}}>My Orders</h2>
      <hr/>
      <div className="row justify-content-center">
        {loading && (<Loading/>)}
        {error && (<Error error='Something went wrong'/>)}
        {orders && orders.map(order=>{
          return <div className="col-md-8 m-2 p-1"
style={{backgroundColor:'red' , color:'white'}}>

            <div className="flex-container">
              <div className='text-left w-100 m-1'>
                <h2 style={{fontSize:'25px'}}>Items</h2>
                <hr/>
                {order.orderItems.map(item=>{
                  return <div>
                    <p>{item.name} [{item.varient}] *
{item.quantity} = {item.price}</p>
                    </div>
                  })}
              </div>
              <div className='text-left w-100 m-1'>
                <h2 style={{fontSize:'25px'}}>Address</h2>
                <hr/>
                <p>Street : {order.shippingAddress.street}</p>
                <p>City : {order.shippingAddress.city}</p>
                <p>Country : {order.shippingAddress.country}</p>
                <p>Pincode : {order.shippingAddress.pincode}</p>
              </div>
              <div className='text-left w-100 m-1'>
                <h2 style={{fontSize:'25px'}}>Order Info</h2>
                <hr/>
                <p>Order Amount :
{order.orderAmount}</p>
                <p>Date : {order.createdAt.substring(0,10)}</p>
                <p>Transaction Id : {order.transactionId}</p>

```

```

        <p>Order Id : {order._id}</p>
      </div>

    </div>
  </div>

  }
}
</div>
</div>
)
}

```

- Registerscreen.js

```

import React, { useState, useEffect } from 'react';
import { render } from 'react-dom';
import { useDispatch, useSelector } from 'react-redux';
import { registerUser } from "../actions/userActions";
import Error from '../components/Error';
import Loading from '../components/Loading';
import Success from '../components/Success';
export default function Registerscreen() {
  const [name, setname] = useState("");
  const [email, setemail] = useState("");
  const [password, setpassword] = useState("");
  const [cpassword, setcpassword] = useState("");
  const registerstate = useSelector(state => state.registerUserReducer);
  const {error, loading, success} = registerstate;
  const dispatch = useDispatch()
  function register(){

    if(password!=cpassword)
    {
      alert("passwords not matched")
    }
    else{
      const user={
        name,
        email,
        password
      }
      console.log(user);
      dispatch(registerUser(user))
    }
  }

  return (
    <div>

```

```

        <div className="row justify-content-center mt-5">
            <div className="col-md-5 mt-5 text-left shadow-lg p-3 mb-5 bg-
white rounded">

                {loading && (<Loading/>)}
                {success && (<Success success='User Registered
Successfully' />)}
                {error && (<Error error='Email already registered' />)}

                <h2 className="text-center m-2" style={{ fontSize:
'35px' }}>Register</h2>

                <div>
                    <input required type="text" placeholder="name"
className="form-control" value={name} onChange={(e)=>{setname(e.target.value)}}
/>
                    <input required type="text" placeholder="email"
className="form-control" value={email}
onChange={(e)=>{setemail(e.target.value)}} />
                    <input required type="text" placeholder="password"
className="form-control" value={password}
onChange={(e)=>{setpassword(e.target.value)}} />
                    <input required type="text" placeholder="confirm
password" className="form-control" value={cpassword}
onChange={(e)=>{setcpassword(e.target.value)}} />
                    <button onClick={register} className="btn mt-3 mb-
3">REGISTER</button>
                    <br/>
                    <a style={{color:'black'}} href="/login">Click Here to
Login</a>

                </div>

            </div>
        </div>

        <div></div>

    </div>
)
}

```

- #App.css

```

.App {
    text-align: center;
}

.App-logo {

```

```

height: 40vmin;
pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

```

- App.js

```

import logo from './logo.svg';
import './App.css';
import bootstrap from 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap'
import {BrowserRouter, Route, Link, Switch} from 'react-router-dom';
import Navbar from './components/Navbar';
import Homescreen from './screens/Homescreen';
import Cartscreen from './screens/Cartscreen';
import Registerscreen from './screens/Registerscreen';
import Loginscreen from './screens/Loginscreen';
import Ordersscreen from './screens/Ordersscreen';
function App() {

```

