SWEET BITES : E-COMMERCE WEB APPLICATION FOR SWEETS SHOP

A Project Report

Submitted in Partial fulfilment of the
Requirements for the award of the Degree of

BACHELOR OF SCIENCE (COMPUTER SCIENCE)

By Anurag Ramniwas Dubey

Seat No._____

Under the esteemed guidance of

Prof. Javed Pathan

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Affiliated to University of Mumbai)

MUMBAI-400050

MAHARASHTRA

2025-2026

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Affiliated to University of Mumbai)

MUMBAI-MAHARASHTRA-400050

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled, "**E-COMMERCE FOR SWEETS SHOP**", is benefited work of **Anurag Ramniwas Dubey** bearing **Seat No.:_____ Roll No. 13** submitted in **Partial fulfilment of the requirements for the award of degree of BACHELOR OF SCIENCE in COMPUTER SCIENCE from University of Mumbai.**

Project Guide **HOD**

External Examiner

Date: ..... **College Seal**

ACKNOWLEDGEMENT

I owe special thanks to the department of Computer Science of **Rizvi College of Arts Science and Commerce** for giving me a chance to prepare this project. I thank the Principal, **Dr. Arunachalam S** for his leadership and management. I thank the Coordinator and the Head of the Department **Professor Arif Patel** for providing us the required facilities and guidance throughout the course which culminated into the thesis. last but not the least to the project guide for this odd semester – **Professor Javed Pathan** and also to my **Parents and Friends** for supporting me throughout the semester and helping me in finalizing the project. Also deep gratitude to the **Staff and Faculty of Rizvi College of Arts Science and Commerce** for their help and support.

SWEET BITES : E-COMMERCE WEB APPLICATION FOR SWEET SHOP USING MONGODB, EXPRESS.JS, REACT.JS, NODE.JS &TAILWIND CSS

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Affiliated to University of Mumbai)

MUMBAI- MAHARASHTRA – 400050

DEPARTMENT OF COMPUTER SCIENCE

DECLARATION

I, **Anurag Ramniwas Dubey**, Roll No.**13**, hereby declare that the project synopsis entitle "**ECOMMERCE FOR SWEETS SHOP"** submitted for approval, for **Bachelors of Science** in Computer Science Sem V project. For academic year **2025-26**.

Signature of the Guide **Signature of the Student**

Place:

INTRODUCTION

The "Sweet Bites" E-commerce platform is designed to transform the way people buy sweets by providing a user-friendly and efficient online platform specifically tailored to the unique needs of the Indian market. The application bridges the gap between traditional sweet shops and the convenience of modern digital solutions, delivering a seamless and enhanced shopping experience for users.

The application's features are designed with both functionality and security in mind. User authentication processes ensure that personal information is protected, while secure payment gateways guarantee the safety of financial transactions. The shopping cart management system allows users to easily add, remove, and review items before making a purchase, enhancing the overall shopping experience. Additionally, the application includes order tracking functionalities, enabling users to monitor the status of their purchases from the time of order placement to delivery.

Designed with the latest web technologies, the platform provides an interactive and dynamic shopping environment where users can easily browse a wide variety of sweets, including fresh, traditional, and specialty Indian confections. With advanced search and filtering options, customers can quickly find what they need by category or personal preference. The application also includes essential features like user authentication, shopping cart management, and secure payment processing, ensuring a smooth and safe shopping journey from start to finish.

Beyond convenience, the "Sweet Bites" E-commerce platform aims to support local sweet shops and vendors by offering them a digital marketplace to reach wider audience. By integrating features such as order tracking, personalized product recommendations, the platform enhances customer satisfaction and builds loyalty over time.

The application is designed with a responsive layout, making it accessible across various devices, including desktops, tablets, and smartphones. This means users can shop for sweets whenever and wherever they want, reflecting the growing trend of mobile commerce and digital convenience. Whether at home or on the go, customers can enjoy the simplicity and efficiency of online sweet shopping with just a few clicks.

OBJECTIVES

1. **Enhance User Experience:**

   To offer a seamless and intuitive online sweet shopping experience through an easy-tonavigate interface, efficient search and filter functions, and a user-friendly checkout process. This aims to make buying sweets more convenient and enjoyable for users.

2. **Enable Real-Time Product Management**:

   To create a dynamic inventory management system that provides real-time updates on the availability, prices, and promotions of sweets. This objective is crucial for maintaining accuracy in product listings to build customer trust.

3. **Facilitate Personalized Shopping:**

   To develop features that enable users to manage their accounts and receive personalized recommendations based on their shopping history and preferences. This goal is to customize the shopping experience to meet individual user needs.

4. **Ensure Secure Transactions:**

   To integrate secure payment gateways and encryption technologies to protect users' financial and personal information, creating a safe and reliable online shopping environment.

5. **Promote Mobile Responsiveness:** To design a responsive web application that ensures optimal performance and usability across various devices, including desktops, tablets, and smartphones. This aims for a consistent and effective shopping experience regardless of the device used.

6. **Support Efficient Order Fulfillment**

   To implement a robust backend system for efficient order processing, tracking, and delivery management. This is focused on streamlining the entire order fulfillment process, from the moment an order is placed to the final delivery at the customer's doorstep.

7. **Adapt to Market Trends**: To continually evolve the application based on user feedback and emerging market trends, ensuring it remains relevant and competitive in the rapidly changing e-commerce landscape. This highlights the importance of ongoing improvement and innovation.

SCOPE

1. User Experience and Interface

This includes designing and implementing a responsive and adaptable user interface that can seamlessly adjust to various devices like desktops, tablets, and smartphones. The main goal is to provide a consistent and enjoyable browsing and shopping experience for users, no matter what device they are using. This also includes providing product listings with detailed descriptions and high-quality images to help customers identify and select products.

2. Dynamic Product Management

The platform will feature a flexible and dynamic system for managing products. This system will allow administrators to easily add, update, and remove sweet items from their inventory. Products will include detailed descriptions, high-quality images, and up-to-date pricing and availability status. This ensures that the online store's product information is always accurate, helping customers make informed purchasing decisions.

3. Advanced Search and Filtering

A powerful search engine will be developed to allow users to quickly and accurately find products. The search functionality will include options for filtering by categories, price ranges, and other relevant criteria, helping customers narrow down their choices with ease.

4.User Accounts and Order Tracking

A secure and straightforward system for user registration and login will be implemented. This will enable new and returning customers to have a hassle-free experience. The system will include secure password management and will allow users to manage their personal information and view their order history. Users will also be able to track the status of their orders from placement to delivery.

5. Shopping Cart and Checkout

A user-friendly shopping cart system will be implemented to allow customers to easily add, remove, and modify items before finalizing a purchase. The shopping cart will display a clear summary of the items, including prices, quantities, and any applicable discounts.
The checkout process will be designed to be simple and secure.

METHODOLOGY

Methodology is an iterative and incremental approach to project management that is highly effective for building a dynamic e-commerce platform like a sweet shop website. Unlike a rigid, sequential model, Scrum breaks down the project into short, manageable cycles called sprints, typically lasting two to four weeks.

1. Sprint Planning

- **Objective**: To collaboratively define the work to be done in the upcoming sprint.
- **Activity**: The development team, led by the Scrum Master, will work with the Product Owner (the client or a key stakeholder) to select and prioritize tasks from the product backlog. This includes identifying user stories, such as "As a customer, I want to filter sweets by type," and breaking them down into actionable tasks for the team.

2. Development and Daily Scrums

- **Objective**: To execute the planned tasks and maintain communication within the team.
- **Activity**: The development team will work on their assigned tasks, which include designing, coding, and testing new features. Daily "Scrum" meetings will be held for 15 minutes to discuss progress, identify any obstacles, and plan for the day's work.

3. Sprint Review

- **Objective**: To showcase the completed work and gather feedback from stakeholders.
- **Activity**: At the end of each sprint, the team will present a live demonstration of the new features and functionalities to the Product Owner and other stakeholders. This is a crucial step for ensuring the project is aligned with business goals and for gathering early feedback.

4. Deployment and Maintenance

- **Objective**: To deploy the new features and ensure the application remains stable and secure.
- **Activity**: The tested and approved features will be deployed to the production environment. Continuous integration and continuous deployment (CI/CD) pipelines will be used to automate this process

TOOLS AND TECHNOLOGIES

The **Sweet Bites** e-commerce platform uses the MERN stack for its development, providing a cohesive and modern technology base. The MERN stack consists of MongoDB, Express.js, React, and Node.js, allowing developers to use a single programming language, JavaScript, throughout the entire application.

The MERN Stack

- **MongoDB:** A NoSQL database that stores data in flexible, JSON-like documents. It is used to manage and store product details, user data, and order information, and its flexible schema is ideal for scaling the application.

- **Express.js**: A lightweight web application framework for Node.js used to build robust APIs. It simplifies handling HTTP requests, defining routes, and managing server-side logic, which is crucial for efficient communication between the front-end and back-end.

- **React:** A JavaScript library that serves as the front-end of the application. React's component-based architecture enables the creation of dynamic and reusable user interface components, ensuring a seamless and interactive user experience.

- **Node.js:** A JavaScript runtime environment that allows the server-side code to be executed. It handles numerous simultaneous connections efficiently, making it suitable for scalable backend services.

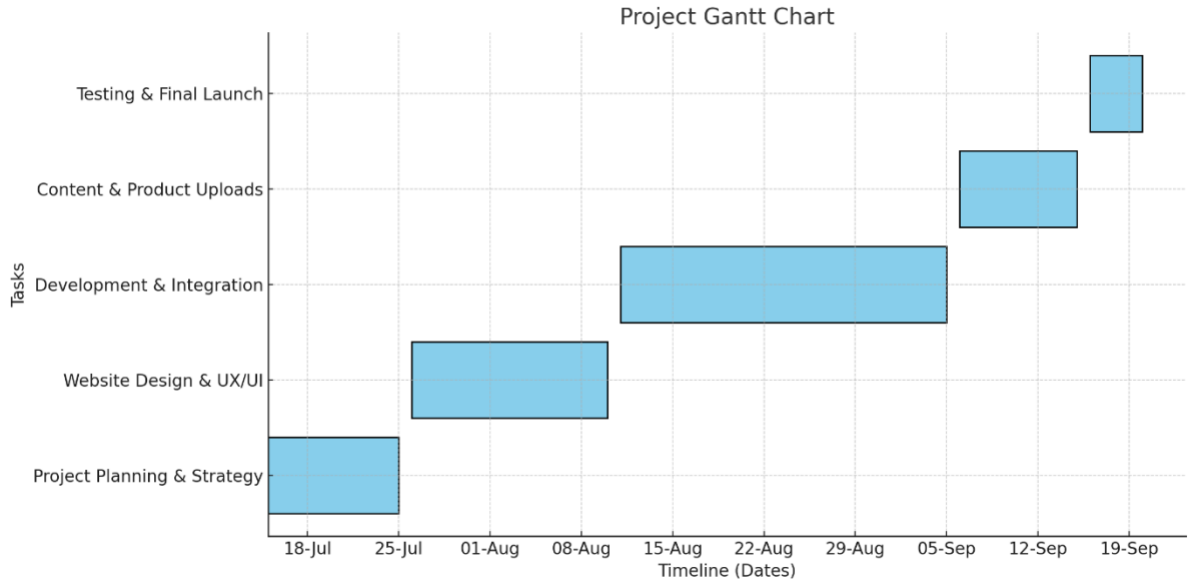
Styling and Payments

- **Tailwind CSS**: Instead of traditional CSS, the website uses Tailwind CSS. This is a utility-first CSS framework that provides low-level utility classes directly within the HTML, allowing for fast and custom design creation without writing extensive custom CSS.

- **Razorpay**: For secure transactions, the platform integrates Razorpay as the payment gateway. Razorpay is a payment processor primarily focused on the Indian market, and it is known for its ease of integration and support for a wide range of local payment methods, including UPI, net banking, and various wallets. In contrast, a company like Stripe is more geared towards a global market and has a more complex pricing structure.

GANTT CHART

The timeline of the overall project can be represented with the Gantt chart below.



TIMELINE

Task	Start Date	End Date	Duration (days)
Project Planning & Strategy	15-Jul	25-Jul	11
Website Design & UX/UI	26-Jul	10-Aug	16
Development & Integration	11-Aug	05-Sep	26
Content & Product Uploads	06-Sep	15-Sep	10
Testing & Final Launch	16-Sep	20-Sep	5

Expected Outcomes

1. **Delightful User Experience:**
   The sweet shop website is expected to offer an engaging and visually appealing interface, allowing customers to explore a wide range of sweets, cakes, and desserts effortlessly. A clean layout, vibrant product images, and intuitive navigation will make the browsing and ordering process enjoyable across devices, including desktops, tablets, and smartphones.

2. **Broader Reach and Accessibility:**
   By establishing a strong online presence, the platform will make premium sweets and desserts accessible to customers beyond physical store limits. Customers will be able to place orders conveniently anytime, anywhere, with features such as scheduled deliveries for special occasions.

3. **Promotion of Local Craftsmanship:**
   The website will act as a digital showcase for locally crafted sweets and bakery products, giving traditional recipes and unique specialties greater visibility. This will help strengthen the identity of the sweet shop while attracting both loyal customers and new audiences.

4. **Enhanced Customer Engagement:**
   Features such as customer reviews, product customization options, festive collections, and loyalty programs are expected to boost customer interaction and satisfaction. Personalized recommendations will further ensure that customers discover products that suit their tastes.

5. **Secure and Seamless Transactions:**
   With the integration of trusted payment gateways, customers will enjoy smooth and secure checkout experiences. The platform will prioritize user data protection and safe transactions, ensuring reliability in every purchase.

6. **Scalability for Business Growth:**
   Built with scalable technologies, the platform will support the addition of new products, seasonal offers, and an expanding customer base without compromising performance. This flexibility ensures long-term growth and adaptability to changing business needs.

ADVANTAGES

1. **Convenient Online Ordering:**

   The sweet shop website provides customers with the ease of browsing and purchasing their favorite sweets, cakes, and desserts from the comfort of their homes. This is especially valuable for busy individuals and those who wish to send sweets as gifts without visiting the store in person.

2. **Exclusive Variety of Products:**

   The platform showcases a wide collection of sweets, festive assortments, customizable cakes, and specialty desserts that may not be readily available in local outlets. Customers gain access to traditional favorites as well as innovative new offerings all in one place.

   By integrating features such as product customization, personalized recommendations, and curated festive collections, the website delivers a shopping experience tailored to each customer's taste and occasion.

LIMITATIONS

1. **Dependency on Internet Connectivity:**

   The platform requires a stable internet connection for customers to browse, customize, and order sweets. In regions with weak or unreliable internet access, users may face difficulties in placing or tracking their orders.

2. **Lack of Physical Inspection:**

   Customers cannot physically examine sweets, cakes, or desserts before purchase, which may cause hesitation regarding freshness, taste, or portion size compared to traditional in-store shopping.

3. **Technical Challenges and Downtime:**

   As with any online platform, there is a possibility of technical glitches, server downtime, or slow performance during peak festive seasons. These issues can temporarily affect the ordering experience and customer trust if not resolved quickly.

4. **Ongoing Maintenance and Upgrades:**

   The website requires continuous maintenance to ensure smooth performance, fix bugs, and introduce new features such as seasonal offers or product categories. Regular updates are necessary to keep the platform secure and aligned with evolving customer expectations.

REFERENCES

1) **Introduction**: Ecommerce For Sweets Shop – Geeks for Geeks

2) **Gemini**: Utilized for conceptual understanding, problem-solving, and generating code snippets.

