

# **Major Project**

## **(23ONMCR-753)**

### **of the programme**

**Master of Computer Applications**

**Batch- July23**

**Fourth Semester**

**CENTRE FOR DISTANCE & ONLINE EDUCATION**  
**CHANDIGARH UNIVERSITY**

Submitted By: **Akash Pathania**

Enrollment No: **O23MCA110008**

# SYNOPSIS

## 1. Title of the Project

E- Restaurant Application using firebase and Stripe Integration

## 2. Objective

The primary objective of this project, titled "**E-Restaurant: An Online Restaurant E-Commerce Platform**", is to design and develop a full-stack web application that streamlines the process of ordering food online for customers while providing restaurant owners with tools to manage their operations efficiently. The system aims to enhance user convenience, reduce manual workload, and promote digital transformation in the food service industry. This project, available at [GitHub Repository](#), has been implemented using the **MERN stack (MongoDB, Express.js, React.js, Node.js)**, and includes features such as user authentication, menu browsing, cart management, and order placement. By offering a robust, scalable, and responsive solution, the objective is to demonstrate the practical application of modern web technologies in building a real-world e-commerce system tailored specifically for restaurants.

## 3. Resources Required

Hardware Component	Specification
Computer/Laptop	Minimum Intel i5 Processor, 8 GB RAM, 256 GB SSD
Internet Connection	Stable connection for installing packages and Git access
Server (Optional)	For deployment (e.g., cloud VM or VPS)

Software / Tool	Purpose
<b>Node.js</b>	JavaScript runtime environment for backend development
<b>npm / yarn</b>	Package managers for managing project dependencies
<b>React.js</b>	Frontend library for building responsive UI
<b>Express.js</b>	Web framework for creating APIs in Node.js
<b>MongoDB</b>	NoSQL database for storing user, product, and order data
<b>Postman</b>	API testing and validation tool
<b>Visual Studio Code</b>	Source code editor with extensions support
<b>Git</b>	Version control system for tracking code changes
<b>GitHub</b>	Repository hosting and project collaboration
<b>Browser (Chrome/Firefox)</b>	For UI testing and application preview

## 4. System Features

- User registration and secure login (JWT-based authentication)
- Product (food item) browsing with dynamic rendering
- Shopping cart with quantity adjustment and item removal
- Order checkout and order history tracking
- Admin-side product and order management
- API integration between frontend and backend
- Responsive UI for desktop and mobile views

## 5. Methodology

The project follows the **Software Development Life Cycle (SDLC)** model, starting with requirement analysis, followed by system design, coding, testing, and deployment. Each component was developed iteratively to ensure modularity and maintainability.

## 6. Expected Outcome

The proposed system is expected to improve restaurant operations by automating the order and delivery workflow, enhancing customer satisfaction, and minimizing manual errors. It also offers a scalable architecture that can be extended with features like online payment integration, real-time order tracking, and analytics dashboards.

## 7. Conclusion

The E-Restaurant project serves as a real-world demonstration of full-stack web development and its ability to solve modern business problems. It is a scalable and robust solution that can be adapted by small to medium-sized restaurants aiming to go digital and serve their customers more efficiently.

# Table Of Contents

## Contents

<b>Title .....</b>	<b>6</b>
<b>Declaration.....</b>	<b>8</b>
<b>Acknowledgement.....</b>	<b>9</b>
<b>Abstract .....</b>	<b>10</b>
<b>Introduction .....</b>	<b>11</b>
<b>SDLC of the project .....</b>	<b>12</b>
<b>Design.....</b>	<b>13</b>
Frontend Design .....	13
Backend Design.....	13
Database Design .....	13
Home View-.....	14
<b>Coding &amp; Implementation (<a href="https://github.com/akashp-007/Restaurant-E-Commerce-Website-main">https://github.com/akashp-007/Restaurant-E-Commerce-Website-main</a>).....</b>	<b>21</b>
Frontend (client/) .....	25
Key Features: .....	25
Notable Components: .....	25
Backend (server/).....	37
Key Features: .....	37
Notable Files and Directories: .....	37
<b>Testing .....</b>	<b>43</b>
<b>Application.....</b>	<b>44</b>
<b>Conclusion.....</b>	<b>45</b>
<b>Bibliography(APA Style).....</b>	<b>47</b>

# Title

## **E- Restaurant Application using firebase and Stripe Integration**

**Major Project Report Submitted in partial fulfillment of the requirement for the  
award of the degree of MASTER OF COMPUTER APPLICATIONS (MCA)**

Submitted By: **Akash Pathania**

Enrollment No: **O23MCA110008**

Centre for Distance and Online Education

**Chandigarh University**

May 2025

# Certificate

This is to certify that the project entitled “**E- Restaurant Application using firebase and Stripe Integration**” is a bona fide work carried out by **Akash Pathania** (Enrollment No: **O23MCA110008**) in partial fulfillment for the award of the degree of **Master of Computer Applications**.

To the best of my knowledge, the work reported herein does not form part of any other project submitted for the award of any degree or diploma in this or any other institution. The project has been carried out as per the guidelines and regulations laid down by **Chandigarh University**.

# Declaration

I, **Akash Pathania**, a student of **Master of Computer Applications (MCA)**, Batch **July 2023-2025**, hereby declare that the project titled:

**“E-Restaurant: An Online Restaurant E-Commerce Platform”**

submitted in partial fulfillment of the requirements for the subject code **23ONMCR-753 (Major Project)**, is my original work and has not been submitted earlier, either partly or wholly, for any other degree or diploma in this or any other institution.

The project has been developed under the guidance and supervision of faculty members at **Chandigarh University**, and all the information and data presented in this report is authentic and based on my independent work and research. Any material or content taken from external sources has been duly acknowledged and referenced.



# Acknowledgement

I would like to express my sincere gratitude to all those who supported and guided me throughout the development of this project, *“E-Restaurant: An Online Restaurant E-Commerce Platform.”*

First and foremost, I extend my heartfelt thanks to my project guide and faculty members at **Chandigarh University** for their invaluable guidance, encouragement, and constructive feedback throughout the course of this work. Their insights helped shape the project into a successful and meaningful academic endeavor.

I also wish to thank my classmates and friends for their constant support and for providing valuable suggestions during the development process. Their critical observations helped improve the functionality and user experience of the system.

Lastly, I am grateful to my family for their continuous motivation, patience, and understanding during the entire period of this project work. Without their support, this accomplishment would not have been possible.

**Akash Pathania**

# Abstract

The project titled “*E-Restaurant: An Online Restaurant E-Commerce Platform*” is a web-based application developed using the **MERN stack** (MongoDB, Express.js, React.js, and Node.js). It aims to provide a complete digital solution for online food ordering and restaurant management. The system enables users to browse food items, add products to a cart, and place orders securely, while restaurant administrators can manage menu items, monitor customer orders, and update order statuses.

With the shift towards online services in the food industry, this project addresses the need for fast, convenient, and contactless food ordering systems. The platform offers a user-friendly interface, secure authentication, responsive design, and efficient backend communication. It adheres to the Software Development Life Cycle (SDLC), ensuring a structured approach to development including planning, design, implementation, testing, and deployment.

This project demonstrates the integration of modern web technologies in solving real-world problems, and it serves as a scalable foundation for building more advanced restaurant management systems with features such as payment gateways, real-time order tracking, and customer feedback.

# Introduction

The growing demand for digital convenience has significantly transformed the food industry, with online food ordering becoming a standard service. The **E-Restaurant: An Online Restaurant E-Commerce Platform** is developed to meet this demand by providing a seamless interface for both customers and restaurant administrators.

This web-based system allows users to browse the menu, add items to their cart, place orders, and make payments online. It also enables restaurant staff to manage menu items, process orders, and update statuses in real time. The platform aims to reduce manual effort, enhance service accuracy, and improve customer satisfaction.

Built using the **MERN stack** (MongoDB, Express.js, React.js, Node.js), the application offers a modern, scalable, and responsive experience. React handles the user interface, while Node.js and Express manage server-side operations, and MongoDB stores user, order, and product data.