```

return (
  <div className="App">
    <Navbar/>

    <BrowserRouter>

      <Route path="/" exact component={Homescreeen} />
      <Route path="/cart" exact component={Cartscreen} />
      <Route path="/register" exact component={Registerscreen} />
      <Route path="/login" exact component={Loginscreen} />
      <Route path="/orders" exact component={Ordersscreen} />

    </BrowserRouter>

  </div>
);
}

export default App;

```

- App.test.js

```

import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});

```

- #index.css

```

@import
url('https://fonts.googleapis.com/css2?family=Montserrat:wght@600&display=swap')
;

body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
'Oxygen',
  'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
  sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-weight: 'Montserrat', sans-serif !important;
}

```

```

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

.nav-link{
  color: black !important;
}

.navbar-brand{
  color:black !important;
}

h1{
  font-size: 20px !important;
}

.flex-container{
  display: flex !important;
}

.btn{
  background-color:red !important;
  color:white !important;
}

select{
  box-shadow: none !important;
  border-color: black !important;
}

.fa-plus{
  color:green;
  margin: 5px;
}

.fa-minus{
  color:red !important;
  margin: 5px;
}

.fa-trash{
  color:red;
}

input{
  margin-top: 10px !important;
  box-shadow: none !important;
}

```



```
}
```

- Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import {Provider} from 'react-redux';
import store from './store';
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

- Pizzasdata.js

```
const pizzas =[
  {
    name: "PEPPER BARBECUE CHICKEN",
    variants: ["small", "medium", "large"],
    prices: [
      {
        small: 200,
        medium: 350,
        large: 400,
      },
    ],
    category: "nonveg",
    image: "https://www.dominos.co.in/files/items/Pepper_Barbeque.jpg",
    description: "Pepper Barbecue Chicken I Cheese",
  },
  {
    name: "Non Veg Supreme",
    variants: ["small", "medium", "large"],
    prices: [
      {
        small: 200,
        medium: 350,
        large: 400,
```

```

    },
  ],
  category: "nonveg",
  image: "https://www.dominos.co.in/files/items/Non-Veg_Supreme.jpg",
  description: "Bite into supreme delight of Black Olives, Onions,
Grilled Mushrooms, Pepper BBQ Chicken, Per-Peri",
},
{
  name: "Golden Corn Pizza",
  variants: ["small", "medium", "large"],
  prices: [
    {
      small: 180,
      medium: 250,
      large: 300,
    },
  ],
},
],
category: "veg",
image: "https://www.crazymasalafood.com/wp-content/images/golden-1.jpg",
description: "Bite into supreme delight of Black Olives",
},
{
  name: "Jalapeno & Red Paprika Pizza",
  variants: ["small", "medium", "large"],
  prices: [
    {
      small: 200,
      medium: 300,
      large: 420,
    },
  ],
},
],
category: "veg",
image: "https://www.dominos.co.in/files/items/Non-Veg_Supreme.jpg",
description: "Bite into supreme delight of Black Olives",
},
{
  name: "Margherita",
  variants: ["small", "medium", "large"],
  prices: [
    {
      small: 200,
      medium: 350,
      large: 400,
    },
  ],
},
],
category: "nonveg",
image: "https://www.dominos.co.in/files/items/Non-Veg_Supreme.jpg",
description: "Bite into supreme delight of Black Olives, Onions,
Grilled Mushrooms, Pepper BBQ",

```

```

    },
    {
      name: "Double Cheese Margherita Pizza",
      variants: ["small", "medium", "large"],
      prices: [
        {
          small: 200,
          medium: 350,
          large: 400,
        },
      ],
      category: "veg",
      image: "https://www.crazymasalafood.com/wp-content/images/double-1.jpg",
      description: "Bite into supreme delight of Black Olives",
    },
  ],
};
export default pizzas

```

- reportWebvitals.js

```

const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) =>
    {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};
export default reportWebVitals;

```

- setupTests.js