3) **ChatGPT**: Consulted for brainstorming, refining logic, and providing alternative code implementations.

4) **Objectives**: https://saaslyft.com/marketing/objectives-of-ecommerce-businesses/

5) **Scope**: https://www.geeksforgeeks.org/computer-networks/e-commerce/

6) **Methodology:** https://www.geeksforgeeks.org/software-engineering/e-commerce-website-project-in-software-development/

7) **Limitations and Advantages**: https://unacademy.com/content/bank-exam/study-material/general-awareness/advantages-and-disadvantages-of-e-commerce/

8) **Gantt chart:** https://www.youtube.com/watch?v=xsxi4qaEnOg&t=21s%20%20

PLAGIARISM REPORT

A plagiarism report is a document or a summary that provides information about the presence of plagiarism in a piece of written or academic work. Plagiarism refers to the act of using someone else's words, ideas, or work without proper attribution or permission, presenting them as your own. Plagiarism is considered unethical and can have serious consequences, particularly in academic and professional settings.

A plagiarism report is typically generated by plagiarism detection software or services. It scans a given document or text for similarities to existing sources, such as published articles, books, websites, and other written material. When the software identifies matching or highly similar content, it highlights or marks the specific passages that may be considered plagiarized.[7]

DECLARATION

I hereby declare that the project entitled, "**E-commerce for Sweets Shop**" done at **Rizvi College of Arts, Science and Commerce** , has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university. The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (COMPUTER SCIENCE)** to be submitted as semester 5th project as part of our curriculum.

Anurag Ramniwas Dubey

ABSTRACT

The Sweet Treats E-Commerce platform is a digital marketplace developed to bring the rich and diverse world of traditional and modern sweets to a global audience. This web application offers a delightful and intuitive user interface, allowing customers to easily browse, select, and purchase a wide variety of confections, including classic Indian sweets (mithai), gourmet chocolates, and artisanal baked goods, from the comfort of their homes. The platform is meticulously designed to provide a visually appealing and convenient shopping experience, catering to the celebratory and gifting culture associated with sweets.

Built using a robust technology stack, including [mention your specific technologies, e.g., React, Node.js, and MongoDB], the application ensures a responsive and seamless experience across all devices. Key functionalities include high-quality product imagery, detailed descriptions of ingredients and preparation, a personalized "Sweet Box" feature for custom assortments, and secure payment options via [mention your payment gateway, e.g., Stripe]. Real-time order tracking and a curated selection of sweets for festivals and special occasions further enhance customer satisfaction and streamline the purchasing journey. Additionally, the platform provides a digital storefront for local sweet makers, enabling them to expand their reach beyond traditional brick-and-mortar locations and tap into a wider customer base.

Despite its advantages, such as increased market reach, a curated selection of delicacies, and enhanced convenience for customers, the platform also addresses potential challenges. These include the logistical complexities of perishable goods, ensuring product freshness and quality during delivery, and the necessity of maintaining a secure and trustworthy online environment to protect sensitive user data.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 INTRODUCTION TO THE WEB-APP

The **Sweets Shop Web Application (Sweet Bites)** is a cutting-edge digital platform developed to meet the evolving needs of consumers in the Indian confectionery market. As the demand for online shopping continues to rise, this web application seeks to transform the way people purchase sweets and desserts by offering a convenient, user-friendly, and comprehensive online solution.

The platform is designed to address the unique challenges and preferences of Indian consumers, providing them with easy access to a wide range of products — from traditional Indian sweets to modern desserts — all within a single, cohesive digital environment. Recognizing the diverse tastes and cultural significance of sweets in India, the platform ensures effortless access to an extensive range of products, catering to every occasion and preference.

This web application is not only focused on enhancing convenience for consumers but also on supporting local sweet shop vendors by offering them a digital marketplace to expand their reach. By integrating advanced technologies and features such as personalized product recommendations, secure payment processing, and a smooth order management system, the Sweets Shop Web Application aims to deliver a delightful and efficient shopping experience**.**

The platform is built using modern web technologies like React for the frontend, Node.js and Express.js for the backend, and MongoDB for data management, ensuring a robust, scalable, and responsive application. Designed with a mobile-first approach, the application is accessible across a variety of devices, allowing users to shop anytime, anywhere.

The Sweets Shop Web Application aspires to set a new standard in the eCommerce confectionery sector. It represents a significant step forward in the digital transformation of Indian retail, offering a solution that not only meets the immediate needs of today's consumers but also anticipates the future demands of a rapidly changing market.

1.2 PROBLEM DEFINITION

The traditional method of purchasing sweets in India, though culturally significant, often presents challenges that reduce consumer convenience and satisfaction. Customers frequently face issues such as limited access to a wide variety of traditional and modern sweets, long queues during festive seasons, inconsistent product availability, and the inconvenience of traveling to physical sweet shops.

For individuals with busy lifestyles, health constraints, or those living in areas where access to quality sweet shops is limited, the traditional approach becomes even more problematic. Additionally, many local sweet shops still rely heavily on offline sales and lack an effective digital presence, which restricts their ability to reach a wider audience. This gap between consumer demand for online convenience and the absence of reliable digital sweet shop platforms highlights the need for an efficient online solution.

1.3 AIM

The aim of the **Sweets Shop Web Application project** is to develop a comprehensive, user-friendly digital platform that redefines the traditional sweets shopping experience in India. The platform seeks to bridge the gap between local sweet shops and the growing demand for online shopping by offering a wide range of sweets, desserts, and confectioneries tailored to diverse consumer preferences.

The primary aim is to create an efficient, accessible, and personalized online platform that allows consumers to conveniently explore, select, and purchase sweets for everyday enjoyment, festivals, or special occasions, while simultaneously helping local sweet shop vendors expand their reach through a scalable digital marketplace.

1.4 OBJECTIVES

The **Sweets Shop Web Application** focuses on transforming the confectionery shopping experience by leveraging modern web technologies. Its objectives are:

- To create a seamless, user-friendly online platform for purchasing a wide range of traditional Indian sweets and modern desserts.
- To ensure accessibility across multiple devices, especially mobile phones, reflecting the growing trend of mobile commerce in India.
- To provide secure and reliable payment processing for customer confidence.

- To support small and medium sweet shop businesses by giving them a digital platform to showcase their products and expand their customer base.
- To enhance convenience by enabling customers to order sweets anytime, anywhere, and have them delivered efficiently.

1.5 GOAL

The primary goal of the **Sweets Shop Web Application project** is to establish a comprehensive and innovative digital platform that enhances the sweets shopping experience for Indian consumers. The project seeks to merge the charm of traditional sweets with the convenience of modern eCommerce by providing a user-centric, efficient, and accessible online solution.

The application will offer a diverse collection of sweets — from regional delicacies to popular festive desserts — within a simple and intuitive interface that allows for easy browsing, selection, and ordering. By integrating advanced web technologies and features like secure payment systems, personalized recommendations, and scheduled delivery options, the platform aims to become a trusted and indispensable tool for both consumers and sweet shop vendors.

1.6 NEED OF THE SYSTEM

The development of the **Sweets Shop Web Application** is driven by the increasing need for a more convenient, efficient, and accessible way to purchase sweets. Traditional sweet shop visits often involve crowded stores, limited product choices, and the inconvenience of physical shopping, especially during festive seasons when demand is at its peak.

In today's fast-paced lifestyle, there is a growing demand for digital platforms that allow customers to browse and purchase sweets from the comfort of their homes. At the same time, many local sweet vendors struggle to adapt to the digital marketplace due to limited resources or technical knowledge, restricting their ability to grow.

This system ensures that consumers have easy access to a wide variety of sweets while enabling local vendors to expand their business digitally. By connecting consumers and sweet shops through a unified online platform, the application fosters a more efficient, reliable, and modern marketplace for confectionery products.

CHAPTER 2. REQUIREMENT SPECIFICATION

2.1 INTRODUCTION

The Requirement Specification for the Sweets Shop Web Application is a critical document that outlines the essential features, functionalities, and constraints of the system. It serves as a comprehensive guide for the development team, ensuring that the application meets the specific needs of its users and stakeholders.

This document provides a clear description of what the system is expected to do, including user interactions, backend processes, and overall user experience. Since sweets hold cultural and festive importance in India, the application is designed to provide a user-friendly and efficient online solution for browsing, ordering, and purchasing sweets and desserts.

The requirement specification covers all aspects of the application, from the core functionalities such as product browsing, search, and secure checkout, to non-functional requirements such as performance, scalability, and security.

2.2 SYSTEM ENVIRONMENT

The System Environment for the **Sweets Shop Web Application** outlines the technological infrastructure required to support development, deployment, and operations. This includes hardware, software, networking, and security considerations to ensure smooth functioning.

2.3 SOFTWARE REQUIREMENTS

- React
- Node.js
- Express.js
- MongoDB
- Tailwind CSS
- Razorpay

2.4 METHODOLOGY

The development of the Sweets Shop Web Application follows a structured and iterative approach to deliver a high-quality, user-centric product.

- Requirements Gathering: Stakeholder consultations and market research are conducted to define functional and non-functional requirements.
- Design Phase: Creation of system architecture, UI/UX prototypes, and wireframes to visualize the application. Feedback is integrated to refine the design.
- Development Phase: Using modern technologies such as React (frontend), Node.js & Express.js (backend), and MongoDB (database) to build the platform.
- Testing Phase: Includes unit testing, integration testing, and user acceptance testing to ensure reliability, security, and performance.
- Deployment & Maintenance: Final product is deployed on a live server and continuously monitored for improvements, bug fixes, and scalability.

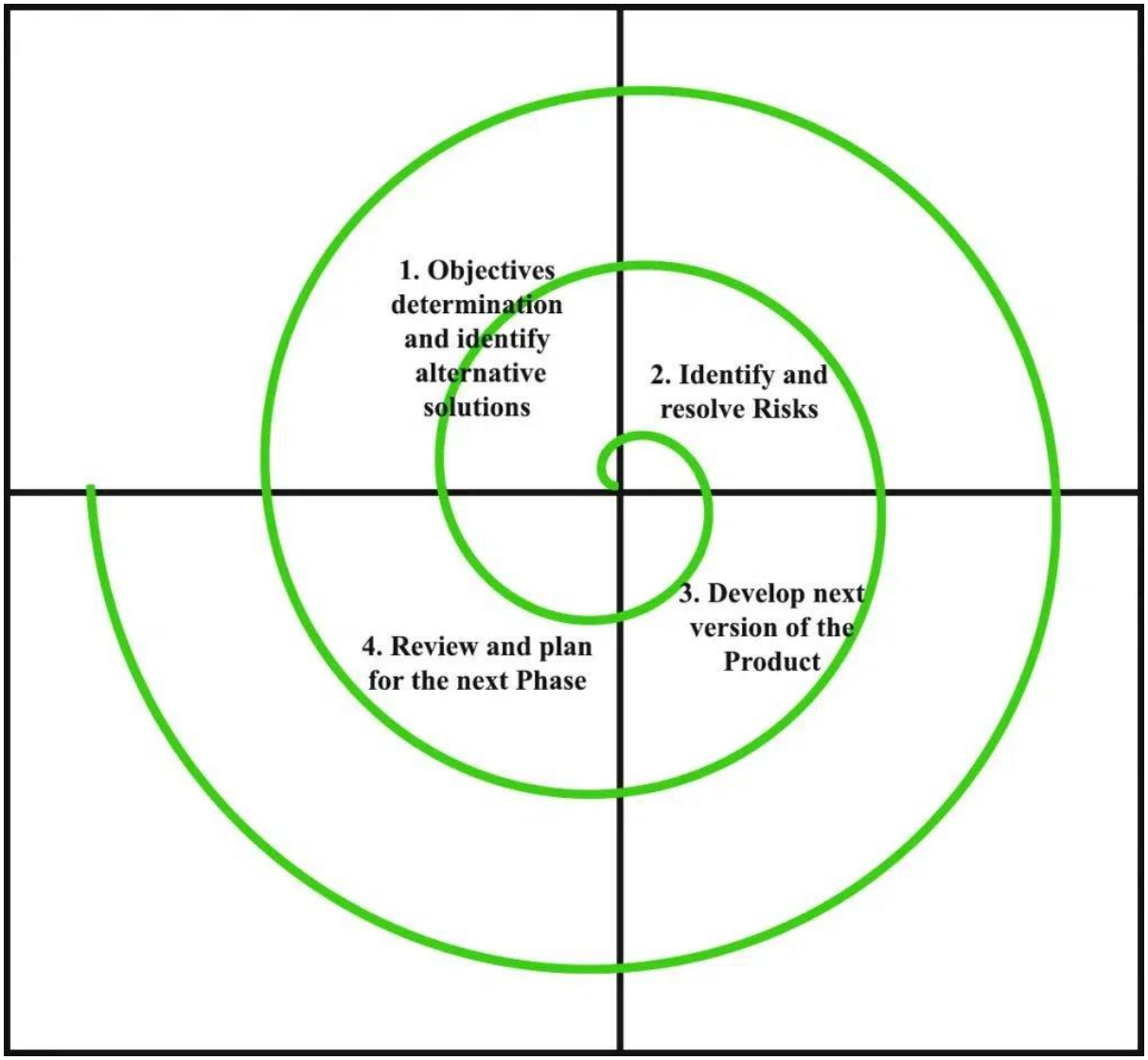
An Agile methodology is adopted throughout to allow continuous feedback and adaptation to evolving business requirements.

2.5 SPIRAL MODEL

The **Spiral Model** is chosen for the **Sweets Shop Web Application** development because it combines the structured approach of the Waterfall Model with the iterative nature of prototyping. It focuses on risk management, flexibility, and continuous refinement.
Each cycle (spiral) consists of four main phases:

1. **Planning** – Define objectives, requirements, and system features.
2. **Risk Analysis** – Identify risks such as scalability, payment gateway security, and usability issues.
3. **Engineering** – Develop, code, and test the application modules.
4. **Evaluation** – Collect feedback and refine the product for the next iteration.

Diagram Of Spiral Model



Why Spiral Model?

- Helps in managing evolving requirements (e.g., seasonal sweets, festival offers).
- Identifies risks early and allows for mitigation strategies.
- Provides flexibility for continuous refinement.
- Client (sweet shop owners) can give feedback after each iteration.

Applications of Spiral Model:

a. Ideal for projects with evolving requirements and seasonal demand variations.
b. Useful for applications requiring frequent updates and customer feedback.
c. Ensures better risk management for security and payment processing.
d. Supports iterative development and proof-of-concept before full release.

CHAPTER 3. SYSTEM ANALYSIS

3.1 SYSTEM ANALYSIS

The System Analysis involves a detailed examination of the requirements, functionalities, and operational environment of the Sweet Shop Website. The objective is to identify the key components, address user needs, and ensure that the system aligns with the expectations of both end-users and stakeholders.

Customers require a **user-friendly interface** that allows them to browse, customize, and purchase a wide range of sweet products, including cakes, pastries, and traditional sweets. Key functionalities include personalized product recommendations, an intuitive search and filtering system, and secure payment options.

The analysis identifies several core functionalities essential for the success of the Sweet Shop Website. These include:

- **User registration and authentication**
- **Shopping cart and order processing**
- **Payment gateway integration**
- **Admin Panel**

Equally important are the non-functional requirements, which include performance, security, scalability, and usability. The system must be able to handle high traffic during festive seasons and special occasions without affecting performance. Security is crucial to protect customer data and enable safe transactions.