Overall, this project demonstrates the practical application of full-stack development to solve real-world problems in the restaurant domain, helping businesses grow their digital presence and operate more efficiently.

# SDLC of the project

The Software Development Life Cycle (SDLC) followed for this project includes:

1. **Requirement Analysis:** Identifying the needs of both restaurant owners and customers.
2. **System Design:** Architecting the system's frontend and backend components.
3. **Implementation:** Developing the application using chosen technologies.
4. **Testing:** Ensuring the application functions as intended and is free of bugs.
5. **Deployment:** Hosting the application for public access.
6. **Maintenance:** Ongoing updates and improvements based on user feedback.

# Design

## Frontend Design

- **Technology Used:** React.js
- **Features:**
  - Responsive design for various devices.
  - User-friendly interface for browsing menus and placing orders.
  - Authentication pages for login and registration.

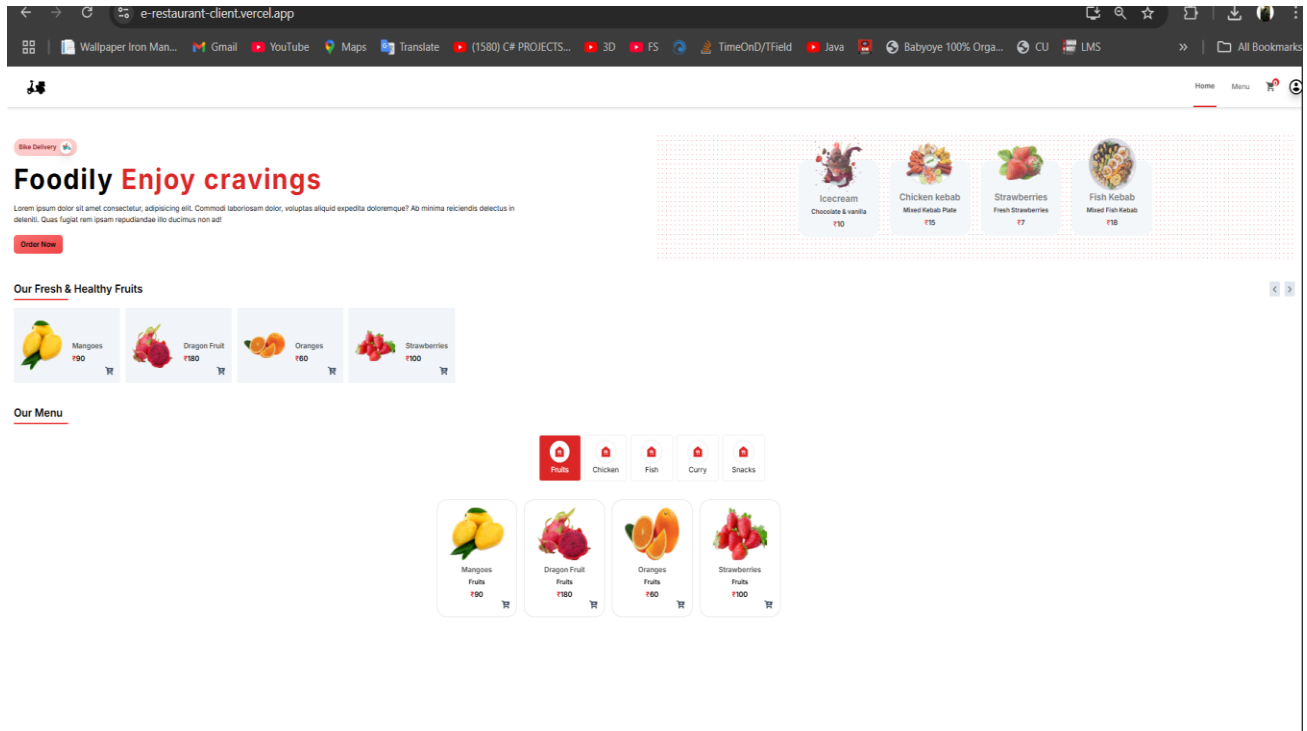
## Backend Design

- **Technology Used:** Node.js with Express.js
- **Features:**
  - RESTful APIs for handling data transactions.
  - Integration with MongoDB for data storage.
  - Secure authentication mechanisms.

## Database Design

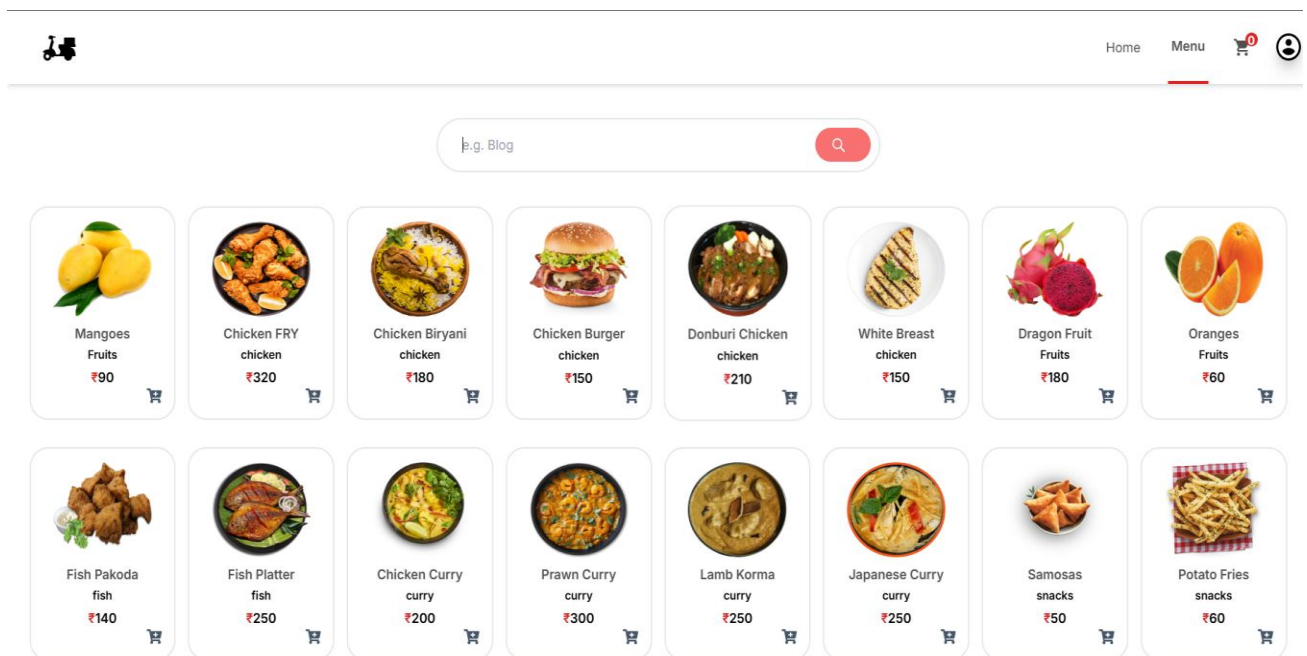
- **Technology Used:** MongoDB
- **Collections:**
  - **Users:** Stores user information and credentials.
  - **Products:** Contains details of menu items.
  - **Orders:** Records customer orders and statuses.