```

// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';

```

- store.js

```

import {combineReducers} from 'redux'

```

```

import {createStore, applyMiddleware} from 'redux'

import thunk from 'redux-thunk'

import {composeWithDevTools} from 'redux-devtools-extension'
import {getAllPizzasReducer} from './reducers/pizzaReducers'
import { cartReducer } from './reducers/cartReducer'
import { registerUser } from './actions/userActions'
import { loginUserReducer, registerUserReducer } from './reducers/userReducer'
import { placeOrderReducer , getUserOrdersReducer } from
'./reducers/orderReducer'

const finalReducer = combineReducers({
  getAllPizzasReducer : getAllPizzasReducer,
  cartReducer : cartReducer,
  registerUserReducer : registerUserReducer,
  loginUserReducer : loginUserReducer,
  placeOrderReducer : placeOrderReducer,
  getUserOrdersReducer : getUserOrdersReducer
})

const cartItems = localStorage.getItem('cartItems') ?
JSON.parse(localStorage.getItem('cartItems')) : []

const currentUser = localStorage.getItem('currentUser') ?
JSON.parse(localStorage.getItem('currentUser')) : null

const initialState = {
  cartReducer: {
    cartItems: cartItems
  },
  loginUserReducer :{
    currentUser : currentUser
  }
}

const composeEnhancers = composeWithDevTools({})

const store = createStoreHook(finalReducer , initialState ,
composeEnhancers(applyMiddleware(thunk)))

export default store

```

- orderModel.js

```

const mongoose = require("mongoose");

const orderSchema= mongoose.Schema({
  name : {type: String , require},

```

```

    email : {type: String , require},
    userid : {type: String , require},
    orderItems : [],
    shippingAddress : {type:Object},
    orderAmount : {type:Number , require},
    isDelivered : {type: String , require , default: false},
    transactionId : {type: String , require}
  },{
    timestamps : true
  })

module.exports = mongoose.model('orders' , orderSchema)

```

- pizzaModel.js

```

const mongoose = require("mongoose");

const pizzaSchema = mongoose.Schema({

  name : {type: String , require},
  variants : [] ,
  prices : [],
  category : {type: String , require},
  image : {type: String , require},
  description : {type: String , require}

} , {
  timestamps:true,
})

const pizzaModel = mongoose.model('pizzas' , pizzaSchema)

module.exports = pizzaModel

```

- userModel.js

```

const mongoose = require("mongoose");

const userSchema = mongoose.Schema({
  name : {type: String , require},
  email : {type: String , require},
  password : {type: String , require},
  isAdmin : {type: Boolean , require , default: false},
} , {
  timestamps : true,
})

module.exports = mongoose.model('users' , userSchema)

```

- ordersRoute.js

```
const express = require("express");
const { resolvePath } = require("react-router-dom");
const router = express.Router();
const { v4: uuidv4 } = require("uuid");
const stripe =
require("stripe")("sk_test_51MhA3KSJhyrIFUV95QTck308L3YFQ2P05583KKyRYZXvDNEa9oz
YlV4mn9QxgpcAhDl7homJSpGdHAGWBU106sg000YYoqCqn6");
const Order = require('../models/orderModel');
router.post("/placeholder", async(req, res) => {

    const {token , subtotal , currentUser , cartItems} = req.body

    try {
        const customer = await stripe.customer.create ({
            email : token.email,
            source:token.id
        })

        const payment = await stripe.charges.create({
            amount:subtotal*100,
            currency:'inr',
            customer : customer.id,
            receipt_email : token.email
        }, {
            idempotencyKey : uuidv4()
        })

        if(payment)
        {
            const neworder = new Order({
                name : currentUser.name,
                email : currentUser.email,
                userid : currentUser._id,
                orderItems : cartItems,
                orderAmount : subtotal,
                shippingAddress : {
                    street : token.card.address_line1,
                    city : token.card.address_city,
                    country : token.card.address_country,
                    pincode : token.card.address_zip,
                },
                transactionId : payment.source.id
            })

            neworder.save()

            res.send('Order placed successfully')
```

```

    }
    else{
        res.send('Payment Failed')
    }

} catch (error) {
    return res.status(400).json({ message: 'Something went wrong' + error});
}

});

router.post("/getuserorders", async(req, res) => {
    const {userid} = req.body
    try {
        const orders = await Order.find({userid : userid}).sort({_id: -1})
        res.send(orders)