3.2 ANALYSIS OF EXISTING SYSTEM

The analysis of existing sweet shop systems and online bakery platforms reveals several challenges that the proposed Sweet Shop Website aims to address. Traditional sweet and cake shops in India often rely heavily on physical outlets with limited digital presence. While a few shops have begun adopting online platforms, many lack the comprehensive features needed to meet evolving customer expectations.

Key issues with existing systems include:

a. Limited Online Presence:

Most local sweet shops rely on offline sales and have either minimal or no online ordering system, restricting their reach beyond the local area.

b. Fragmented User Experience:

Existing bakery and sweet platforms often lack smooth navigation, product customization features, and effective filtering/searching tools. Customers struggle to find exactly what they want quickly.

c. Security and Payment Concerns:

Many small-scale platforms do not integrate secure payment gateways, leading to trust issues among customers who hesitate to pay online.

d. Scalability Issues:

Current systems often fail to handle high user traffic during festivals (Diwali, Raksha Bandhan, etc.), leading to delays, transaction failures, and poor customer experiences.

3.3 ANALYSIS OF THE PROPOSED SYSTEM

The Sweet Shop Website is designed to overcome the challenges identified in existing systems and deliver a **modern, customer-centric solution** tailored to both consumers and shop owners. The proposed system focuses on enhancing the digital sweet shopping experience with advanced features and reliable performance.

Key benefits of the proposed system include:

a. Enhanced User Experience:

The system prioritizes a seamless and intuitive user interface that works across all devices (desktop, tablet, and mobile). Customers can easily browse sweets, customize cakes, and schedule deliveries, making the shopping experience smooth and enjoyable.

b. Comprehensive Vendor Support:

The platform empowers the shop owner with tools to manage inventory, update product listings, Delete Product, Add products.

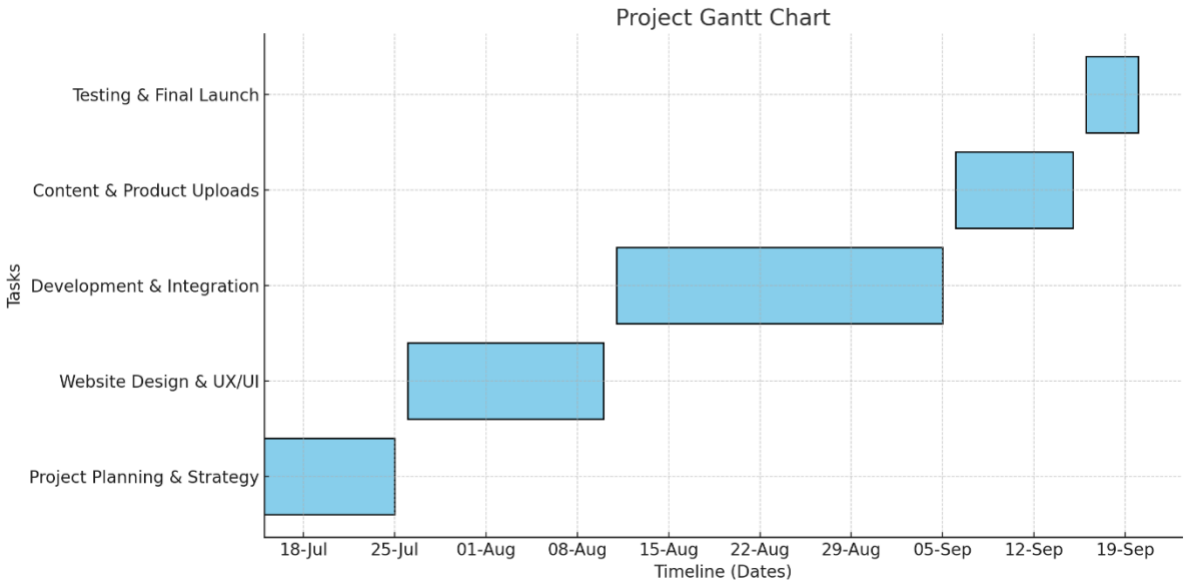
c. Scalability and Performance:

Built with scalability in mind, the backend system (using technologies like Node.js, Express, MongoDB and Postman API) ensures the platform can handle a growing number of users and high-volume transactions, especially during festive rushes.

d. Secure and Versatile Payment Processing:

To address payment concerns, the system integrates with secure gateways like Razorpay. Customers can choose from multiple payment options (credit/debit cards, UPI, wallets, net banking) with strong encryption to ensure safe and reliable transactions.

3.4 GANTT CHART[12]



Task	Start Date	End Date	Duration (days)
Project Planning & Strategy	15-Jul	25-Jul	11
Website Design & UX/UI	26-Jul	10-Aug	16
Development & Integration	11-Aug	05-Sep	26
Content & Product Uploads	06-Sep	15-Sep	10
Testing & Final Launch	16-Sep	20-Sep	5

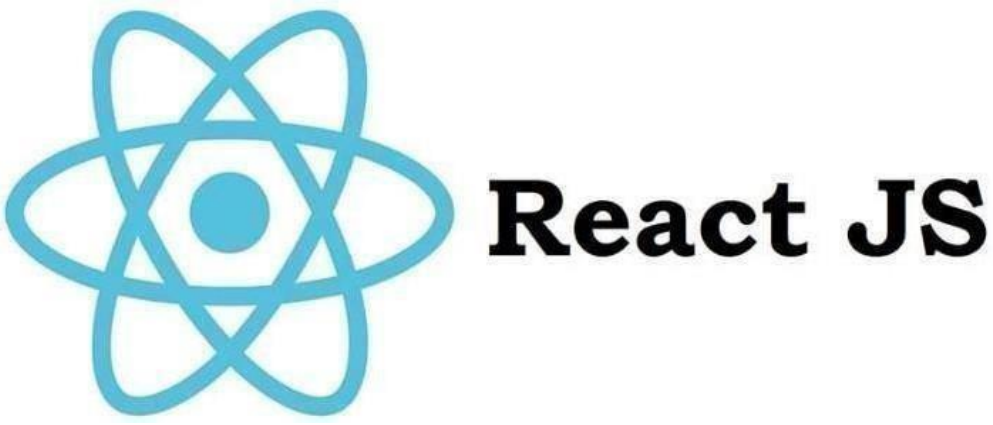
CHAPTER 4. SURVEY OF TECHNOLOGY

4.1 React JS

ReactJS is a widely adopted JavaScript library developed by Facebook, primarily used for building interactive and responsive user interfaces. Since its release in 2013, React has become one of the leading technologies for developing modern web applications. In the context of the **Sweets Shop Project**, ReactJS enables the creation of a dynamic and engaging frontend where customers can browse sweets, customize orders, and interact with the application seamlessly.

React's component-based architecture allows the UI to be broken down into reusable and self-contained components such as product cards, shopping carts, and checkout forms. This modular approach enhances maintainability and scalability, making it easier to update and extend features.

A major strength of React is its Virtual DOM implementation, which improves performance by updating only the components that change rather than re-rendering the entire page. Combined with React Router for smooth navigation and strong community support, React ensures a high-performance, scalable, and user-friendly frontend for the sweets shop application.[1]



4.2 Tailwind CSS

Tailwind CSS is a modern utility-first CSS framework designed to provide developers with low-level utility classes for designing responsive and customizable user interfaces. Unlike traditional CSS or frameworks like Bootstrap, Tailwind CSS allows developers to style elements directly within their markup using predefined utility classes, leading to faster development and consistent designs.

For the **Sweets Shop Project**, Tailwind CSS simplifies the process of designing visually appealing layouts, ensuring that the application is both responsive and user-friendly across devices. With features such as flexbox utilities, grid systems, spacing controls, and colour utilities, developers can rapidly build a professional and customized design without writing extensive custom CSS.

Additionally, Tailwind CSS supports animations, and responsive breakpoints, which enhance the overall aesthetic and user experience. This approach makes it possible to create a modern, elegant, and responsive UI for showcasing sweets, managing user accounts, and streamlining the checkout process.[2]

4.3 Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment built on Chrome's V8 engine. It allows developers to execute JavaScript on the server side, enabling the creation of scalable, event-driven applications. Introduced in 2009 by Ryan Dahl, Node.js has become a popular choice for backend development due to its non-blocking I/O operations and asynchronous event-driven architecture.

In the **Sweets Shop Project**, Node.js powers the server-side logic, handling tasks such as managing user authentication, processing orders, and interacting with the database. Its ability to manage multiple simultaneous connections makes it ideal for e-commerce applications where numerous users may browse and place orders concurrently.

With access to the Node Package Manager (NPM), Node.js offers a vast ecosystem of libraries that accelerate development, enabling the integration of functionalities such as image uploads, API handling, and payment processing.[3]

4.4 Express.js

Express.js is a lightweight and flexible web application framework built on top of Node.js, designed to simplify server-side application development. Released in 2010, Express provides a straightforward API for managing routes, handling HTTP requests, and integrating middleware.

In the **Sweets Shop Project**, Express.js serves as the backbone of the backend, enabling efficient handling of operations such as product management, cart functionality, order tracking, and user authentication. Its middleware architecture allows developers to add custom logic for security, logging, and validation, ensuring a robust and secure application.

Express's simplicity, flexibility, and ability to integrate seamlessly with databases like MongoDB make it a powerful framework for building scalable APIs and backend services required for the sweets shop application.[4]

4.5 MongoDB with Mongoose

MongoDB is a leading NoSQL, document-oriented database designed to handle large volumes of semi-structured and unstructured data. It stores data in a flexible JSON-like BSON format, allowing developers to model complex and evolving datasets efficiently.

For the **Sweets Shop Project**, MongoDB is used to store and manage data such as product details, user accounts, order history, and payment transactions. Its schema-less structure enables easy modifications as business requirements evolve. To enhance interactions with MongoDB, Mongoose, an Object Data Modeling (ODM) library, is employed.

Mongoose provides a structured schema-based approach, simplifying database operations such as validation, queries, and relationships between collections. This combination ensures efficient data management, scalability, and reliability for the sweets shop application.[5]

4.6 Razorpay

Razorpay is a leading Indian payment gateway platform that enables businesses to accept, process, and disburse payments online. Founded in 2014, Razorpay provides a seamless and secure solution for handling digital payments, making it a preferred choice for e-commerce platforms in India.

In the **Sweets Shop Project**, Razorpay is integrated to manage customer payments during the checkout process. It supports multiple payment methods, including credit/debit cards, UPI, net banking, and popular digital wallets, ensuring convenience for users.

Razorpay's API-driven approach allows for smooth integration into the application, while features such as real-time payment tracking, subscription billing, and advanced security mechanisms (including encryption and fraud detection) ensure safe and reliable transactions.

By leveraging Razorpay, the sweets shop application offers a secure and efficient payment experience, enhancing customer trust and satisfaction.[6]

CHAPTER 5. SYSTEM DESIGN

5.1 INTRODUCTION

In the rapidly growing digital commerce ecosystem, the system design of our **Sweets Shop Web Application** represents a crucial stage in bridging the conceptual vision of an online sweets store with its real-world implementation. This design phase ensures that the platform provides a seamless, secure, and user-friendly experience for both customers and administrators.

The design is structured to support key functionalities, including browsing a wide range of sweets, customizing orders, managing carts, processing secure payments through Razorpay, and enabling administrators to efficiently manage products, pricing, and orders.

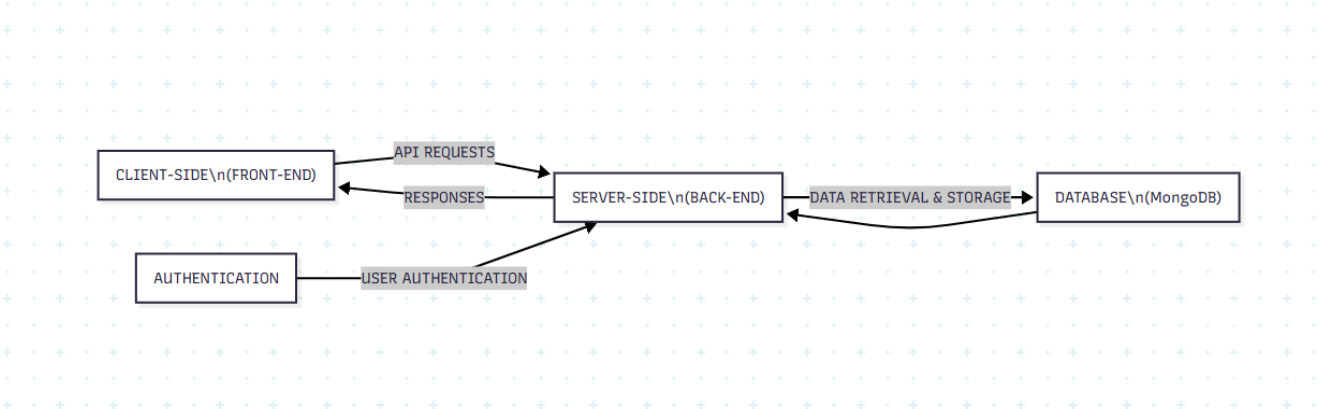
Through this chapter, we outline the core components, technical architecture, workflows, and data models that form the backbone of the application. The design ensures scalability, reliability, and smooth operations, laying the foundation for future upgrades such as personalized recommendations, scheduled deliveries, and integration of loyalty programs.

5.2 SYSTEM ARCHITECTURE DIAGRAM

The system architecture of the **Sweets Shop Web Application** follows a three-tier architecture model, consisting of:

1. **Frontend (Client-Side):**
    - Built using ReactJS and Tailwind CSS for an interactive and responsive user interface.
    - Handles product browsing, cart management, user login/registration, and checkout.
2. **Backend (Server-Side):**
    - Developed with Node.js and Express.js, managing APIs, authentication, business logic, and secure communication with the database.
3. **Database Layer:**
    - Implemented with MongoDB (via Mongoose) to store user details, product catalog, cart data, order history, and payment records.
4. **Payment Gateway Integration:**
    - Razorpay handles secure payment processing, supporting credit/debit cards, UPI, net banking, and digital wallets.

Together, these layers ensure a robust, scalable, and secure system to deliver smooth user and admin experiences.



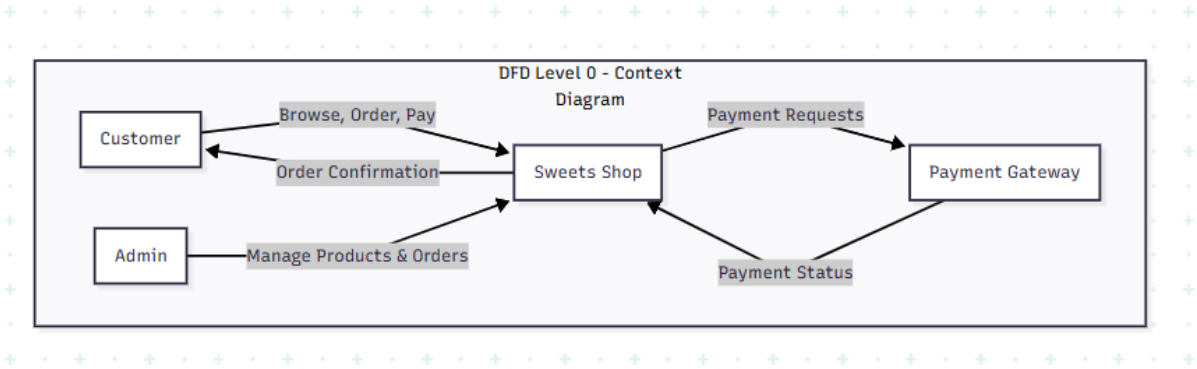
5.3 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) illustrates how data moves between users, the sweets shop application, and external services (like Razorpay). It highlights interactions such as user registration, login, browsing sweets, adding/removing items in the cart, processing payments, and order management.