## Home View-



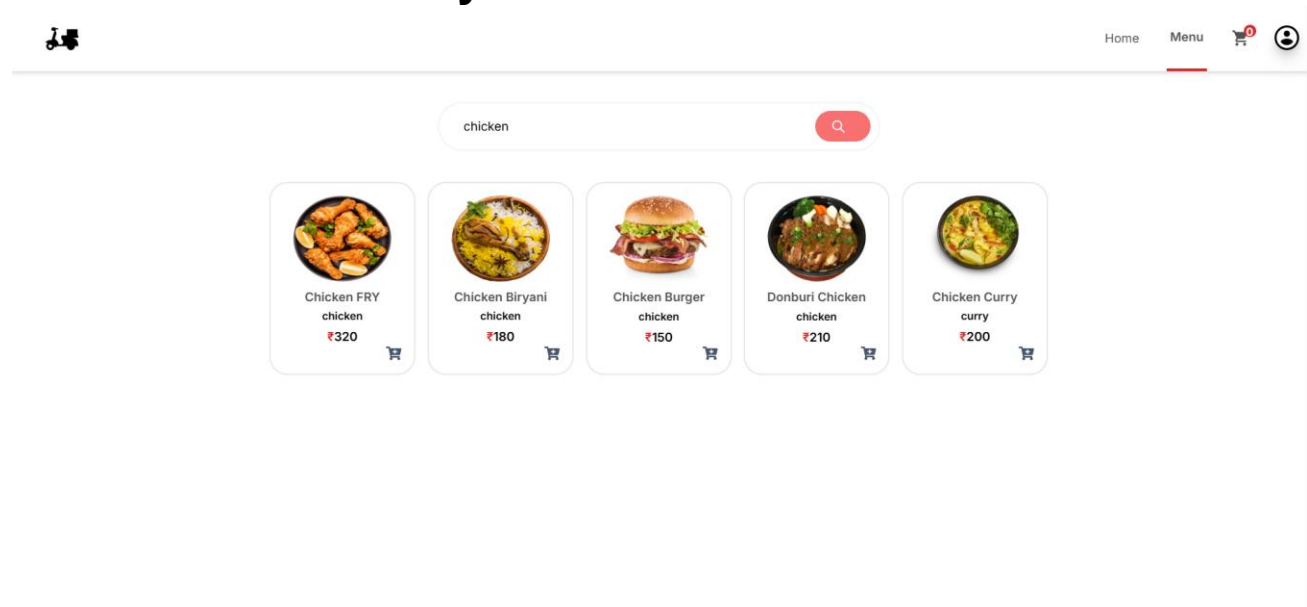
Home View 1

## Menu View-



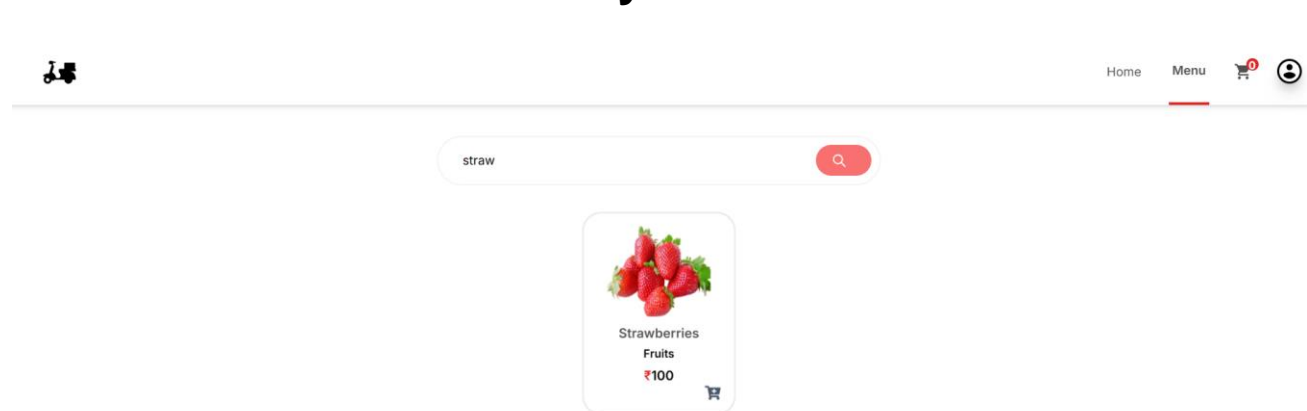
Menu View 1

## Search Functionality-



*Search Bar 1*

## Partial Search Functionality



*Partial Search 1*



## Cart-



Home Menu  

### Your Shopping Cart

Product

	Chicken Burger ₹150 - 1 +	Total : ₹150
	Patties ₹30 - 1 +	Total : ₹30

Price Summary	
Cart Total	₹180
Delivery Charge	₹0
<b>Total</b>	<b>₹180</b>
<a href="#">Payment</a>	

*Cart 1*

## Not Login Notification-





your are not login!!

Home Menu  

### Your Shopping Cart

Product

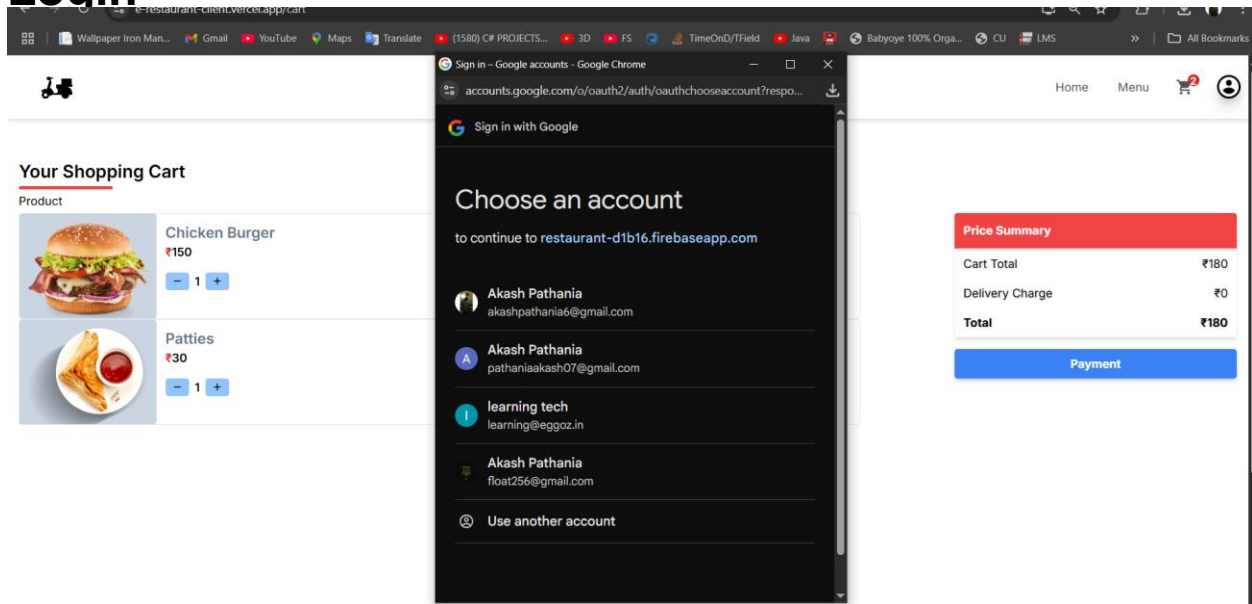
	Chicken Burger ₹150 - 1 +	Total : ₹150
	Patties ₹30 - 1 +	Total : ₹30

Price Summary	
Cart Total	₹180
Delivery Charge	₹0
<b>Total</b>	<b>₹180</b>
<a href="#">Payment</a>	