    } catch (error) {
        return res.status(400).json({ message: 'Something went wrong' });
    }
});

module.exports = router

```

- pizzasRoute.js

```

const express = require("express");
const router = express.Router();
const Pizza = require('../models/pizzaModel')

router.get("/getallpizzas", async(req, res) => {

    try {
        const pizzas = await Pizza.find({})
        res.send(pizzas)
    } catch (error) {
        return res.status(400).json({ message: error });
    }

});

module.exports = router;

```

- userRoute.js

```

const express = require("express");
const router = express.Router();

```

```

const User = require("../models/userModel")

router.post("/register", async(req, res) => {

  const {name , email , password} = req.body

  const newUser = new User({name , email , password})

  try {
    newUser.save()
    res.send('User Registered successfully')
  } catch (error) {
    return res.status(400).json({ message: error });
  }

});

module.exports = router

router.post("/login", async(req, res) => {

  const {email , password} = req.body

  try {

    const user = await User.find({email , password})

    if(user.length > 0)
    {
      const currentUser = {
        name : user[0].name,
        email : user[0].email,
        isAdmin : user[0].isAdmin,
        _id : user[0]._id
      }
      res.send(currentUser);
    }
    else{
      return res.status(400).json({ message: 'User Login Failed' });
    }
  } catch (error) {
    return res.status(400).json({ message: 'Something went wrong' });
  }