5.3.1 DFD Level – 0

At this level, the **Sweets Shop Web Application** is represented as a single process. External entities include:

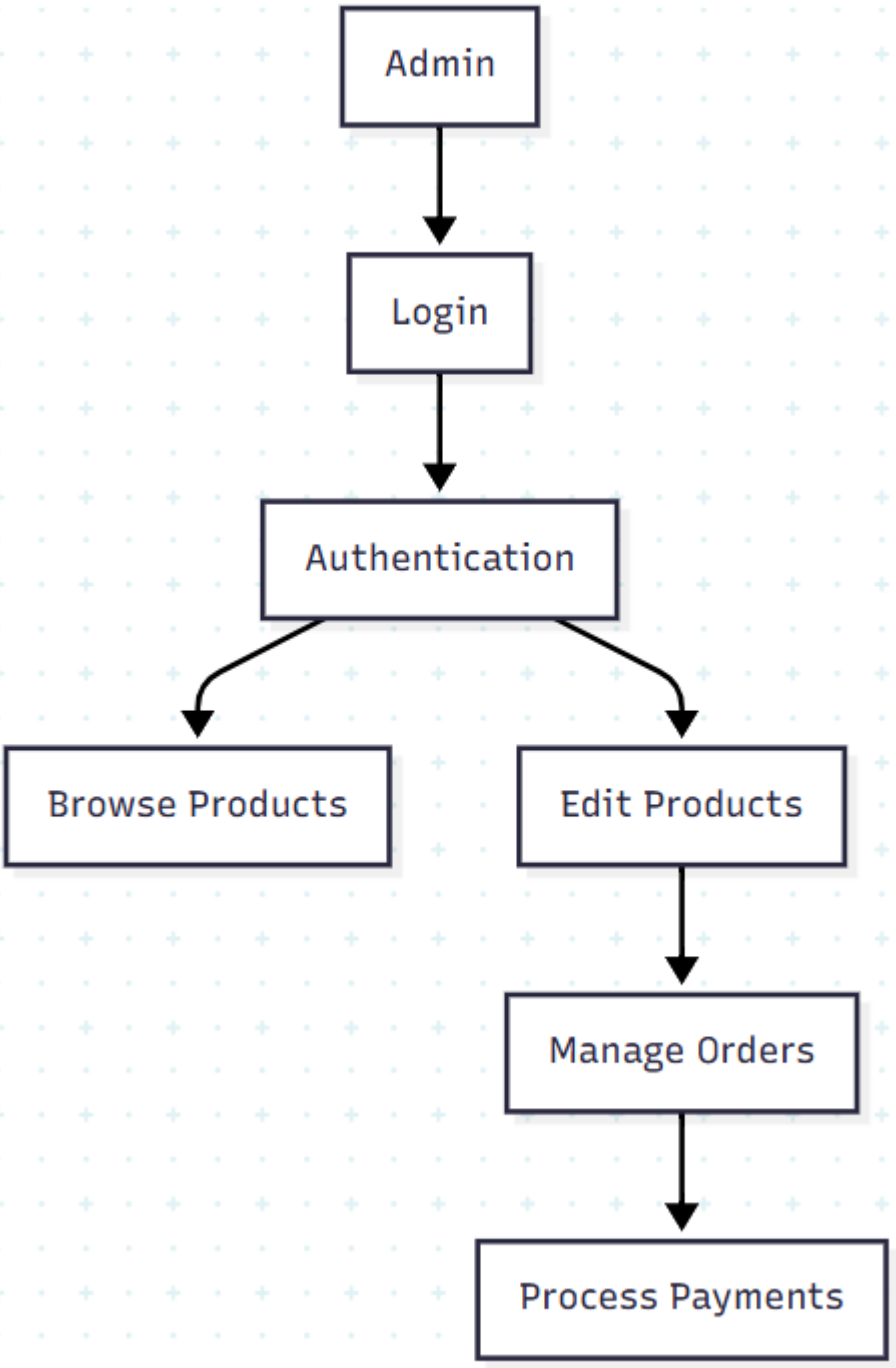
- **User/Customer** → Interacts with the system for browsing, ordering, and payments.
- **Admin** → Manages products, pricing, and orders.
- **Razorpay Payment Gateway** → Handles secure transactions.[11]



5.3.2 DFD Level – 1

This level provides more detail on internal processes:

- **User Authentication:** Registration and login handled with secure validation.
- **Product Browsing & Search:** Fetches sweets data from MongoDB and displays on the frontend.
- **Cart Management:** Add, update, or remove sweets from the cart.
- **Order Processing:** Validates cart items, calculates totals, and stores order details.
- **Payment Processing:** Integrates with Razorpay for secure payment handling.
- **Admin Operations:** Add/edit sweets, update stock and prices, and manage user orders.[11]

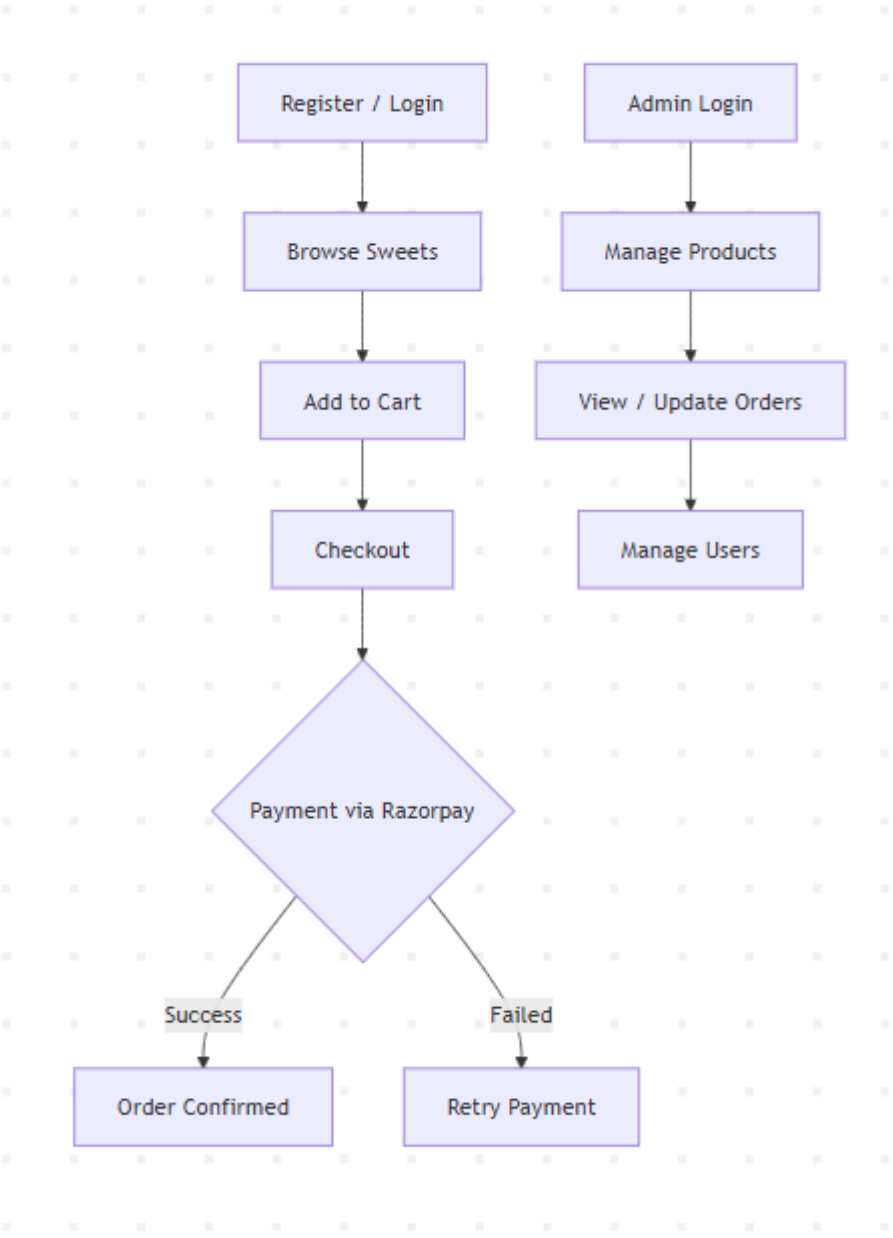

Admin
Login
Authentication
Browse Products
Edit Products
Manage Orders
Process Payments

5.4 ACTIVITY DIAGRAM

The activity diagram represents workflows within the sweets shop application, such as:

1. **User Journey:**
    - Register/Login → Browse sweets → Add to cart → Checkout → Pay via Razorpay → Order confirmed.
2. **Admin Journey:**
    - Login → Add or manage products → View and update orders → Manage users.

This helps visualize sequential flows, decision points (e.g., successful or failed payments), and system responses.[10]

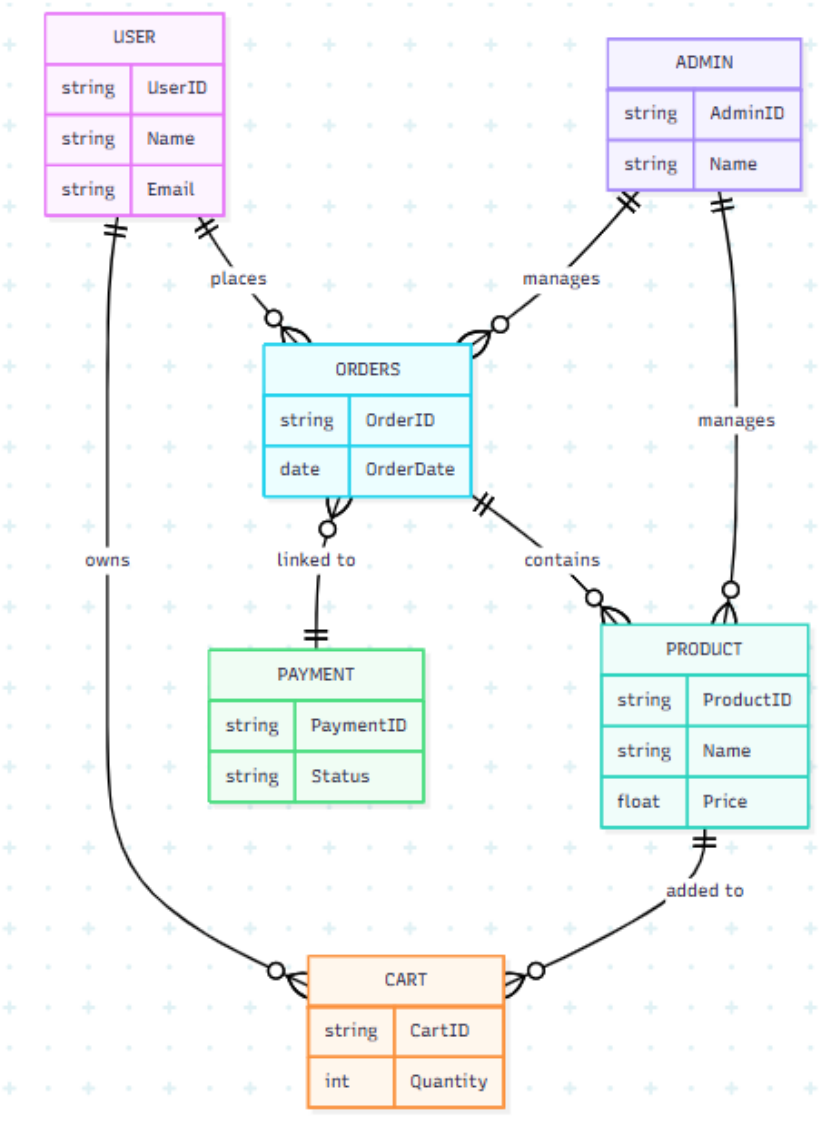


5.5 E-R DIAGRAM

The **Entity-Relationship (ER) Diagram** defines the logical structure of the database for the sweets shop:

- **User Entity:** Attributes include UserID, Name, Email, Password.
- **Product Entity (Sweets):** Attributes include ProductID, Name, Price.
- **Cart Entity:** Stores Quantity, and CartID.
- **Order Entity:** Includes OrderID and OrderDate
- **Payment Entity:** Includes PaymentID and Status.

This ER model ensures that data relationships are clear, enabling efficient database operations and reliable system performance.[9]



CHAPTER 6. SYSTEM IMPLEMENTATION

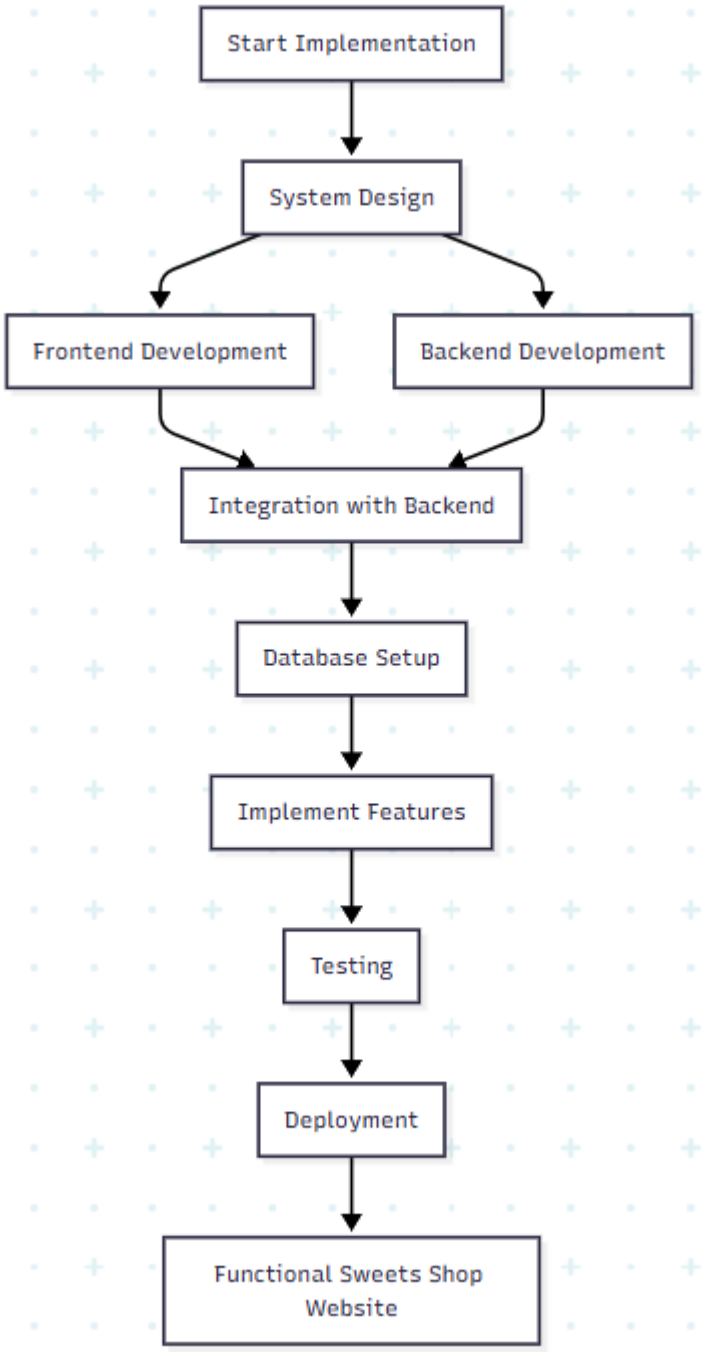
6.1 INTRODUCTION

System implementation is a crucial phase in the development of the **Sweets Shop Web Application**, as it involves transforming the planned design and architecture into a fully functional and interactive platform. This stage covers the deployment of the system, including the integration of the frontend and backend, setting up the database, and ensuring smooth communication between the user interface and the server.