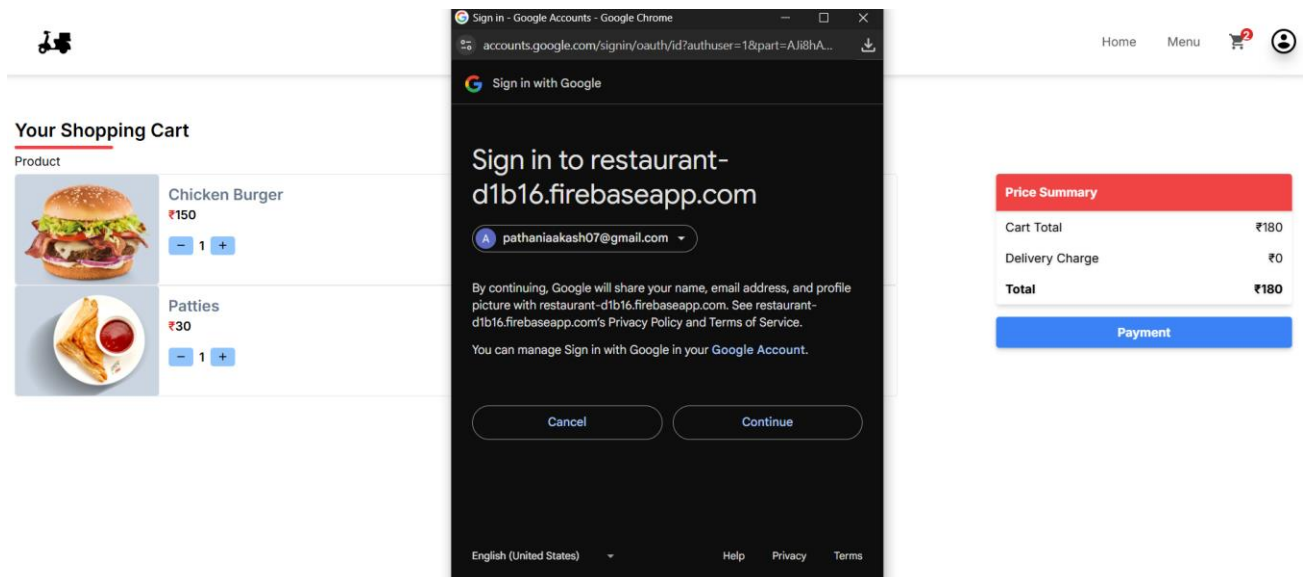
*Not Login 1*



## Login-



Login 1



## Logged In-

Akash Pathania
Home
Menu

### Your Shopping Cart

Product

	<b>Chicken Burger</b> ₹150 - 1 +	Total : ₹150
	<b>Patties</b> ₹30 - 1 +	Total : ₹30

### Price Summary

Cart Total	₹180
Delivery Charge	₹0
<b>Total</b>	<b>₹180</b>

[Payment](#)

*Login 2 Logged In User*

## Stripe Integration (Payment Details)-

← TEST MODE

Pay

**₹180.00**

	Chicken Burger Qty 1	₹150.00
	Patties Qty 1	₹30.00

[Pay with link](#)

Or

Email  
email@example.com

Card information  
1234 1234 1234 1234  
MM / YY CVC

Cardholder name  
Full name on card

Country or region  
India

☐ Securely save my information for 1-click checkout  
Pay faster on this site and everywhere Link is accepted.

[Pay](#)

Powered by stripe

*Payment Details 1*

## Incorrect Details Notification-

←

TEST MODE

Pay

₹180.00

Chicken Burger

Qty 1 ▾

₹150.00

Patties

Qty 1 ▾

₹30.00

Pay with link

Or

Email

akashpathania6@gmail.com

Card information

4242 4242 4242 1212

12 / 29

123

Your card number is invalid.

Cardholder name

Akash Pathania

Country or region

India ▾

☐ Securely save my information for 1-click checkout  
 Pay faster on this site and everywhere Link is accepted.

Pay

Incorrect Details 1

## Entered Details-

←

TEST MODE

Pay

₹180.00

Chicken Burger

Qty 1 ▾

₹150.00

Patties

Qty 1 ▾

₹30.00

Pay with link

Or

Email

akashpathania6@gmail.com

Card information

4242 4242 4242 4242

12 / 29

123

Cardholder name

Akash Pathania

Country or region

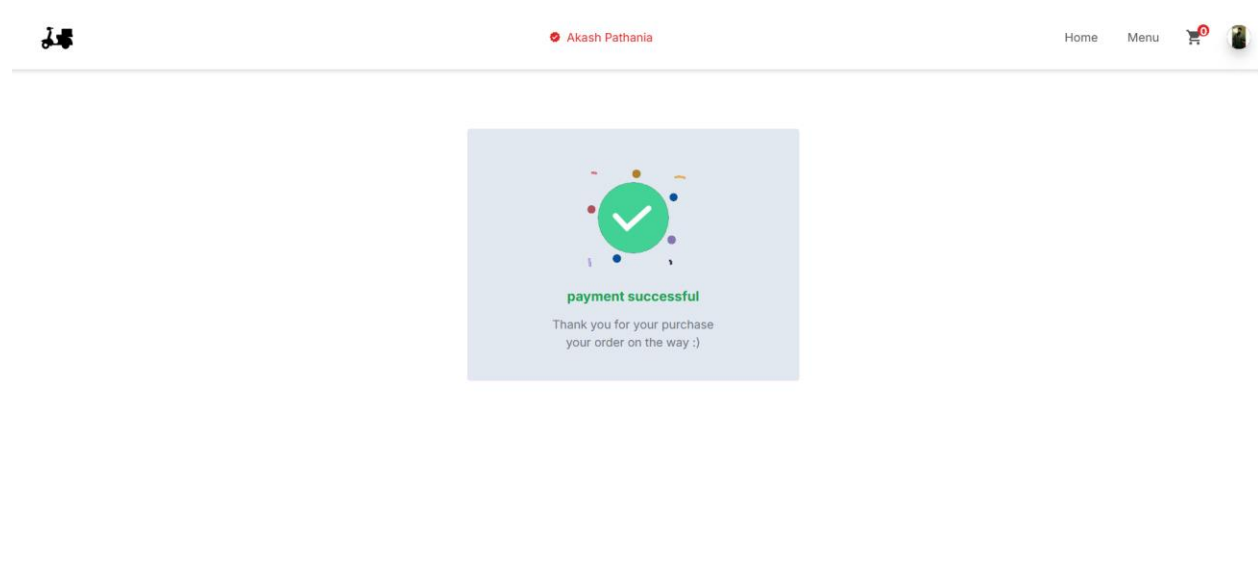
India ▾

☐ Securely save my information for 1-click checkout  
 Pay faster on this site and everywhere Link is accepted.

Pay

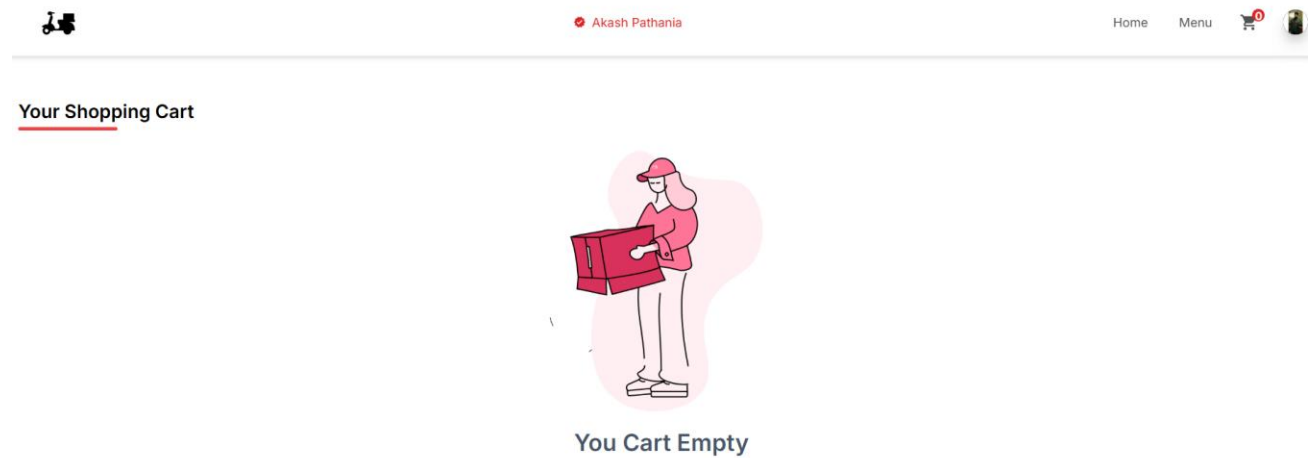
Details Entered 1

## Success-



*Success 1*

## Empty Cart Notification-



*Empty Cart 1*

# Coding & Implementation

## **HTML: -**

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets and scripting languages such as JavaScript. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as `<img/>` and `<input />` directly introduce content into the page. Other tags such as `<p>` surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. The inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997. A form of HTML, known as HTML5, is used to display video and audio, primarily using the `<canvas>` element, in collaboration with javascript.