})

```


OUTPUT

SHEY PIZZA Login Cart 0

Login

LOGIN

[Click Here To Register](#)

SHEY PIZZA Login Cart 0

Register

REGISTER


[Click Here To Login](#)

SHEY PIZZA victor ▾ Cart 0

All ▾

FILTER

PEPPER BARBECUE CHICKEN



Varients

Quantity


small ▾

1 ▾

Price : 200 Rs/-

ADD TO CART

Non Veg Supreme Gold



Varients

Quantity


small ▾

1 ▾

Price : 200 Rs/-

ADD TO CART

Golden Corn Pizza Special



Varients

Quantity

small ▾

1 ▾

Price : 180 Rs/-

ADD TO CART

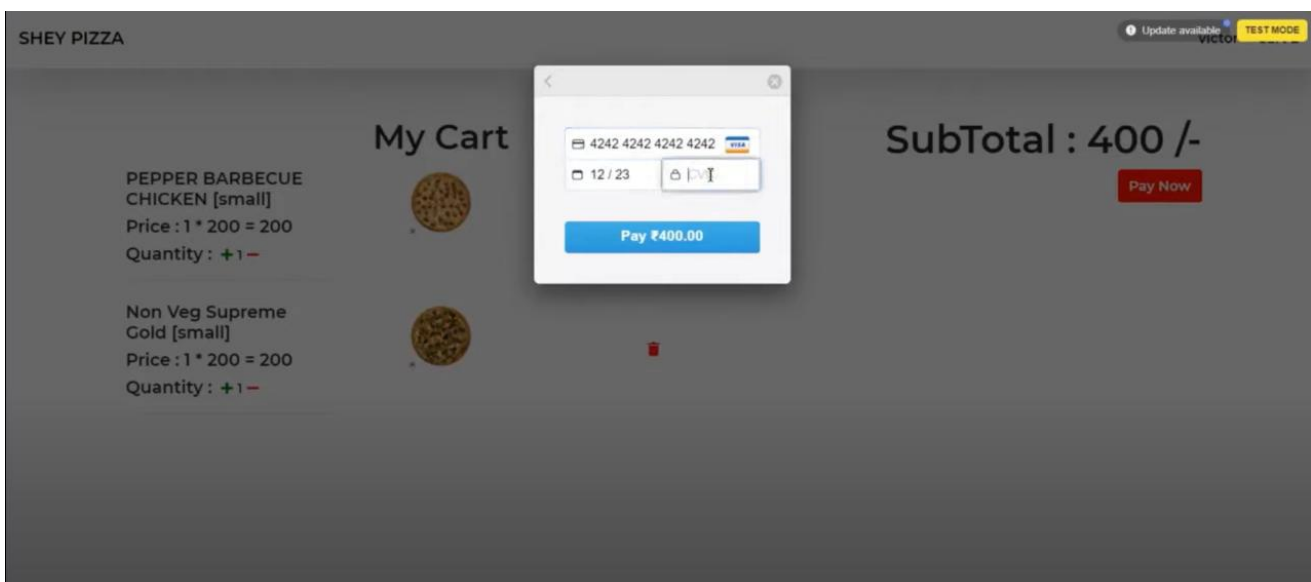
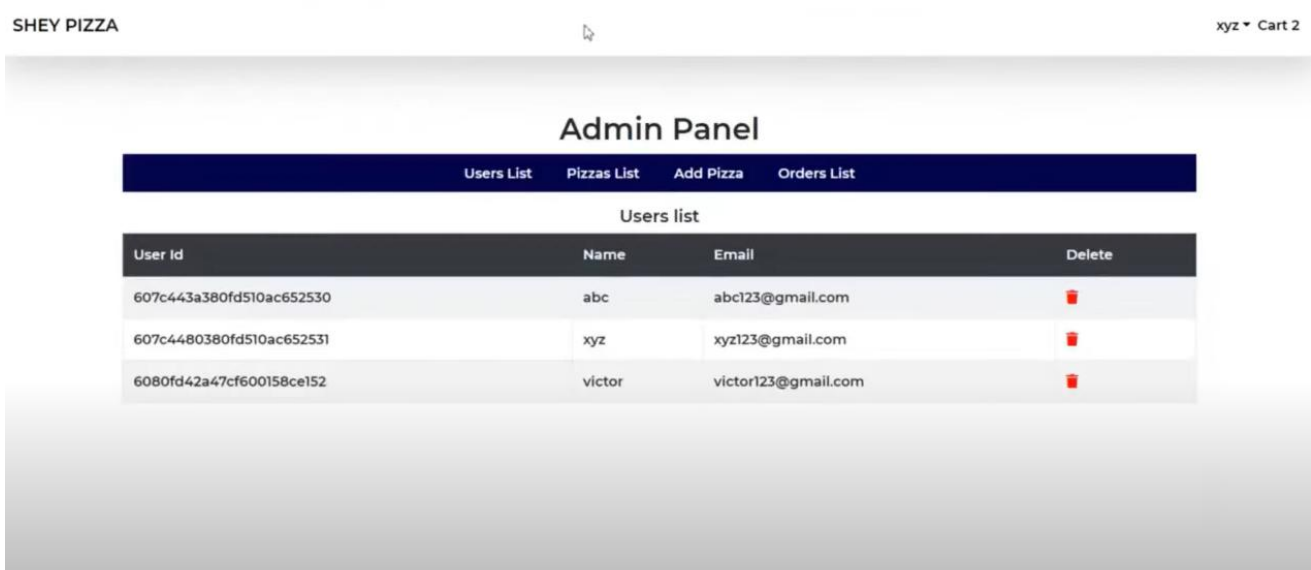


Figure 2 : Output of the Food-Delivery Website “Client Screen”



Admin Panel

[Users List](#) [Pizzas List](#) [Add Pizza](#) [Orders List](#)

Pizzas List

Name	Prices	Category	Actions
PEPPER BARBECUE CHICKEN	Small : 200 Medium : 350 Large : 400	nonveg	 
Non Veg Supreme Gold	Small : 200 Medium : 350 Large : 400	nonveg	 
Golden Corn Pizza Special	Small : 180 Medium : 250 Large :	veg	 
Jalapeno & Red Paprika Pizza	Small : 200 Medium : 300	veg	 

Admin Panel

[Users List](#) [Pizzas List](#) [Add Pizza](#) [Orders List](#)

Add Pizza

Admin Panel

[Users List](#) [Pizzas List](#) [Add Pizza](#) [Orders List](#)

Order Id	Email	User Id	Amount	Date	Status
607d834f177fee40d47dc277	abc123@gmail.com	607c443a380fd510ac652530	200	2021-04-19	Delivered
607d8c1b63acf81c0c05e41e	abc123@gmail.com	607c443a380fd510ac652530	580	2021-04-19	Delivered
607feca894f9f141148799d0	xyz123@gmail.com	607c4480380fd510ac652531	350	2021-04-21	<input type="button" value="Deliver"/>
6080fe39a47cf600158ce153	victor123@gmail.com	6080fd42a47cf600158ce152	400	2021-04-22	<input type="button" value="Deliver"/>

Figure 3 : Output of the Food-Delivery Website “Server Screen”

4. CONCLUSION AND RECOMMENDATIONS

****Conclusion :**

Food delivery websites have become a significant part of modern dining culture, offering convenience, variety, and efficiency to both users and restaurant partners. These platforms have evolved to meet the changing needs of consumers, particularly those with busy lifestyles. They provide a user-friendly experience, allowing customers to explore diverse culinary options, customize orders, and track deliveries in real time. For restaurant partners, food delivery websites offer increased visibility, additional revenue streams, and efficient order management.

However, the success of a food delivery website depends on various factors, including the quality of service, user experience, and effective marketing. Challenges such as competition, logistics, and food safety standards must be addressed to maintain and grow in this dynamic industry.

****Recommendations :**

Based on the analysis of food delivery websites and industry trends, here are some recommendations for operators and stakeholders:

1. **Focus on User Experience:** Continuously improve the user interface and experience to make it user-friendly, responsive, and visually appealing. Prioritize speed and ease of use to enhance customer satisfaction.
2. **Expand Restaurant Partnerships:** Attract a diverse range of restaurant partners, including local eateries and international cuisines, to offer users a wider selection of dining options.
3. **Efficient Order Processing:** Invest in efficient order processing and delivery coordination systems to minimize wait times and ensure timely deliveries.