During this phase, the various features of the system such as user authentication, product browsing, cart management, order processing, payment integration, and admin operations are made fully operational according to the predefined requirements. Rigorous testing is carried out to detect and fix any issues, ensuring reliability and efficiency.

The primary goal of system implementation is to deliver a secure, scalable, and user-friendly web application that allows customers to conveniently browse sweets, manage their carts, place orders, and make secure payments, while also enabling administrators to manage products and orders effectively. Ultimately, this phase ensures that the Sweets Shop Web Application performs efficiently and provides an enjoyable online shopping experience.

6.2 FLOWCHART[12]



6.3 CODING

FRONTEND

Topbar.jsx

```
import { TbBrandMeta } from "react-icons/tb";
import { IoLogoInstagram } from "react-icons/io";
import { RiTwitterXLine } from "react-icons/ri";
import { useEffect, useState } from "react";
const Topbar = () => {
  const [visible, setVisible] = useState(true);
  useEffect(() => {
    const handleScroll = () => {
      // Hide topbar when scrolling down, show when at top
      if (window.scrollY > 100) {
        setVisible(false);
      } else {
        setVisible(true);
      }
    };
    window.addEventListener('scroll', handleScroll);
    return () => window.removeEventListener('scroll', handleScroll);
  }, []);
  return (
      <div className={`bg-amber-50 text-gray-700 text-sm border-b border-amber-100 transition-all duration-300 ${visible ? 'opacity-100' : 'opacity-0 -translate-y-full'}`}>
    <div className="container mx-auto flex justify-between items-center py-2 px-4 sm:px-6 lg:px-12">
        {/* Social Icons */}
        <div className="hidden md:flex items-center space-x-4">
          <a href="#" className="hover:text-amber-600 transition-colors duration-200">
            <TbBrandMeta className="h-4 w-4" />
          </a>
          <a href="#" className="hover:text-amber-600 transition-colors duration-200">
```

```
          <IoLogoInstagram className="h-4 w-4" />
        </a>
        <a href="#" className="hover:text-amber-600 transition-colors duration-200">
          <RiTwitterXLine className="h-4 w-4" />
        </a>
      </div>
      {/* Center Text */}
      <div className="flex-grow text-center font-medium tracking-wide animate-fadeIn">
        Welcome to <span className="text-amber-600 font-semibold">Sweet Bites</span>
      </div>
      {/* Contact */}
      <div className="hidden md:block text-sm">
        <a href="tel:+919999999999"
          className="hover:text-amber-600 font-medium transition-colors duration-200">
          +91 9999 99 9999
        </a>
      </div>
    </div>
  </div>
 );
};
export default Topbar;
```

Navbar.jsx

```
import { Link } from "react-router-dom";
import { HiBars3BottomRight, HiOutlineUser } from "react-icons/hi2";
import { HiOutlineShoppingBag } from "react-icons/hi2";
import SearchBar from "./SearchBar";
import CartDrawer from "../Layout/CartDrawer";
import React, { useState, useEffect } from "react";
import { IoMdClose } from "react-icons/io";
import { useSelector } from "react-redux";
```

```
const Navbar = () => {
  const [cartDrawerOpen, setCartDrawerOpen] = useState(false);
  const [navDrawerOpen, setNavDrawerOpen] = useState(false);
  const [scrolled, setScrolled] = useState(false);
  const {cart} = useSelector((state) => state.cart);
  const {user} = useSelector((state) => state.auth);

  const cartItemCount =
  cart?.products?.reduce((
    total,
    product) =>
      total + product.quantity, 0) ||
    0;

  // Handle scroll effect
  useEffect(() => {
    const handleScroll = () => {
      const isScrolled = window.scrollY > 10;
      if (isScrolled !== scrolled) {
        setScrolled(isScrolled);
      }
    };

    window.addEventListener('scroll', handleScroll);
    return () => {
      window.removeEventListener('scroll', handleScroll);
    };
  }, [scrolled]);

  const toggleNavDrawer = () => setNavDrawerOpen(!navDrawerOpen);
  const toggleCartDrawer = () => setCartDrawerOpen(!cartDrawerOpen);

  return (
```

```
<>
 <nav className={`sticky top-0 left-0 right-0 z-40 transition-all duration-300 ${scrolled ? 'bg-white/95 shadow-md backdrop-blur-sm' : 'bg-white shadow-sm'}`}>
    <div className="container mx-auto flex justify-between items-center py-4 px-4 sm:px-6 lg:px-12">
    {/* Left - Logo */}
    <div className="animate-fadeIn">
     <Link to="/" className="flex items-center">
       <span className="text-xl font-semibold text-amber-600 hover:text-amber-700 leading-none transition-colors duration-200">
         Sweet Bites
       </span>
     </Link>
    </div>

   {/* Center - Navigation Links */}
   <div className="hidden md:flex space-x-8 animate-fadeIn">
     <Link
       to="/collections/all"
         className="text-gray-700 hover:text-amber-600 relative group text-sm font-medium uppercase tracking-wide transition-colors duration-200"
     >
       All Sweets
         <span className="absolute -bottom-1 left-0 w-0 h-0.5 bg-amber-400 group-hover:w-full transition-all duration-300"></span>
     </Link>
     <Link
       to="/collections/all?category=Rajbhog"
         className="text-gray-700 hover:text-amber-600 relative group text-sm font-medium uppercase tracking-wide transition-colors duration-200"
     >
       Rajbhogs
         <span className="absolute -bottom-1 left-0 w-0 h-0.5 bg-amber-400 group-hover:w-full transition-all duration-300"></span>
     </Link>
```

```
<Link
  to="/contact"
    className="text-gray-700 hover:text-amber-600 relative group text-sm font-medium uppercase tracking-wide transition-colors duration-200"
>
  Contact Us
    <span className="absolute -bottom-1 left-0 w-0 h-0.5 bg-amber-400 group-hover:w-full transition-all duration-300"></span>
</Link>
<Link
  to="/about"
    className="text-gray-700 hover:text-amber-600 relative group text-sm font-medium uppercase tracking-wide transition-colors duration-200"
>
  About Us
    <span className="absolute -bottom-1 left-0 w-0 h-0.5 bg-amber-400 group-hover:w-full transition-all duration-300"></span>
</Link>
</div>


{/* Right - Icons */}
<div className="flex items-center space-x-5 animate-fadeIn">
 {user && user.role === "admin" && ( <Link
  to="/admin"
    className="block bg-amber-600 px-3 py-1 rounded-full text-sm text-white hover:bg-amber-700 transition-all duration-300 shadow-sm hover:shadow-md"
>
  Admin
</Link> )}
<Link
  to="/Profile"
  className="text-gray-700 hover:text-amber-600 transition-colors duration-200"
>
  <HiOutlineUser className="h-6 w-6" />
```

```
          </Link>
          <button
            onClick={toggleCartDrawer}
            className="relative hover:text-amber-600 transition-colors duration-200"
          >
            <HiOutlineShoppingBag className="h-6 w-6 text-gray-700 hover:text-amber-600 transition-colors duration-200" />
            {cartItemCount > 0 && (<span className="absolute -top-1 -right-1 bg-amber-600 text-white text-xs rounded-full h-5 w-5 flex items-center justify-center shadow-sm">
              {cartItemCount}
            </span> )}

          </button>
          <div className="overflow-hidden">
            <SearchBar />
          </div>
          <button className="md:hidden" onClick={toggleNavDrawer}>
            <HiBars3BottomRight className="h-6 w-6 text-gray-700 hover:text-amber-600 transition-colors duration-200" />
          </button>
        </div>
      </div>
    </nav>

    {/* No spacer needed with sticky positioning */}

    {/* Cart Drawer */}
    <CartDrawer
      drawerOpen={cartDrawerOpen}
      toggleCartDrawer={toggleCartDrawer}
    />

    {/* Mobile Navigation Drawer with Overlay */}
     <div
```

```
        onClick={toggleNavDrawer}
          className={`fixed inset-0 bg-black/30 backdrop-blur-sm transition-opacity z-50 ${navDrawerOpen ? "opacity-100" : "opacity-0 pointer-events-none"}`}
      ></div>


      {/* Mobile Navigation Drawer */}
      <div
          className={`fixed top-0 left-0 w-3/4 sm:w-1/2 md:w-1/4 h-full bg-white/95 backdrop-blur-sm shadow-lg transform transition-transform
          duration-300 z-50 ${navDrawerOpen ? "translate-x-0" : "-translate-x-full"
          }`}
      >
        <div className="flex justify-between items-center p-4 border-b border-gray-100">
            <h2 className="text-xl font-semibold text-amber-600 animate-fadeIn">Sweet Menu</h2>
          <button
            onClick={toggleNavDrawer}
            className="text-gray-600 hover:text-amber-600 hover:bg-gray-100 p-2 rounded-full transition-colors duration-200"
          >
            <IoMdClose className="h-6 w-6" />
          </button>
        </div>
        <div className="p-4">
          <nav className="space-y-4 animate-slideUp">
            <Link
              to="/collections/all"
              className="flex items-center text-gray-700 hover:text-amber-600 text-base font-medium uppercase transition-colors duration-200"
              onClick={toggleNavDrawer}
            >
              <span className="h-1.5 w-1.5 rounded-full bg-amber-400 mr-2"></span>
              All Sweets
            </Link>
```

```
<Link
  to="/rajbhog"
  className="flex items-center text-gray-700 hover:text-amber-600 text-base font-medium uppercase transition-colors duration-200"
  onClick={toggleNavDrawer}
>
  <span className="h-1.5 w-1.5 rounded-full bg-amber-400 mr-2"></span>
  Rajbhogs
</Link>
<Link
  to="/contact"
  className="flex items-center text-gray-700 hover:text-amber-600 text-base font-medium uppercase transition-colors duration-200"
  onClick={toggleNavDrawer}
>
  <span className="h-1.5 w-1.5 rounded-full bg-amber-400 mr-2"></span>
  Contact Us
</Link>
<Link
  to="/about"
  className="flex items-center text-gray-700 hover:text-amber-600 text-base font-medium uppercase transition-colors duration-200"
  onClick={toggleNavDrawer}
>
  <span className="h-1.5 w-1.5 rounded-full bg-amber-400 mr-2"></span>
  About us
</Link>
<div className="border-t border-gray-100 my-4"></div>
<Link
  to="/profile"
  className="flex items-center text-gray-700 hover:text-amber-600 text-base font-medium uppercase transition-colors duration-200"
  onClick={toggleNavDrawer}
>
```

```
            <span className="h-1.5 w-1.5 rounded-full bg-amber-400 mr-2"></span>
            My Account
          </Link>
          <Link
            to="/admin"
            className="flex items-center text-gray-700 hover:text-amber-600 text-base font-medium uppercase transition-colors duration-200"
            onClick={toggleNavDrawer}
          >
            <span className="h-1.5 w-1.5 rounded-full bg-amber-400 mr-2"></span>
            Admin Portal
          </Link>
        </nav>
      </div>
    </div>
  </>
 );
};
export default Navbar;
```

Header.jsx

```
import Topbar from "../Layout/Topbar";
import Navbar from "./Navbar";
const Header = () => {
 return (
  <header>
    {/* Topbar */}
    <Topbar />
    {/* Navbar */}
    <Navbar />
    {/* Cart Drawer */}
  </header>
 );
```

```
};
export default Header;
```

Footer.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import { TbBrandMeta } from 'react-icons/tb';
import { IoLogoInstagram } from 'react-icons/io';
import { RiTwitterXLine } from 'react-icons/ri';
import { FiPhoneCall } from 'react-icons/fi';
import { HiOutlineLocationMarker } from "react-icons/hi";
import { HiMail } from "react-icons/hi";

const Footer = () => {
   return (
      <footer className="border-t py-14 bg-amber-50">
         <div className="container mx-auto px-6">
            {/* Grid wrapper */}
            <div className="grid grid-cols-1 gap-10 md:grid-cols-4">
               {/* Newsletter Section */}
               <div className="md:col-span-1">
                <h3 className="text-lg font-semibold text-gray-800 mb-4">Newsletter</h3>
                  <p className="text-gray-600 mb-3 text-sm leading-relaxed">
                     Be the first to hear about new products, exclusive events,
                     and special offers.
                  </p>
                  <p className="text-gray-600 mb-5 text-sm">
                        Sign up now & enjoy <span className="font-semibold text-amber-
600">10% off</span> on your first order.
                  </p>

                  <form className="flex">
                     <input
```

```
                    type="email"
                    placeholder="Enter your Email"
                    className="p-3 w-full text-sm border border-amber-200 rounded-l-md focus:outline-none
                     focus:ring-2 focus:ring-amber-500 transition-all placeholder-gray-400"
                    required
                  />
                  <button
                    type="submit"
                     className="bg-amber-600 text-white px-6 text-sm rounded-r-md hover:bg-amber-700
                      transition-all focus:outline-none focus:ring-2 focus:ring-amber-500"
                  >
                    Subscribe
                  </button>
                </form>
              </div>

              {/* Other sections */}
            <div className="grid grid-cols-2 gap-10 sm:grid-cols-2 md:grid-cols-3 md:col-span-3">
                {/* Shop Links */}
                <div>
                  <h3 className="text-lg font-semibold text-gray-800 mb-4">Shop</h3>
                  <ul className="space-y-3 text-gray-600">
                      <li><Link to="/collections/all" className="hover:text-amber-600 transition-colors text-sm flex items-center gap-2">All Sweets</Link></li>
                    <li><Link to="/rajbhog" className="hover:text-amber-600 transition-colors text-sm">Rajbhogs</Link></li>
                                    <li><Link to="/collections/all?sweetType=mithai" className="hover:text-amber-600 transition-colors text-sm">Mithai <span className="text-xs bg-amber-100 text-amber-800 px-2 py-0.5 rounded-full">New</span></Link></li>
```

```
                                    <li><Link to="/collections/all?sweetType=ladoo"
className="hover:text-amber-600 transition-colors text-sm">Ladoos <span
className="text-xs bg-amber-100 text-amber-800 px-2 py-0.5 rounded-
full">New</span></Link></li>
                  </ul>
                </div>

                {/* Support Links */}
                <div>
                 <h3 className="text-lg font-semibold text-gray-800 mb-4">Support</h3>
                   <ul className="space-y-3 text-gray-600">
                      <li><Link to="/contact" className="hover:text-amber-600 transition-
colors text-sm">Contact Us</Link></li>
                       <li><Link to="/about" className="hover:text-amber-600 transition-
colors text-sm">About Us</Link></li>
                      <li><Link to="#" className="hover:text-amber-600 transition-colors
text-sm">FAQs</Link></li>
                      <li><Link to="#" className="hover:text-amber-600 transition-colors
text-sm">Features</Link></li>
                   </ul>
                </div>

                {/* Follow Us / Contact */}
                <div>
                      <h3 className="text-lg font-semibold text-gray-800 mb-4">Follow
Us</h3>
                   <div className="flex items-center space-x-5 mb-6">
                      <a href="https://www.facebook.com" target="_blank" rel="noopener
noreferrer"
                       className="text-gray-500 hover:text-amber-600 transition-colors">
                        <TbBrandMeta className="h-6 w-6" />
                      </a>
                       <a href="https://www.instagram.com" target="_blank" rel="noopener
noreferrer"
                       className="text-gray-500 hover:text-amber-600 transition-colors">
                        <IoLogoInstagram className="h-6 w-6" />
```