## **CSS: -**

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML (including) XML dialects such as SVG, MathML or XHTML). CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of content and presentation, including layout, colors, and fonts. This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate

.css file, which reduces complexity and repetition in the structural content; and enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device. The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable. The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

In addition to HTML, other markup languages support the use of CSS including XHTML, plain XML, SVG, and XUL.

## **JAVASCRIPT: -**

JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. As of 2022, 98% of websites use JavaScript on the client side for webpage behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on user's devices.

JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard. 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 programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O.

JavaScript engines were originally used only in web browsers, but are now core components of some servers and a variety of applications. The most popular runtime system for this usage is Node.js.

Although Java and JavaScript are similar in name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design.

## **REACTJS: -**

React (also known as React.js or ReactJS) is a free and open-source front- end JavaScript library for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

## **NODEJS: -**

Node.js is an open-source server environment. Node.js is cross-platform and runs on Windows, Linux, Unix, and macOS. Node.js is a back-end JavaScript runtime environment. Node.js runs on the V8 JavaScript Engine and executes JavaScript code outside a web browser.

Node.js lets developers use JavaScript to write command line tools and for server-side scripting. The functionality of running scripts server-side produces dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

The Node.js distributed development project was previously governed by the Node.js Foundation, and has now merged with the JS Foundation to form the OpenJS Foundation. OpenJS Foundation is facilitated by the Linux Foundation's Collaborative Projects program.

## **SOCKET IO :-**

Socket.IO is an event-driven library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It consists of two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API.

Socket.IO primarily uses the WebSocket protocol with polling as a fallback option, while providing the same interface. Although it can be used simply as a wrapper for WebSockets, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O. It can be installed with the Node Package Manager (NPM).

Socket.IO provides the ability to implement real-time analytics, binary streaming, instant messaging, and document collaboration. Notable users include Microsoft Office, Yammer, and Zendesk. Socket.IO handles the connection transparently and will automatically upgrade to WebSocket if possible. This means that the developer does not need to know how to use the WebSocket protocol in order to use Socket.IO. Socket.IO is not a WebSocket library with fallback options to other real-time protocols. It is a custom real-time transport protocol implementation on top of other real-time protocols. A Socket.IO implementing server cannot connect to a non-Socket.IO WebSocket client. A Socket.IO implementing client cannot talk to a non-Socket.IO WebSocket or Long Polling Comet server. Socket.IO requires using the Socket.IO libraries on both client and server side.

As of version 2.0, Socket.IO makes use of WebSockets as the underlying WebSocket library.

## **EXPRESSJS : -**

Express.js, or simply Express, is a back end web application framework for building RESTful APIs with Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

The original author, TJ Holowaychuk, described it as a Sinatra-inspired server, meaning that it is relatively minimal with many features available as plugins. Express is the back-end component of popular development stacks like the MEAN, MERN or MEVN stack, together with the MongoDB database software and a JavaScript front-end framework or library. Express.js was founded by TJ Holowaychuk. The first release, according to Express.js's GitHub repository, was on 22 May 2010.



In June 2014, rights to manage the project were acquired by StrongLoop. StrongLoop was acquired by IBM in September 2015; in January 2016, IBM announced that it would place Express.js under the stewardship of the Node.js Foundation incubator.

## Frontend (client/)

The frontend is built using **React.js**, offering a responsive and interactive user interface.

- **Setup:** Initialized using Create React App.
- **Components:**
  - **Navbar:** Navigation links and user authentication status.
  - **ProductList:** Displays available menu items.
  - **Cart:** Shows selected items and total price.

**Checkout:** Handles order placement and payment.

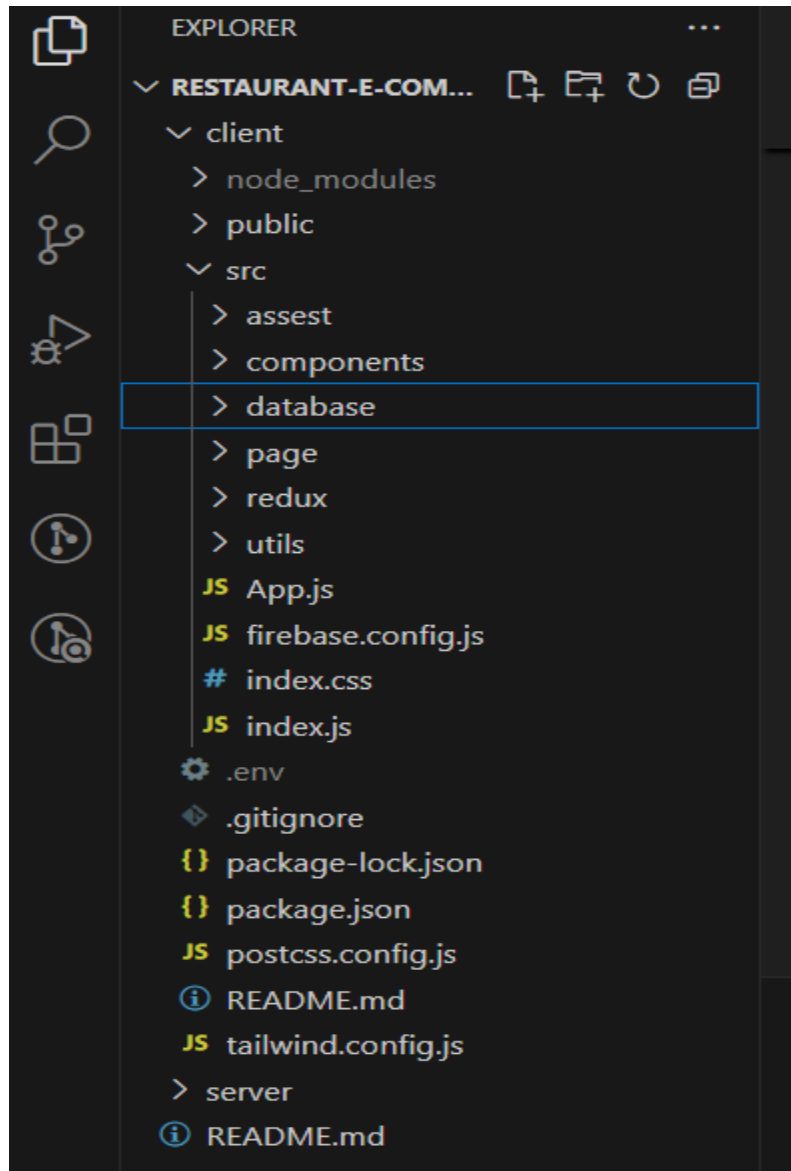
### Key Features:

- **User Authentication:** Allows users to register and log in to their accounts.
- **Product Browsing:** Displays a list of available menu items fetched from the backend.
- **Shopping Cart:** Enables users to add items to their cart and manage quantities.
- **Order Placement:** Facilitates the checkout process, allowing users to place orders.

### Notable Components:

- **App.js:** The root component that sets up routing and renders other components.
- **components/:** Contains reusable UI components like Navbar, ProductList, Cart, and Checkout.
- **pages/:** Includes page-level components such as Home, Login, Register, and OrderConfirmation.