4. **Security and Data Privacy:** Implement robust security measures to protect user data and payment information. Comply with data protection regulations to build trust among users.
5. **Loyalty Programs and Promotions:** Offer loyalty programs and promotions to incentivize repeat business and attract new users. Personalize offers based on user preferences.
6. **Quality Control:** Collaborate closely with restaurant partners to ensure food quality and safety standards are met consistently. Conduct regular inspections and quality checks.
7. **Marketing and Branding:** Develop a strong marketing strategy to promote the website and attract users and restaurant partners. Leverage social media, email marketing, and partnerships for increased visibility.
8. **Customer Support:** Provide efficient and responsive customer support channels to address user inquiries and resolve issues promptly.

In conclusion, food delivery websites have reshaped the way people dine, offering a world of culinary possibilities at their fingertips. By focusing on user satisfaction, efficient operations, and continuous innovation, operators can navigate the competitive landscape and deliver a superior dining experience to users while building strong partnerships with restaurants.

5. REFERENCES

1. **Official Websites:** Visit the official websites of popular food delivery platforms such as Uber Eats, DoorDash, Grubhub, Postmates, and others. These websites often provide information about their services, features, and how they operate.
2. **Industry Publications:** Explore industry publications, blogs, and news websites that cover developments and trends in the food delivery industry. Websites like TechCrunch, Forbes, and The Spoon often publish articles and reports on this topic.
3. **Academic Journals:** Look for academic research and studies related to online food delivery, e-commerce, and restaurant technology in academic journals. These sources may provide in-depth insights and data.
4. **Books:** Some books discuss the evolution of food delivery and its impact on the restaurant industry. Consider searching for relevant books on platforms like Amazon or in your local library.
5. **Market Research Reports:** Market research companies often publish reports on the online food delivery market. Reports from companies like Statista, MarketResearch.com, and Euromonitor International can provide valuable insights and statistics.
6. **Company Reports:** Publicly traded food delivery companies are required to publish financial and operational reports. These reports can offer insights into their performance and strategies.
7. **Government Reports:** Government agencies may conduct studies or publish reports related to the food delivery industry, especially in terms of regulations and safety standards.
8. **Blogs and Forums:** Explore blogs, forums, and discussion boards where industry professionals and users share their experiences and insights related to food delivery websites.

Note :- Remember to verify the credibility and relevance of the sources you consult, and always check for the most up-to-date information, as the food delivery industry is continually evolving.

- https://en.wikipedia.org/wiki/Online_food_ordering
- <https://www.ubereats.com/>
- <https://www.swiggy.com/>
- <https://www.foodpanda.com/>
- <https://www.zomato.com/india>

-----THANK U-----