```
                    </a>
                    <a href="https://www.x.com" target="_blank" rel="noopener noreferrer"
                      className="text-gray-500 hover:text-amber-600 transition-colors">
                      <RiTwitterXLine className="h-5 w-5" />
                    </a>
                  </div>

                  <h3 className="text-lg font-semibold text-gray-800 mb-2">Contact Us</h3>
                  <p className="text-gray-500 flex items-center text-sm hover:text-amber-600 transition-colors">
                    <FiPhoneCall className="mr-2 text-base text-amber-500" />
                    +91 9999 99 9999
                  </p>

                  <p className="text-gray-500 flex items-center hover:text-amber-600 transition-colors text-sm">
                    <HiMail className="mr-2 text-base text-amber-500" />
                    sweetbites@example.com
                  </p>

                  <p className="text-gray-500 flex items-center hover:text-amber-600 transition-colors text-sm">
                    <HiOutlineLocationMarker className="mr-2 text-base" />
                    Shop no. 1 virar west,gaothan road,
                  </p>
                </div>
              </div>
            </div>
          </div>
          {/* Footer Bottom */}
          <div className="container mx-auto mt-10 px-6 border-t border-gray-200 pt-6">
            <p className="text-center text-gray-500 text-sm">
```

```
                &copy; {new Date().getFullYear()} <span className="font-semibold text-amber-600">Sweet Bites</span>. All rights reserved.
              </p>
            </div>
          </footer>
    );
};
export default Footer;
```

Home.jsx

```
import React, { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import axios from 'axios';
import Hero from "../components/Layout/Hero";
import CakeCollectionSection from "../components/Products/CakeCollectionSection";
import Customized from "../components/Products/Customized";
import FeaturedCollection from "../components/Products/FeaturedCollection";
import ProductDetails from "../components/Products/ProductDetails";
import ProductGrid from "../components/Products/ProductGrid";
import FeaturesSection from "../components/Products/FeaturesSection";
import { fetchProductsByFilters } from '../redux/slices/productsSlice'; // Correct import path
const Home = () => {
  const dispatch = useDispatch();
  const { products, loading, error } = useSelector((state) => state.products);
  const [bestSellerProducts, setBestSellerProducts] = useState(null);
  const [bestSellerLoading, setBestSellerLoading] = useState(true);
  useEffect(() => {
    dispatch(fetchProductsByFilters({
      category: "Premium Sweets",
      limit: 8,
```

```
    sweetType: "classic",
  }));
  const fetchBestSellers = async () => {
    try {
 const response = await axios.get(`${import.meta.env.VITE_BACKEND_URL}/api/products/best-seller`);
      setBestSellerProducts(response.data);
    } catch (error) {
      console.error("Error fetching best seller products:", error);
    } finally {
      setBestSellerLoading(false);
    }
  };
  fetchBestSellers();
 }, [dispatch]);
 return (
  <div>
    <Hero />
    <div className="animate-slideUp">
      <CakeCollectionSection />
    </div>
    <div className="animate-slideUp animation-delay-200">
      <Customized />
      <h2 className="text-3xl text-center font-bold mb-4">
        Best Seller
      </h2>
      {bestSellerLoading ? (
        <p className='text-center'>Loading best sellers...</p>) : (
        bestSellerProducts ? (
          <ProductDetails productId={bestSellerProducts._id} />) : (
          <p className='text-center'>No best sellers found.</p>
        ))}
    </div>
```

```
      <div className="my-16 animate-slideUp animation-delay-300">
        <div className='container mx-auto'>
          <h2 className="text-3xl text-center font-bold mb-4">
            Explore Our Sweet Products
          </h2>
          <ProductGrid products={products.slice(0,8)} loading={loading} error={error} />
        </div>
      </div>
      <div className="animate-slideUp animation-delay-400">
        <FeaturedCollection />
      </div>
      <div className="animate-fadeIn animation-delay-600">
        <FeaturesSection />
      </div>
    </div>
  );
};
export default Home;
```

ProductGrid.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
const ProductGrid = ({ products, loading, error }) => {
  if (loading) return <p className="text-center py-10">Loading...</p>;
  if (error) return <p className="text-center py-10 text-red-500">Error: {error}</p>;
  return (
    <div className="grid grid-cols-2 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6">
      {products.map((product, index) => {
        const imageUrl = product.images?.[0]?.url || '/placeholder.png';
        const imageAlt = product.images?.[0]?.altText || product.name;
        return (
                <Link key={product._id || index} to={`/products/${product._id}`}
className="block">
```

```
        <div
            className="bg-white p-4 rounded-lg shadow hover:shadow-lg transition-all duration-300"
        style={{ animation: `fadeIn 0.5s ease forwards`, animationDelay: `${index * 0.1}s`, opacity: 0 }}>
            <div className="w-full h-40 sm:h-48 md:h-56 lg:h-64 mb-4 overflow-hidden rounded-lg">
            <img
              src={imageUrl}
              alt={imageAlt}
                 className="w-full h-full object-cover hover:scale-105 transition-transform duration-300"/>
          </div>
 <h3 className="text-base font-medium mb-1 line-clamp-1">{product.name}</h3>
          <p className="text-gray-600 font-semibold text-sm">₹{product.price}</p>
        </div>
       </Link>
      );
    })}
   </div>
  );
};
export default ProductGrid;
```

ProductsDetails.jsx

```
// Filename: ProductDetails.jsx
import React, { useState, useEffect } from "react";
import { toast } from "sonner";
import ProductGrid from "./ProductGrid";
import { useParams } from "react-router-dom";
import { useDispatch, useSelector } from "react-redux";
import { addToCart } from "../../redux/slices/cartSlice";
import {
  fetchProductDetails,
```

```
  fetchSimilarProducts,
} from "../../redux/slices/productsSlice";
import { motion } from "framer-motion";

const ProductDetails = ({ productId }) => {
  const { id } = useParams();
  const dispatch = useDispatch();

  const { selectedProduct, loading, error, similarProducts } = useSelector(
    (state) => state.products
  );
  const { user, guestId } = useSelector((state) => state.auth);

  const productFetchId = productId || id;

  const [selectedWeight, setSelectedWeight] = useState("");
  const [quantity, setQuantity] = useState(1);
  const [isButtonDisabled, setIsButtonDisabled] = useState(false);
  const [currentPrice, setCurrentPrice] = useState(0);

  useEffect(() => {
    if (productFetchId) {
      // ✅ FIX: Pass the ID string directly to fetchProductDetails
      dispatch(fetchProductDetails(productFetchId));
      // ✅ `fetchSimilarProducts` already correctly handles the object payload
      dispatch(fetchSimilarProducts({ id: productFetchId }));
    }
  }, [dispatch, productFetchId]);

  // Calculate price dynamically based on weight
  const calculatePrice = (basePricePerKg, weight) => {
    const weightNumber = parseFloat(weight);
    if (isNaN(weightNumber)) {
```

```
    return basePricePerKg;
  }

  let weightInKg;
  if (weight.toLowerCase().includes("kg")) {
    weightInKg = weightNumber;
  } else if (weight.toLowerCase().includes("g")) {
    weightInKg = weightNumber / 1000;
  } else {
    return basePricePerKg;
  }
  return basePricePerKg * weightInKg;
};

useEffect(() => {
  if (selectedProduct) {
    const basePrice = selectedProduct.basePrice || selectedProduct.price;
    if (selectedProduct.weights && selectedProduct.weights.length > 0) {
      const initialWeight = selectedProduct.weights[0];
      setSelectedWeight(initialWeight);
      const newPrice = calculatePrice(basePrice, initialWeight);
      setCurrentPrice(newPrice);
    } else {
      setSelectedWeight("");
    }
  }
}, [selectedProduct]);

useEffect(() => {
  if (selectedProduct && selectedWeight) {
    const basePrice = selectedProduct.basePrice || selectedProduct.price;
    const newPrice = calculatePrice(basePrice, selectedWeight);
    setCurrentPrice(newPrice);
```

```
  }
}, [selectedWeight, selectedProduct]);

// ✅ Limit quantity between 1 and 4
const handleQuantityChange = (action) => {
  if (action === "plus" && quantity < 4) {
    setQuantity((prev) => prev + 1);
  }
  if (action === "minus" && quantity > 1) {
    setQuantity((prev) => prev - 1);
  }
};

// ✅ Ensure cannot add more than 4 to cart
const handleAddToCart = () => {
  if (!selectedWeight) {
    toast.error("Please select a weight option.", { duration: 1000 });
    return;
  }
  if (!selectedProduct) {
    toast.error("Product not found.", { duration: 1000 });
    return;
  }
  if (quantity > 4) {
    toast.error("You can only add up to 4 units of this product.", { duration: 1000 });
    return;
  }

  setIsButtonDisabled(true);

  dispatch(
    addToCart({
      productId: productFetchId,
```

```
      quantity,
      weights: selectedWeight,
      guestId,
      userId: user?._id,
    })
  )
    .then(() => {
      toast.success("Added to cart!", { duration: 1000 });
    })
    .finally(() => {
      setIsButtonDisabled(false);
    });
};

if (loading || !selectedProduct) {
  return <p className="text-center">Loading product details...</p>;
}

if (error) {
  return <p className="text-center text-red-500">Error: {error}</p>;
}

const product = selectedProduct;

return (
  <motion.div
    className="p-6"
    initial={{ opacity: 0, y: 40 }}
    animate={{ opacity: 1, y: 0 }}
    transition={{ duration: 0.7, ease: "easeOut" }}
  >
    <div className="max-w-6xl mx-auto">
      <div className="flex flex-col md:flex-row gap-10">
```

```
        {/* Thumbnails */}
        <div className="hidden md:flex flex-col space-y-4 mr-6">
          {product.images?.map((image, index) => (
            <motion.img
              key={image.url || index}
              src={image.url}
              alt={image.altText || `Thumbnail ${index}`}
              whileHover={{ scale: 1.1, rotate: 2 }}
              transition={{ type: "spring", stiffness: 200 }}
                 className="w-20 h-20 object-cover rounded-xl cursor-pointer border border-
gray-200"
            />
          ))}
        </div>

        {/* Main Image */}
        <div className="md:w-1/2">
          <motion.div
            className="mb-4"
            animate={{ y: [0, -10, 0] }}
            transition={{ repeat: Infinity, duration: 4, ease: "easeInOut" }}
          >
            <motion.img
              src={product.images?.[0]?.url}
              alt="Main Product"
              initial={{ scale: 0.9, opacity: 0 }}
              animate={{ scale: 1, opacity: 1 }}
              transition={{ duration: 0.8 }}
              className="w-full h-[500px] object-cover rounded-2xl shadow-lg"
            />
          </motion.div>
        </div>
```

```
{/* Details */}
<motion.div
  className="md:w-1/2 md:ml-6"
  initial={{ opacity: 0, x: 50 }}
  animate={{ opacity: 1, x: 0 }}
  transition={{ delay: 0.3, duration: 0.7 }}
>
  <h1 className="text-3xl md:text-4xl font-extrabold text-gray-900 mb-3">
    {product.name}
  </h1>
  <p className="text-lg text-gray-600 mb-4">
    Category:{" "}
    <span className="font-medium text-gray-800">
      {product.category}
    </span>
  </p>
  <div className="flex items-center mb-6">
    <span className="text-2xl font-bold text-red-600">
      ₹{currentPrice.toFixed(2)}
    </span>
    {product.originalPrice && (
      <span className="text-sm line-through ml-3 text-gray-500">
        ₹{product.originalPrice}
      </span>
    )}
  </div>
  <p className="text-gray-700 leading-relaxed mb-8">
    {product.description}
  </p>

  {/* Weight Selection */}
  <div className="mb-6">
    <label
```

```
            htmlFor="weight"
            className="block text-sm font-semibold text-gray-800 mb-2"
          >
            Select Weight:
          </label>
          <motion.select
            whileFocus={{ scale: 1.02 }}
            id="weight"
                  className="border border-gray-300 rounded-xl p-3 w-full focus:ring-2
focus:ring-amber-400 focus:outline-none"
            value={selectedWeight}
            onChange={(e) => setSelectedWeight(e.target.value)}
          >
            <option value="">-- Please select --</option>
            {product.weights?.map((wt, index) => (
              <option key={index} value={wt}>
                {wt}
              </option>
            ))}
          </motion.select>
        </div>

        {/* Quantity */}
        <div className="mb-6">
          <p className="text-gray-800 font-medium">Quantity:</p>
          <div className="flex items-center space-x-4 mt-2">
            <motion.button
              whileTap={{ scale: 0.9 }}
              onClick={() => handleQuantityChange("minus")}
                 className="px-3 py-1 bg-gray-200 rounded-lg hover:bg-gray-300 transition
disabled:opacity-50"
              disabled={quantity === 1}
            >
```

```
            -
          </motion.button>
          <span className="text-lg font-semibold">{quantity}</span>
          <motion.button
            whileTap={{ scale: 0.9 }}
            onClick={() => handleQuantityChange("plus")}
            disabled={quantity === 4}
              className="px-3 py-1 bg-gray-200 rounded-lg hover:bg-gray-300 transition
disabled:opacity-50"
          >
            +
          </motion.button>
        </div>
        {quantity === 4 && (
          <p className="text-sm text-red-500 mt-1">
            Max 4 units allowed per product
          </p>
        )}
      </div>

      {/* Add to Cart */}
      <motion.button
        whileTap={{ scale: 0.95 }}
        whileHover={{ scale: 1.02 }}
        onClick={handleAddToCart}
        disabled={isButtonDisabled || !selectedWeight}
        className={`bg-amber-600 text-white font-semibold px-6 py-3 rounded-xl w-full
mb-6 shadow-md transition ${
          isButtonDisabled || !selectedWeight
            ? "cursor-not-allowed opacity-50"
            : "hover:bg-amber-700"
        }`}
      >
```

```
  {isButtonDisabled ? "Adding..." : "Add to Cart"}
</motion.button>

{/* Characteristics Section */}
<div className="mb-6">
  <h3 className="text-xl font-bold mb-3">Characteristics:</h3>
  <div className="flex flex-col space-y-2 text-gray-700">
    {product.category && (
      <div className="flex items-center">
        <span className="font-semibold w-24">Category:</span>
        <span className="ml-2">{product.category}</span>
      </div>
    )}
    {product.collections && (
      <div className="flex items-center">
        <span className="font-semibold w-24">Collections:</span>
        <span className="ml-2">{product.collections}</span>
      </div>
    )}
    {product.sweetType && (
      <div className="flex items-center">
        <span className="font-semibold w-24">Sweet:</span>
        <span className="ml-2">{product.sweetType}</span>
      </div>
    )}
    {product.types && (
      <div className="flex items-center">
        <span className="font-semibold w-24">Type:</span>
        <span className="ml-2">{product.types}</span>
      </div>
    )}
    {product.ingredients && (
      <div className="flex items-center">
```

```
            <span className="font-semibold w-24">Ingredients:</span>
            <span className="ml-2">
              {product.ingredients.join(", ")}
            </span>
          </div>
        )}
      </div>
    </div>
  </motion.div>
</div>

{/* Similar Products */}
<div className="mt-20">
  <h2 className="text-2xl text-center font-semibold mb-10">
    You May Also Like
  </h2>
  <motion.div
    initial="hidden"
    whileInView="visible"
    viewport={{ once: true }}
    variants={{
      hidden: { opacity: 0, y: 30 },
      visible: {
        opacity: 1,
        y: 0,
        transition: { staggerChildren: 0.2 },
      },
    }}
  >
    <ProductGrid products={similarProducts} />
  </motion.div>
</div>
</div>
```

```
    </motion.div>
  );
};

export default ProductDetails;
```