## Code Structure-



## Home-

```
import React from "react";
import { useSelector } from "react-redux";
import DeliveryImage from "../assest/img/delivery.png";
import HeaderProduct from "../components/HeaderProduct";
import { headerData } from "../database/headerData";
import { Link } from "react-router-dom";
const Home = () => {
  const notHaveData = new Array(7).fill(null)
```

```
// console.log(notHaveData)

const productItem = useSelector((state) => state.productItem.productItem);
const productLoading = useSelector(
  (state) => state.productItem.productLoading
);

// console.log(productItem)
return (
  <div className="grid grid-cols-1 md:grid-cols-2 gap-2 overflow-hidden p-2 md:p-4">
    <div className="flex-1 flex flex-col items-start justify-center gap-5 py-3">
      <div className="flex items-center gap-2 justify-center bg-red-200 py-1 px-3 rounded-full">
        <p className="text-sm font-semibold text-red-800">Bike Delivery</p>
        <div className="w-7 h-7 bg-white rounded-full overflow-hidden drop-shadow-xl">
          <img
            src={DeliveryImage}
            className="w-full h-full object-contain"
            alt="Delivery"
          />
        </div>
      </div>
    </div>

    <p className="text-4xl font-bold tracking-wider text-headingColor md:text-5xl lg:text-6xl">
      Foodily
      <span className="text-red-600 text-4xl md:text-5xl lg:text-6xl">
        { " " }
        Enjoy cravings
      </span>
    </p>

    <p className="text-base text-color text-center md:text-left md:w-4/5">
      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Comodi
      laboriosam dolor, voluptas aliquid expedita doloremque? Ab minima
      reiciendis delectus in deleniti. Quas fugiat rem ipsam repudiandae
      illo ducimus non ad!
    </p>

    <Link to={"/menu"}
      type="button"
      className="bg-gradient-to-br from-red-400 to-red-500 w-full md:w-auto px-4 py-2 rounded-lg
      hover:shadow-lg transition-all ease-in-out font-semibold duration-300 flex justify-center md:justify-
      self-start hover:scale-105 "
    >
      Order Now
    </Link>
  </div>
)
```

```

</div>
<div className="p-10 md:pt-10 flex-1 designHomeRight h-full px-2 ">
  <div className="flex flex-wrap justify-center items-center gap-1 md:gap-6">
    {headerData &&
      headerData.map((el) => {
        return (
          <HeaderProduct
            key={el.id}
            id={el.id}
            name={el.name}
            img={el.img}
            decs={el.decs}
            price={el.price}
          />
        );
      })}
  </div>
</div>
</div>
);
};

export default Home;

```

## Menu-

```

import React from "react";
import { MdOutlineStarPurple500, MdOutlineStarHalf } from "react-icons/md";
import { FaCartPlus } from "react-icons/fa";
import Menu from "../components/Menu";
import { useNavigate, useParams } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { setCartProduct } from "../redux/cartSlice";

const MenuPage = () => {
  const productItem = useSelector(state => state.productItem.productItem)
  const params = useParams()
  const navigate = useNavigate()

  const data = productItem.filter(product => product.id === params.productId ,[])[0]
  // console.log(data)

  const dispatch = useDispatch()

```

```
const cartProduct = useSelector(state => state.cartProduct)

const handleCartProduct = (e) => {
  e.stopPropagation()
  e.preventDefault()

  dispatch(setCartProduct(data))
}

const handleBuyProduct = (e) => {
  handleCartProduct(e)
  navigate("/cart")
}

return (
  <div className="h-full">
    <div className="w-full p-4 max-w-3xl m-auto h-auto flex flex-col sm:flex-row">
      <div className="group w-full md:min-w-350 md:w-80 md:p-9 bg-slate-100 cursor-pointer">
        <img className="w-full rounded group-hover:scale-125 transition-all duration-300 "
src={data.imgURL} />
      </div>
      <div className="h-full sm:px-6 py-4">
        <h1 className="text-slate-600 font-bold capitalize text-xl md:text-2xl">
          {data.title}
        </h1>
        <div>
          <div className="text-red-600 flex items-center my-2">
            <MdOutlineStarPurple500 />
            <MdOutlineStarPurple500 />
            <MdOutlineStarPurple500 />
            <MdOutlineStarPurple500 />
            <MdOutlineStarHalf />
            <span className="text-slate-600 px-1">8.4</span>
            <span className="bg-slate-600 p-1 rounded-full"></span>
            <span className="text-slate-600 px-1">{data.category}</span>
          </div>
        </div>

        <p className="text-xl font-semibold text-headingColor">
          <span className="text-red-600">₹</span>
          {data.price}
        </p>
      </div>
    </div>
  </div>
)
```

```

    <div className="flex gap-4 w-full my-2">
      <button className="flex justify-center items-center py-1 px-5 border-2 border-solid w-
full min-w-[100px] max-w-[250px] whitespace-nowrap bg-red-600 text-white hover:bg-white
hover:text-black font-semibold rounded" onClick={handleBuyProduct}>Buy</button>
      <button className="flex justify-center items-center py-1 px-5 border-2 border-solid w-full
min-w-[100px] max-w-[200px] hover:bg-red-600 hover:text-white font-semibold rounded"
onClick={handleCartProduct}>
        <span className="whitespace-nowrap">Add to Cart</span>
        <FaCartPlus className="text-xl text-slate-600 hover:text-red-600 cursor-pointer self-end
mx-2" />
      </button>
    </div>

    <div>
      <h4 className="font-semibold">Descriptions : </h4>
      <span>{data.desc}</span>
    </div>
  </div>
  <div>
    <Menu heading={"Related Products"} filterbyProps={data.category} />
  </div>
);
};

export default MenuPage;

```

## Menu Static-

```

import React, { useEffect, useLayoutEffect, useState } from "react";
import { MdOutlineStarPurple500, MdOutlineStarHalf } from "react-icons/md";
import { FaCartPlus } from "react-icons/fa";
import Menu from "../components/Menu";
import { useNavigate, useParams } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { setCartProduct } from "../redux/cartSlice";
import { getAllProductItems } from "../utils/firebaseFunctions";
import { setProductItem } from "../redux/productSlice";
import RenderFilter from "../components/renderFilter";
import Loading from "../utils/Loading";

const MenuStatic = () => {
  const dispatch = useDispatch();
  const productItem = useSelector((state) => state.productItem.productItem);

```

```
const productLoading = useSelector(state => state.productItem.productLoading)
useEffect(() => {
  (async () => {
    try {
      const data = await getAllProductItems();
      // console.log("data - ", data);
      setRenderProducts(data)
      dispatch(setProductItem(data));
    } catch (error) {
      console.log(error);
    }
  })();
}, []);

// const params = useParams()
const navigate = useNavigate();

const data = productItem.filter(
  (product) => product.id == "1719488732134",
  []
)[0];
// console.log(productItem);

const cartProduct = useSelector((state) => state.cartProduct);

const [inputSearch, setInputSearch] = useState("");
const [renderProducts, setRenderProducts] = useState([])
// console.log(renderProducts)
// setRenderProducts(productItem)
// console.log(inputSearch)
const methodForSearch = ()=>{
  if(inputSearch.length > 0){
    const filteredProducts= renderProducts.filter(product
=>product.title.toLowerCase().includes(inputSearch.toLowerCase()));
    setRenderProducts(filteredProducts)
  }
  else{
    setRenderProducts(productItem)
  }
}

return (
```

```

<div className="h-full">
  <div class="relative w-full max-w-xl mx-auto bg-white rounded-full">
    <input
      placeholder="e.g. Blog"
      class="rounded-full w-full h-16 bg-transparent py-2 pl-8 pr-32 outline-none border-2 border-gray-100 hover:outline-none focus:ring-red-100 focus:border-slate-200"
      type="text"
      name="query"
      id="query"
      onChange={(e)=>setInputSearch(e.target.value)}
    />
    <button
      type="submit"
      class="absolute inline-flex items-center h-10 px-2 py-2 text-sm text-white transition duration-150 ease-in-out rounded-full outline-none right-3 top-3 bg-red-400 sm:px-6 sm:text-base sm:font-medium hover:bg-red-500"
      onClick={()=>methodForSearch()}
    >
      <svg
        class="-ml-0.5 sm:-ml-1 mr-2 w-4 h-4 sm:h-5 sm:w-5"
        xmlns="http://www.w3.org/2000/svg"
        fill="none"
        viewBox="0 0 24 24"
        stroke="currentColor"
      >
        <path
          stroke-linecap="round"
          stroke-linejoin="round"
          stroke-width="2"
          d="M21 21l-6-6m2-5a7 7 0 11-14 0 7 7 0 0114 0z"
        ></path>
      </svg>
    </button>
  </div>
  <div className="my-6 flex flex-wrap gap-4 max-w-screen m-auto justify-center">
    {productLoading ?
      (
        <div className="bg-slate-100 w-full p-16 flex justify-center items-end">
          <Loading/>
        </div>
      )
      : (
        // filterData.map((el) => (
        //   <RenderFilter
        //     key={el.id+"menu"}

```



```

    // id={el.id}
    // name={el.title}
    // img={el.imgURL}
    // decs={el.category}
    // price={el.price}

    // />
    // ))

    renderProducts.map((el) => {
      return <RenderFilter
        key={el.id + "menu"}
        id={el.id}
        name={el.title}
        img={el.imgURL}
        decs={el.category}
        price={el.price}
      />;
    })
  })
</div>

  { /* <Menu filterbyProps="all" /> */ }
</div>
);
};

export default MenuStatic;

```

## Cart-

```

import React from "react";
import { useSelector } from "react-redux";
import EmptyCart from "../components/EmptyCart";
import { MdOutlineAdd, MdOutlineRemove } from "react-icons/md";
import CartAddedItem from "../components/CartAddedItem";
import { Link } from "react-router-dom";
import PaymentSummary from "../components/PaymentSummary";
// import {loadStripe} from '@stripe/stripe-js';

import { Elements } from "@stripe/react-stripe-js";
import { loadStripe } from "@stripe/stripe-js";
import { PaymentElement } from "@stripe/react-stripe-js";
import CheckoutForm from "../components/CheckoutForm";

```

```
import toast from 'react-hot-toast';

const Cart = () => {
  const user = useSelector((state) => state.user);
  console.log(user.email)

  const cartProduct = useSelector((state) => state.cartProduct);
  // console.log(cartProduct);

  const cartTotal = cartProduct.cartProductItem.reduce(
    (acc, curr) => acc + curr.total,
    0
  );
  const deliveryCharge = 0;
  const Total = cartTotal + deliveryCharge;

  /***** */

  const stripeURL = process.env.REACT_APP_BACKEND_URL+'/create-checkout-session'
  const handlePayment = async(e)=>{
    e.preventDefault()
    if(user.email){
      console.log("fetc")
      const stripePromise = await loadStripe(process.env.REACT_APP_STRIPE_PUBLIC_KEY);
      const res = await fetch(stripeURL,{
        method : "POST",
        headers : {
          'content-type' : "application/json",
        },
        body : JSON.stringify(cartProduct.cartProductItem)
      })

      if(res.statusCode === 500) return ;

      const data = await res.json()
      console.log(data)
      toast("Redirect to payment gateway")
      stripePromise.redirectToCheckout({sessionId : data})
    }
    else{
      toast("your are not login!!")
    }
  }
}
```

```

}
/***** */
return (
  <div className="p-4">
    <h1 className="capitalize text-lg md:text-2xl font-semibold before:rounded-lg relative
before:absolute before:-bottom-2 before:content before:left-0 before:w-32 before:h-1 before:bg-red-
500 transition-all ease-in-out duration-100">
      Your Shopping Cart
    </h1>

    {cartProduct.cartProductItem[0] ? (
      <>
        <p className="mt-3 ">Product</p>
        <div className="flex flex-col md:flex-row w-full mt-1 ">
          <div className="w-full ">
            {cartProduct.cartProductItem.map((el) => {
              return (
                <CartAddedItem
                  id={el.id}
                  img={el.imgURL}
                  title={el.title}
                  qty={el.qty}
                  price={el.price}
                  total={el.total}
                />
              );
            })}
          </div>
          <div className="w-full min-w-210 mt-5 md:mt-0 md:min-w-350 max-w-lg relative">
            <PaymentSummary
              cartTotal={cartTotal}
              deliveryCharge={deliveryCharge}
              Total={Total}
              handlePayment = {handlePayment}
            />
          </div>
        </div>
      </>
    ) : (
      <EmptyCart />
    )}

    <></>
  </div>

```

```
);
};

export default Cart;
```

### Cancelled-

```
import React from 'react'

const Canceled = () => {
  return (
    <div className='w-full h-full flex justify-center items-center pt-10 px-2 md:p-10'>
      <div className='bg-slate-100 w-full max-w-md text-center flex justify-center items-center flex-
col rounded'>
        {/* <div className="">
          <img src={iconSuccess} className="h-40" />
        </div> */}
        <h1 className='text-red-600 font-bold text-lg py-3 '>payment Cancel</h1>
      </div>
    </div>
  )
}

export default Canceled
```

### Success-

```
import React from 'react'
import iconSuccess from "../assest/img/success.gif"

const Success = () => {
  return (
    <div className='w-full h-full flex justify-center items-center pt-10 px-2 md:p-10'>
      <div className='bg-slate-200 w-full max-w-md text-center flex justify-center items-center flex-
col rounded py-10'>
        <div className="">
          <img src={iconSuccess} className="h-40" />
        </div>
        <h1 className='text-green-600 font-bold text-lg py-3 '>payment successful</h1>
        <p className='text-gray-500'>Thank you for your purchase</p>
        <p className='text-gray-500'>your order on the way :)</p>
      </div>
    </div>
  )
}

export default Success
```

## Backend (server/)

The backend is developed using **Node.js** and **Express.js**, handling API requests and database operations.

- **Setup:** Initialized Node.js project with Express.js.
- **Routes:**
  - **/api/users:** Handles user registration and login.
  - **/api/products:** Manages menu items.
  - **/api/orders:** Processes customer orders.

### Key Features:

- **User Management:** Handles user registration, authentication, and profile management.
- **Product Management:** Manages CRUD operations for menu items.
- **Order Processing:** Processes customer orders and maintains order history.

### Notable Files and Directories:

- **server.js:** The main entry point that initializes the Express server and connects to MongoDB.
- **routes/:** Defines API routes for users (userRoutes.js), products (productRoutes.js), and orders (orderRoutes.js).
- **models/:** Contains Mongoose schemas for User, Product, and Order.
- **controllers/:** Implements the logic for handling requests and responses for each route.

## Controller-

```
const productmodels = require("../Models/productModel")

const addProduct = async (req, res) => {
  try {
    const { title, imgURL, category, calories, price, desc, id } = req.body;
    const product = await productmodels.create({
      title,
      imgURL,
      category,
      calories,
      price,
      desc,
      id,
    });
    if(product){
      return res.status(200).send(product);
    }
    else{
      return res.status(400).send("Un-able to add product :");
    }
  } catch (error) {
    return res.status(502).send("Internal server error");
  }
};

const getAllProduct = async(req,res)=>{

  try {
    const products = await productmodels.find();

    if(products){
      return res.status(200).send(products);
    }
  } catch (error) {
    return res.status(502).send("Internal server error");
  }
}

module.exports = {addProduct,getAllProduct}
```

## Model-

```
const mongoose = require('mongoose')
const productSchema = mongoose.Schema({
  id:{type:String},
  title:{type:String},
  imgURL:{type:String},
  category:{type:String},
  calories:{type:Number},
  qty:{type:Number,default:1},
  price:{type:Number},
  desc:{type:String}
})

const productmodels = mongoose.model('productmodels',productSchema)
module.exports=productmodels
```