SortOptions.jsx

```
import React from 'react'
import { useSearchParams } from 'react-router-dom'
const SortOptions = () => {
  const [searchParams, setSearchParams] = useSearchParams();
  const handleSortChange = (e) => {
    const sortBy = e.target.value;
    searchParams.set("sortBy", sortBy)
    setSearchParams(searchParams)
  }
  return (
    <div className='mb-4 flex items-center justify-end'>
      <select id="sort"
      onChange={handleSortChange}
      value={searchParams.get("sortBy") || ""}
      className='border p-2 rounded-md focus:outline-none'>
        <option value="">Default</option>
        <option value="priceAsc">Price: Low to High</option>
        <option value="priceDsc">Price: Hight to Low</option>
        <option value="popularity">Popularity</option>
      </select>
    </div>
  )}
export default SortOptions;
```

FilterSideBar.jsx

```
import React, { useEffect, useState } from "react";
```

```
import { useSearchParams } from "react-router-dom";
const FilterSidebar = ({ isSidebarOpen, sidebarRef, toggleSidebar }) => {
  const [searchParams, setSearchParams] = useSearchParams();
  const [filters, setFilters] = useState({
    category: "",
    type: "",
    sweetType: "",
    collections: "",
    weight: "",
    customWeight: "",
    minPrice: 0,
    maxPrice: 1500,
  });
  const [priceRange, setPriceRange] = useState([0, 1500]);
  // Options
  const categories = ["Rajbhog", "Premium Sweets"];
  const types = ["Traditional", "Special", "Halwa"];
  const sweetTypes = ["classic", "ladoo", "barfi", "halwa", "mithai", "bengali"];
  const collections = [
    "Festival Specials",
    "Bengali Classics",
    "Milk Sweets",
    "Dryfruit Collection", ];
  const weights = ["250g", "500g", "1kg", "2kg"];
  // Sync state with URL params
  useEffect(() => {
    const params = Object.fromEntries([...searchParams]);
    setFilters({
      category: params.category || "",
      type: params.type || "",
      sweetType: params.sweetType || "",
      collections: params.collections || "",
      weight: params.weight || "",
```

```
    customWeight: params.customWeight || "",
    minPrice: Number(params.minPrice) || 0,
    maxPrice: Number(params.maxPrice) || 1500,
  });
  setPriceRange([0, Number(params.maxPrice) || 1500]);
}, [searchParams]);
// Handle filter changes
const handleFilterChange = (e) => {
  const { name, value } = e.target;
  const newFilters = { ...filters, [name]: value, customWeight: "" };
  setFilters(newFilters);
  updateURLParams(newFilters);
};
// Handle custom weight input
const handleCustomWeightChange = (e) => {
  const value = e.target.value;
  const newFilters = {
    ...filters,
    weight: "",
    customWeight: value && value >= 250 ? value : "",
  };
  setFilters(newFilters);
  updateURLParams(newFilters);
};
// Update URL params
const updateURLParams = (newFilters) => {
  const params = new URLSearchParams();
  Object.keys(newFilters).forEach((key) => {
    const value = newFilters[key];
    if (value !== "" && value !== 0 && value !== null) {
      params.append(key, value);
    }
  });
```

```
  setSearchParams(params, { replace: true });
};
const handlePriceChange = (e) => {
  const newPrice = e.target.value;
  setPriceRange([0, newPrice]);
  const newFilters = { ...filters, minPrice: 0, maxPrice: newPrice };
  setFilters(newFilters);
  updateURLParams(newFilters);
};
// Reset all filters
const handleResetFilters = () => {
  const newFilters = {
    category: "",
    type: "",
    sweetType: "",
    collections: "",
    weight: "",
    customWeight: "",
    minPrice: 0,
    maxPrice: 1500,
  };
  setFilters(newFilters);
  setPriceRange([0, 1500]);
  setSearchParams({});
};
return (
  <div className="h-full bg-white lg:shadow-lg lg:rounded-lg">
    {/* Header */}
    <div className="sticky top-0 bg-white z-10 px-6 py-4 border-b border-gray-200">
      <div className="flex justify-between items-center">
        <h3 className="text-lg font-semibold text-gray-900">Filter Sweets</h3>
        <button
          onClick={handleResetFilters}
```

```
          className="text-sm text-amber-600 hover:text-amber-700 font-medium transition-colors duration-200">
            Reset All
          </button>
        </div>
      </div>
      {/* Scrollable Content */}
      <div className="px-6 py-4 space-y-6 overflow-y-auto max-h-screen filter-sidebar-scrollbar">
        {/* Note: The max-h-screen will be overridden by the parent container */}
        {/* Category */}
        <FilterGroup
          label="Category"
          options={categories}
          selected={filters.category}
          name="category"
          onChange={handleFilterChange}/>
        {/* Types */}
        <FilterGroup
          label="Types"
          options={types}
          selected={filters.type}
          name="type"
          onChange={handleFilterChange}/>
        {/* Sweet Type */}
        <FilterGroup
          label="Sweet Type"
          options={sweetTypes}
          selected={filters.sweetType}
          name="sweetType"
          onChange={handleFilterChange}
          capitalize/>
        {/* Collections */}
```

```
        <FilterGroup
          label="Collections"
          options={collections}
          selected={filters.collections}
          name="collections"
          onChange={handleFilterChange}/>
        {/* Weights */}
        <FilterGroup
          label="Weight"
          options={weights}
          selected={filters.weight}
          name="weight"
          onChange={handleFilterChange}/>
        {/* Price Range */}
        <div className="space-y-3">
          <label
            htmlFor="priceRange"
            className="block text-sm font-semibold text-gray-800">
       Price Range: <span className="text-amber-600 font-bold">₹{priceRange[1]}</span>
          </label>
          <input
            type="range"
            id="priceRange"
            min={0}
            max={1500}
            value={priceRange[1]}
            onChange={handlePriceChange}
              className="w-full h-2 bg-gray-200 rounded-lg appearance-none cursor-pointer
slider"
            style={{
              background: `linear-gradient(to right, #f59e0b 0%, #f59e0b ${(priceRange[1] /
1500) * 100}%, #e5e7eb ${(priceRange[1] / 1500) * 100}%, #e5e7eb 100%)`}}/>
        <div className="flex justify-between text-xs text-gray-500">
```

```
            <span>₹0</span>
            <span>₹1500</span>
          </div>
        </div>
      </div>
      {/* Footer with Apply Filters Button */}
      <div className="sticky bottom-0 bg-white border-t border-gray-200 px-6 py-4">
        <button
          onClick={handleResetFilters}
          className="w-full px-4 py-3 bg-amber-600 text-white font-medium rounded-lg hover:bg-amber-700 focus:outline-none focus:ring-2 focus:ring-amber-500 focus:ring-offset-2 transition-colors duration-200 shadow-sm">
          Clear All Filters
        </button>
      </div>
    </div>
  );
};
// Reusable Filter Group Component
const FilterGroup = ({ label, options, selected, name, onChange, capitalize }) => (
  <div className="space-y-3">
    <label className="block text-sm font-semibold text-gray-800 uppercase tracking-wide">{label}</label>
    <div className="space-y-2.5">
      {options.map((opt) => (
        <label
          key={opt}
          htmlFor={`${name}-${opt}`}
          className="flex items-center space-x-3 cursor-pointer group py-1 px-2 rounded-md hover:bg-amber-50 transition-colors duration-150">
          <input
            type="radio"
            id={`${name}-${opt}`}
            name={name}
```

```
          value={opt}
          checked={selected === opt}
          onChange={onChange}
             className="h-4 w-4 text-amber-600 focus:ring-amber-500 focus:ring-offset-0 border-gray-300 cursor-pointer"/>
        <span className="text-sm text-gray-700 group-hover:text-gray-900 transition-colors duration-150 select-none">
          {capitalize ? opt.charAt(0).toUpperCase() + opt.slice(1) : opt}
         </span>
       </label>
     ))}
   </div>
  </div>
);
export default FilterSidebar;
```

Collection.jsx

```
import React, { useEffect, useRef, useState } from 'react';
import { FaFilter } from "react-icons/fa";
import FilterSidebar from '../components/Products/FilterSidebar';
import SortOptions from '../components/Products/SortOptions';
import ProductGrid from '../components/Products/ProductGrid';
import Pagination from '../components/Products/Pagination';
import { useParams, useSearchParams, useNavigate } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { fetchProductsByFilters } from '../redux/slices/productsSlice';
const createPaginationUrl = (params, page) => {
   const newParams = { ...params, page: page.toString() };
   const searchString = new URLSearchParams(newParams).toString();
   return `?${searchString}`;
};
const CollectionPage = () => {
   const { collection } = useParams();
```

```
  const [searchParams, setSearchParams] = useSearchParams();
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { products, loading, error } = useSelector((state) => state.products);
  const queryParams = Object.fromEntries([...searchParams]);
  const sidebarRef = useRef(null);
  const [isSidebarOpen, setIsSidebarOpen] = useState(false);
  // Pagination constants
  const productsPerPage = 48; // Set products per page
  const currentPage = parseInt(queryParams.page) || 1;
   const totalProducts = products.length; // Assumes all products are loaded, though for better performance you would get this from the API
  const indexOfLastProduct = currentPage * productsPerPage;
  const indexOfFirstProduct = indexOfLastProduct - productsPerPage;
  const currentProducts = products.slice(indexOfFirstProduct, indexOfLastProduct);
  useEffect(() => {
    // Fetch products, including filters and pagination info
    dispatch(fetchProductsByFilters({ collection, ...queryParams }));
  }, [dispatch, collection, searchParams]);
  const toggleSidebar = () => {
    setIsSidebarOpen(!isSidebarOpen);
  };
  const handleClickOutside = (e) => {
    if (sidebarRef.current && !sidebarRef.current.contains(e.target)) {
      setIsSidebarOpen(false);
    }};
  useEffect(() => {
    document.addEventListener("mousedown", handleClickOutside);
    return () => {
      document.removeEventListener("mousedown", handleClickOutside);
    };
  }, []);
  const pageTitle = collection
```

```
      ? `${collection.charAt(0).toUpperCase() + collection.slice(1)} Collection`
      : 'All Products';
  const handlePageChange = (pageNumber) => {
    const newUrl = createPaginationUrl(queryParams, pageNumber);
    setSearchParams(new URLSearchParams(newUrl));
    window.scrollTo({ top: 0, behavior: 'smooth' });
  };
  return (
    <div className="min-h-screen bg-gray-50">
      <div className="flex flex-col lg:flex-row">
        {/* Mobile filter button */}
        <button
          onClick={toggleSidebar}
          className="lg:hidden mx-4 my-4 p-3 flex justify-center items-center bg-white text-gray-700 rounded-lg shadow-sm border border-gray-200 hover:bg-gray-50 transition-colors duration-200">
          <FaFilter className="mr-2" />
          Filter Products
        </button>
        {/* Filter Sidebar */}
        <div
          ref={sidebarRef}
          className={`${isSidebarOpen ? "translate-x-0" : "-translate-x-full"}
            fixed inset-y-0 z-50 left-0 w-80 bg-white shadow-xl transition-transform duration-300
            lg:static lg:translate-x-0 lg:w-72 lg:min-h-screen lg:shadow-lg lg:border-r lg:border-gray-200`}>
          <FilterSidebar />
        </div>
        {/* Main Content */}
        <div className="flex-1 bg-white lg:bg-gray-50">
          <div className="max-w-7xl mx-auto p-6 lg:p-8">
            <div className="mb-6">
              <h1 className="text-3xl font-bold text-gray-900 mb-2">
```

```
                    {pageTitle}
                </h1>
                    <p className="text-gray-600">Discover our handcrafted sweet collections</p>
            </div>

             <div className="flex flex-col sm:flex-row sm:justify-between sm:items-center mb-6 gap-4">
                <div className="text-sm text-gray-600">
                    {products?.length > 0 && (
                        <span>{totalProducts} product{totalProducts !== 1 ? 's' : ''} found</span>
                    )}
                </div>
                <SortOptions />
            </div>
          <div className="bg-white lg:bg-transparent lg:rounded-lg lg:shadow-sm lg:p-6">
          <ProductGrid products={currentProducts} loading={loading} error={error} />
            </div>
            {/* Pagination component */}
            {totalProducts > productsPerPage && (
              <div className="mt-8 flex justify-center">
                <Pagination
                  currentPage={currentPage}
                  totalProducts={totalProducts}
                  productsPerPage={productsPerPage}
                  onPageChange={handlePageChange}/>
              </div>
            )}
          </div>
        </div>
      </div>
```

```
    )};
export default CollectionPage;
```

Login.jsx

```
import React from 'react'
import { useState } from 'react';
import { Link } from 'react-router-dom';
import login from "../assets/login.png";
import { loginUser } from '../redux/slices/authSlice';
import { useDispatch } from 'react-redux';
import { useNavigate, useLocation } from 'react-router-dom';
import { useEffect } from 'react';
import { useSelector } from 'react-redux';
import { mergeCart } from '../redux/slices/cartSlice';
const Login = () => {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const dispatch = useDispatch();
    const navigate = useNavigate();
    const location = useLocation();
    const {user, guestId, loading } = useSelector((state) => state.auth);
    const {cart} = useSelector((state) => state.cart);
    // get redirect parameter and check if its chechkout or something
    const redirect = new URLSearchParams(location.search).get("redirect") || "/";
    const isCheckoutRedirect = redirect.includes("checkout");
    useEffect(() => {
        if(user){
            if (cart?.products.length > 0 && guestId) {
                dispatch(mergeCart({guestId, user})).then(() => {
                    navigate(isCheckoutRedirect ? "/checkout" : "/");
                });
            } else {
                navigate(isCheckoutRedirect ? "/checkout" : "/");
            }}
```

```
  }, [user, navigate, isCheckoutRedirect, cart, guestId, dispatch]);
    const handleSubmit = (e) => {
    e.preventDefault();
    dispatch(loginUser({email,password}));
  };
 return <div className='flex'>
  <div className='w-full md:w-1/2 flex-col justify-center items-center p-8 md:p-12'>
  <form onSubmit={handleSubmit}
   className='w-full max-w-md bg-white p-8 rounded-lg border shadow-sm lg:ml-40 lg:mt-9'>
    <div className='flex justify-center mb-6 '>
      <h2 className='text-xl font-medium '>CreamyCo</h2>
    </div>
    <h2 className='text-2xl font-bold text-center mb-6 '>Hey there! 👋</h2>
    <p className='text-center mb-6'>
      Enter Your username and password to Login
    </p>
    <div className='mb-4'>
      <label className='block text-sm font-semibold mb-2'>Email</label>
      <input
      type="email"
      value={email}
      onChange={(e) => setEmail(e.target.value)}
      className='w-full p-2 border rounded'
      placeholder='Enter your email address'/>
    </div>
    <div className='mb-4 '>
      <label className='block text-sm font-semibold mb-2'>Password</label>
      <input type="password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      className='w-full p-2 border rounded'
      placeholder='Enter your password'/>
```

```
        </div>
         <button type="submit" className='w-full bg-black text-white p-2 rounded-lg font-
semibold hover:bg-gray-800
        transition'>{loading ? "loading...": "Sign In"}</button>
        <p className='mt-6 text-center text-sm'>Don't have an account ?
           <Link to ={`/register?redirect=${encodeURIComponent(redirect)}`}
           className="text-blue-500">
           Register</Link>
        </p>
    </form>
    </div>
    <div className='hidden md:block w-1/2 bg-gray-800 '>
    <div className='h-full felx flex-col justify-center items-center'>
      <img src={login} alt="login to Account" className='h-[650px] w-full object-cover />
    </div>
    </div>
  </div>}
export default Login;
```