## Index.js-

```
const dotenv = require("dotenv").config();
const mongoose = require('mongoose')
const Stripe = require("stripe");
const express = require("express");
const cors = require("cors");
const { addProduct, getAllProduct } = require("./Controllers/productController");
const app = express();

app.use(cors());
app.use(express.json());
app.use(express.static("public"));
// https://restaurant-e-commerce-server.vercel.app/webhook
const PORT = process.env.PORT || 8080;

const YOUR_DOMAIN = "https://e-restaurant-client.vercel.app/"

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);

app.post('/product',addProduct)
app.get('/product',getAllProduct);
app.post("/create-checkout-session", async (req, res) => {
  console.log(req.body);
  try{
    const params = {
      submit_type: "pay",
      mode: "payment",
```

```
payment_method_types: ["card"],
billing_address_collection: "auto",
// shipping_options: [{ shipping_rate: "shr_1MJTX8SHnOGYGLnPO1haAXqp" }],

line_items: req.body.map((item) => {
  return {
    price_data: {
      currency: "inr",
      product_data: {
        name: item.title,
        images: [item.imgURL],
      },
      unit_amount: item.price * 100,
    },
    adjustable_quantity: {
      enabled: true,
      minimum: 1,
    },
    quantity: item.qty,
  };
}),

//mode: "payment",
success_url: `${YOUR_DOMAIN}success`,
cancel_url: `${YOUR_DOMAIN}canceled`,
};

const session = await stripe.checkout.sessions.create(params);
res.status(200).json(session.id)
// res.redirect(303, session.url);
} catch (err) {
  res.status(err.statusCode || 500).json(err.message);
}

});

// This is your Stripe CLI webhook secret for testing your endpoint locally.
const endpointSecret =
"whsec_eb4a5f8d2421cf7cafafc42199c2f362e38a3d53865f35a5c6066cb25c31c439";

app.post('/webhook', express.raw({type: 'application/json'}), (request, response) => {
  const sig = request.headers['stripe-signature'];

  let event;
```



```
try {
  event = stripe.webhooks.constructEvent(request.body, sig, endpointSecret);
} catch (err) {
  response.status(400).send(`Webhook Error: ${err.message}`);
  return;
}

// Handle the event
switch (event.type) {
  case 'checkout.session.async_payment_failed':
    const checkoutSessionAsyncPaymentFailed = event.data.object;
    console.log("checkoutSessionAsyncPaymentFailed", checkoutSessionAsyncPaymentFailed)
    // Then define and call a function to handle the event checkout.session.async_payment_failed
    break;
  case 'checkout.session.async_payment_succeeded':
    const checkoutSessionAsyncPaymentSucceeded = event.data.object;
    console.log("checkoutSessionAsyncPaymentSucceeded", checkoutSessionAsyncPaymentSucceeded)
    // Then define and call a function to handle the event
    checkout.session.async_payment_succeeded
    break;
  case 'checkout.session.completed':
    const checkoutSessionCompleted = event.data.object;
    console.log("checkoutSessionCompleted", checkoutSessionCompleted)
    // Then define and call a function to handle the event checkout.session.completed
    break;
  case 'checkout.session.expired':
    const checkoutSessionExpired = event.data.object;
    console.log("checkoutSessionExpired", checkoutSessionExpired)
    // Then define and call a function to handle the event checkout.session.expired
    break;
  // ... handle other event types
  default:
    console.log(`Unhandled event type ${event.type}`);
}

// Return a 200 response to acknowledge receipt of the event
response.send();
});

mongoose.connect(process.env.MONGO_URL, { dbName: "RESTAURANT" })
.then(() => { console.log("Database connected succesfully")
```

```
app.listen(PORT, () => console.log("Running on port " + PORT));  
  
})  
.catch((e)=>{"Error to connect DB ",e})
```

## SOURCE CODE & PPT —

<https://github.com/akashp-007/Restaurant-E-Commerce-Website-main>

# Testing

Testing was conducted to ensure the reliability and functionality of the application:

- **Unit Testing:** Individual components and functions were tested using Jest.
- **Integration Testing:** Ensured that different parts of the application work together seamlessly.
- **User Acceptance Testing:** Gathered feedback from potential users to refine the user interface and experience.

# Application

The E-Restaurant application allows users to:

- Register and log in to their accounts.
- Browse the restaurant's menu.
- Add items to their cart.
- Place orders and make payments.
- Receive order confirmations and updates.

For restaurant administrators, the application provides functionalities to:

- Manage menu items.
- View and process customer orders.
- Update order statuses.

# Conclusion

The **E-Restaurant: An Online Restaurant E-Commerce Platform** project successfully demonstrates the power and versatility of full-stack web development using the MERN stack (MongoDB, Express.js, React.js, and Node.js). This application offers a comprehensive solution for both restaurant owners and customers by creating a seamless and interactive platform for online food ordering.

From the customer's perspective, the application ensures ease of use, fast navigation, and convenient order placement. The user interface, built with React.js, is responsive and user-friendly, allowing users to browse through available food items, view details, add items to their cart, and securely place orders. The cart and checkout functionalities have been designed with simplicity and efficiency in mind, mirroring the experience of leading food ordering platforms.

For the restaurant administration, the platform provides a centralized dashboard to manage menu items, handle customer orders, and track order statuses. The backend, powered by Node.js and Express.js, ensures smooth API communication and business logic processing, while MongoDB serves as a scalable and efficient database system for managing user data, product catalogs, and order histories.

This project adheres to the modern standards of web development by following the software development life cycle (SDLC), starting from requirement analysis and system design, to development, testing, and deployment. Each phase was carefully executed to ensure that the application not only functions correctly but also offers an optimal user experience. Unit and integration testing helped ensure the reliability and stability of the platform before final deployment.

From an academic and technical perspective, this project allowed for deep engagement with core development concepts including RESTful API design, user authentication, state management, component-based architecture, and asynchronous data handling. It also reinforced best practices in coding, debugging, version control using GitHub, and deployment readiness.

The implementation of this e-commerce solution for restaurants reflects real-world industry needs, making the project both relevant and practical. In an era where digital transformation is critical for business growth, this platform enables small and medium-

sized restaurants to expand their reach, reduce manual labor, and enhance customer satisfaction without relying heavily on third-party food delivery services.

In conclusion, the E-Restaurant system is not just a technical achievement but also a valuable learning experience that demonstrates how software can solve everyday business challenges. It stands as a scalable base for future enhancements like payment gateway integration, real-time order tracking, user feedback systems, and mobile responsiveness. The project holds strong potential for further development and real-time deployment in the restaurant industry.

The user-friendly interface and intuitive design make it accessible to users with varying degrees of technical expertise, facilitating smooth adoption without requiring extensive training. By leveraging intelligent algorithms, this system enhances decision-making processes related to student performance, curriculum development, and resource allocation. Ultimately, it aims to improve overall learning outcomes, helping educators identify areas where students may need additional support and enabling personalized learning experiences.

In essence, this intelligent educational system is a powerful tool that not only supports academic stakeholders in making informed decisions but also contributes meaningfully to the evolution of modern educational practices by integrating cutting-edge machine learning techniques.

# Bibliography(APA Style)

- React.js Official Documentation - <https://reactjs.org>
- Node.js Official Docs - <https://nodejs.org/en/docs/>
- MongoDB Manual - <https://docs.mongodb.com/manual/>
- Stripe API Reference - <https://stripe.com/docs/api>
- JWT Introduction - <https://jwt.io/introduction/>
- bcrypt GitHub Repository - <https://github.com/kelektiv/node.bcrypt.js>
- Express.js Guide - <https://expressjs.com/en/starter/guide.html>
- □ Acharya, K. (2024). *Online Food Ordering System Project Report*. Retrieved from [https://www.researchgate.net/publication/380401936\\_ONLINE\\_FOOD\\_ORDERING\\_SYSTEM\\_PROJECT\\_REPORT](https://www.researchgate.net/publication/380401936_ONLINE_FOOD_ORDERING_SYSTEM_PROJECT_REPORT)
- □ GitHub. (n.d.). *akashp-007/Restaurant-E-Commerce-Website-main*. Retrieved from <https://github.com/akashp-007/Restaurant-E-Commerce-Website-main>
- □ Scribd. (2022). *Restaurant Website Project Report*. Retrieved from <https://www.scribd.com/document/640185044/Restaurant-Website>
- □ Wikipedia contributors. (2025). *Online food ordering*. In *Wikipedia, The Free Encyclopedia*. Retrieved from [https://en.wikipedia.org/wiki/Online\\_food\\_ordering](https://en.wikipedia.org/wiki/Online_food_ordering)

**\*\*END OF REPORT\*\***