CartContent.jsx

```
import React from 'react'
import { RiDeleteBin6Line } from "react-icons/ri";
import { useDispatch } from 'react-redux';
import { removeFromCart, updateCartItemQuantity } from '../../redux/slices/cartSlice';

const CartContents = ({ cart, userId, guestId }) => {
  const dispatch = useDispatch();

  // handle adding or subtracting to cart
  const handleAddToCart = (productId, delta, quantity, weights) => {
    const newQuantity = quantity + delta;

    // Restrict quantity between 1 and 4
```

```
    if (newQuantity >= 1 && newQuantity <= 4) {
      const id = typeof productId === 'object' ? productId._id : productId;
      dispatch(
        updateCartItemQuantity({
          productId: id,
          quantity: newQuantity,
          guestId,
          userId,
          weights,
        })
      );
    }
  };

  const handleRemoveFromCart = (productId, weights) => {
    const id = typeof productId === 'object' ? productId._id : productId;
    dispatch(removeFromCart({ productId: id, guestId, userId, weights }));
  };

  return (
    <div>
      {cart.products.map((product, index) => (
        <div
                key={`${product.productId?._id   ||   product.productId}-${product.weights}-
${index}`}
          className="flex items-start justify-between py-4 border-b"
        >
          <div className="flex items-start">
            <img
              src={product.image}
              alt={product.name}
              className="w-20 h-20 object-cover rounded ml-2 mr-4 mt-2"
            />
```

```
<div>
  <h3 className="font-semi-bold text-lg">{product.name}</h3>
  <p className="text-sm text-gray-500">
    Weight: {product.weights} | Quantity: {product.quantity}
  </p>
  <div className="flex items-center mt-2">
    <button
      onClick={() =>
        handleAddToCart(
          product.productId,
          -1,
          product.quantity,
          product.weights
        )
      }
      disabled={product.quantity <= 1}
      className="border rounded px-2 py-1 text-xl font-medium disabled:opacity-50 disabled:cursor-not-allowed"
    >
      -
    </button>
    <span className="mx-2">{product.quantity}</span>
    <button
      onClick={() =>
        handleAddToCart(
          product.productId,
          1,
          product.quantity,
          product.weights
        )
      }
      disabled={product.quantity >= 4} // disable at 4
```

```
              className="border rounded px-2 py-1 text-xl font-medium disabled:opacity-50
disabled:cursor-not-allowed"
            >
              +
            </button>
          </div>
        </div>
      </div>
      <div>
        <p className="mr-4 mt-7">₹ {product.price * product.quantity}</p>
        <button
          onClick={() =>
            handleRemoveFromCart(product.productId, product.weights)
          }
          className="mt-2 ml-auto block"
        >
          <RiDeleteBin6Line className="h-6 w-6 mr-4 mt-1 text-gray-600 hover:text-red-
500" />
        </button>
      </div>
    </div>
  ))}
 </div>
 );
};

export default CartContents;
```

CartDrawer.jsx

```
import React from 'react'
import { IoMdClose } from "react-icons/io";
import CartContents from '../Cart/CartContents';
import { useNavigate } from 'react-router-dom';
```

```
import { useSelector } from 'react-redux';

const CartDrawer = ({drawerOpen, toggleCartDrawer}) => {
  const navigate = useNavigate();
  const {user, guestId} = useSelector((state) => state.auth);
  const {cart} = useSelector((state) => state.cart);
  const userId = user ? user._id : null;

  const handleCheckout = () => {
    toggleCartDrawer();
    if(!user) {
        navigate("/login?redirect=checkout");
    } else {
        navigate("/checkout");
    }
  };

  return (
    <>
      {/* Overlay */}
      <div
       onClick={toggleCartDrawer}
         className={`fixed inset-0 bg-black/30 backdrop-blur-sm transition-opacity z-40 ${drawerOpen ? "opacity-100" : "opacity-0 pointer-events-none"}`}
      ></div>

      {/* Cart Drawer */}
      <div className={`fixed top-0 right-0 w-3/4 sm:w-1/2 md:w-[25rem] h-full bg-white shadow-lg transform
        transition-transform duration-300 flex flex-col z-50 ${drawerOpen ? "translate-x-0 animate-fadeIn" : "translate-x-full"}`}>
        {/* Close Button */}
        <div className="flex justify-between items-center p-4 border-b border-gray-100">
```

```
        <h2 className="text-xl font-semibold text-amber-600 animate-fadeIn">Your Sweet Cart</h2>
        <button
          onClick={toggleCartDrawer}
           className="text-gray-600 hover:text-amber-600 hover:bg-gray-100 p-2 rounded-full transition-colors duration-200"
        >
          <IoMdClose className="h-6 w-6" />
        </button>
      </div>

      {/* Cart Items */}
      <div className="flex-grow overflow-y-auto p-2 animate-slideUp">
        {cart && cart?.products?.length > 0 ? (<CartContents cart={cart} userId={userId} guestId = {guestId}/>) : (
          <p>Your Cart is empty</p>
        )}
        {/* Component for Cart Contents */}
      </div>

      {/* Checkout Button Fixed At Bottom */}
        <div className="p-4 bg-white sticky bottom-0 border-t border-gray-100 shadow-inner">
        {cart && cart?.products?.length > 0 && (
          <>
          <button
          onClick={handleCheckout}
             className='w-full bg-amber-600 text-white py-3 rounded-full font-semibold hover:bg-amber-700 transition-colors duration-200 shadow-md'
        >
          Proceed to Checkout
        </button>
        <p className='text-sm tracking-tighter text-gray-500 mt-3 text-center'>
          Shipping, taxes, and discount codes calculated at checkout
```

```
          </p>
            </>
          )}
        </div>
      </div>
    </>
  );
};

export default CartDrawer;
```

Admin.jsx

```
import React, { useState } from 'react'
import { FaBars } from "react-icons/fa";
import AdminSidebar from "../Admin/AdminSidebar";
import { Outlet } from 'react-router-dom';
const AdminLayout = () => {
  const [isSideBarOpen, setIsSideBarOpen] = useState(false);
  const toggleSideBar = () => {
    setIsSideBarOpen(!isSideBarOpen);
  };
    return (
    <div className='min-h-screen flex flex-col md:flex-row relative'>
      {/* Mobile Toggle Button */}
      <div className="flex md:hidden p-4 bg-gray-900 text-white z-20">
        <button onClick={toggleSideBar}>
          <FaBars size={24} />
        </button>
        <h1 className="ml-4 text-xl font-medium">Admin Dashboard</h1>
      </div>
      {/* Overlay for mobile SideBar */}
      {isSideBarOpen && (
        <div
        className="fixed inset-0 z-10 bg-black bg-opacity-50 md:hidden"
```

```
        onClick={toggleSideBar}></div>
      )}
      {/* Sidebar */}
        <div className={`bg-gray-900 w-64 min-h-screen text-white absolute md:relative transform
        ${isSideBarOpen ? "translate-x-0 " : "-translate-x-full"}
        transition-transform duration-300 md:translate-x-0 md:static md:block z-20`}>
          {/* SideBar */}
          <AdminSidebar />
        </div>
        {/* Main Content */}
        <div className="flex-grow p-6 overflow-auto">
          <Outlet />
        </div>
    </div>
  )}
export default AdminLayout
```

Backend

Server.js

```
const express = require("express");
const cors = require("cors");
const dotenv = require("dotenv");
const path = require("path");
const connectDB = require("./config/db");
const userRoutes = require("./routes/userRoutes");
const productRoutes = require("./routes/productRoutes");
const cartRoutes = require("./routes/cartRoutes");
const checkoutRoutes = require("./routes/checkoutRoutes");
const orderRoutes = require("./routes/orderRoutes");
const addressRoutes = require("./routes/addressRoutes");
const uploadRoutes = require("./routes/uploadRoutes");
```

```
const subscribeRoute = require("./routes/subscribeRoute");
const contactRoutes = require("./routes/contactRoutes");
const adminRoutes = require("./routes/adminRoutes");
const productAdminRoutes = require("./routes/productAdminRoutes");
const adminOrderRoutes = require("./routes/adminOrderRoutes");
dotenv.config();
const app = express();
app.use(express.json());
const allowedOrigins = [
  "http://localhost:5173",
  "https://sweet-bites-lsfx.vercel.app",
];
app.use(
  cors({
    origin: function (origin, callback) {
      if (!origin || allowedOrigins.includes(origin) || !origin) {
        callback(null, true);
      } else {
        callback(new Error("Not allowed by CORS"));
      }
    },
    credentials: true,
  }));
app.use("/uploads", express.static(path.join(__dirname, "/uploads")));
connectDB();
const PORT = process.env.PORT || 3000;
app.get("/", (req, res) => {
  res.send("WELCOME TO CREAMYCAKE&CO API!");
});
app.use("/api/users", userRoutes);
app.use("/api/products", productRoutes);
app.use("/api/cart", cartRoutes);
app.use("/api/checkout", checkoutRoutes);
```

```
app.use("/api/orders", orderRoutes);
app.use("/api/addresses", addressRoutes);
app.use("/api/upload", uploadRoutes);
app.use("/api", subscribeRoute);
app.use("/api/contact", contactRoutes);
app.use("/api/admin/users", adminRoutes);
app.use("/api/admin/products", productAdminRoutes);
app.use("/api/admin/orders", adminOrderRoutes);
app.listen(PORT, () => {
  console.log(`✅ Server running at http://localhost:${PORT}`);
});
```

AuthMiddleware.js

```
const jwt = require("jsonwebtoken");
const User = require("../models/User");
const protect = async (req, res, next) => {
  let token;
if (
    req.headers.authorization &&
    req.headers.authorization.startsWith("Bearer") ) {
    try {
      token = req.headers.authorization.split(" ")[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = await User.findById(decoded.user.id).select("-password"); // Exclude
password
      next();
    } catch (error) {
      console.error("Token Verification failed:", error.message);
      res.status(401).json({ message: "Not authorized, token failed" });
    }
  } else {
    res.status(401).json({ message: "Not authorized, no token provided" });
  }};
```

```
const admin = (req, res, next) => {
  if (req.user && req.user.role === "admin") {
    next();
  } else {
    res.status(403).json({ message: "Not authorized as an admin" });
  }};
module.exports = { protect,admin };
```

Models

Product.js

```
const mongoose = require("mongoose");
const productSchema = mongoose.Schema(
  {
    name: { type: String, required: true },
    description: { type: String },
    longDescription: { type: String },
    price: { type: Number, required: true },
    originalPrice: { type: Number },
    discountPrice: { type: Number },
    discount: { type: Number },
    countInStock: { type: Number, default: 0 },
    category: { type: String },
    weights: [String],
    collections: { type: String },
    sweetType: { type: String },
    types: { type: String },
    images: [{ url: { type: String, required: false }, altText: { type: String },
      }, ],
    availability: { type: Boolean, default: true },
    stock: { type: Number },
    ingredients: [String],
    tags: [String],
    rating: { type: Number, default: 0 },
```

```
    reviewCount: { type: Number, default: 0 },
    isFeatured: { type: Boolean, default: false },
    isPublished: { type: Boolean, default: true },
    sku: { type: String },
    sold: { type: Number, default: 0 }, // ✅ NEW FIELD for best-seller logic
    user: { type: mongoose.Schema.Types.ObjectId, ref: "User" }, },
  { timestamps: true }
);
module.exports = mongoose.model("Product", productSchema);
```

User.js

```
const mongoose = require("mongoose");
const bcrypt = require("bcryptjs");
const userSchema = new mongoose.Schema({
        name: {type: String, required: true,trim: true,
        },
        email: {type: String,required: true,unique: true,trim: true,
           match: [/.+\@.+\..+/, "Please enter a valid email address"],
        },
        password: {type: String,required: true,minLength: 6,
        },
        phone: {type: String,trim: true,
        },
        role: {type: String,default: "customer",
        },
},
{timestamps: true }
);
userSchema.pre("save", async function(next) {
    if (!this.isModified("password")) return next();
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(this.password, salt);
    next();
```

```
});
userSchema.methods.matchPassword = async function (enteredPassword) {
   return await bcrypt.compare(enteredPassword, this.password);
};
module.exports = mongoose.model("User", userSchema);
```

Routes

UserRoute.js

```
const express = require("express");
const User = require("../models/User");
const Address = require("../models/Address");
const jwt = require("jsonwebtoken");
const {protect} = require("../middleware/authMiddleware");
const router = express.Router();
router.post("/register", async (req, res) => {
   const {name, email, password} = req.body;
   try {
      // Registration Logic
      let user = await User.findOne({email});
      if (user) return res.status(400).json({message: "User already exists"});
      user = new User({name, email, password });
      await user.save();
      //  Create JWT Payload
      const payload = {user: {id: user._id, role: user.role} };
      //  sign and return the token along with the user data
      jwt.sign(payload, process.env.JWT_SECRET, {expiresIn: "40h"}, (err, token) => {
         if(err) throw err;
         // send the user and token in response
         res.status(201).json({
            user: {
               _id: user._id,
               name: user.name,
               email: user.email,
```

```
                role: user.role,
            },
            token,
        })});
    } catch (error) {
        console.log(error);
        res.status(500).send("Server Error");
    }
});
router.post("/login", async (req, res) =>{
    const {email,password}= req.body;
    try {
        let user = await User.findOne({email});
        if(!user) return res.status(400).json({message: "Invalid Credentails"});
        const isMatch = await user.matchPassword(password);
        if(!isMatch) return res.status(400).json({message: "Invalid Credentails"});
        const payload = {user: {id: user._id, role: user.role} };
        jwt.sign(payload, process.env.JWT_SECRET, {expiresIn: "40h"}, (err, token) => {
            if(err) throw err;
            res.json({
                user: {
                    _id: user._id,
                    name: user.name,
                    email: user.email,
                    role: user.role,
                },
                token,
            });},);
    } catch (error) {
        console.log(error);
        res.status(500).send("Server Error");
    }});
router.get("/profile",protect, async (req,res) => {
```

```
    res.json(req.user);
});
router.put("/profile", protect, async (req, res) => {
    try {
        const { name, email, phone, password } = req.body;
        const user = await User.findById(req.user._id);
        if (!user) {
            return res.status(404).json({ message: "User not found" });
        }
        if (email && email !== user.email) {
            const existingUser = await User.findOne({ email });
            if (existingUser) {
                return res.status(400).json({ message: "Email already in use" });
            } }
        user.name = name || user.name;
        user.email = email || user.email;
        user.phone = phone || user.phone;
        if (password) {
            user.password = password;
        }
        const updatedUser = await user.save();
        res.json({
            _id: updatedUser._id,
            name: updatedUser.name,
            email: updatedUser.email,
            phone: updatedUser.phone,
            role: updatedUser.role,
        });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: "Server Error" });
    }});
router.delete("/profile", protect, async (req, res) => {
```

```
    try {
        // Delete user's addresses
        await Address.deleteMany({ user: req.user._id });
        await User.findByIdAndDelete(req.user._id);
        res.json({ message: "Account deleted successfully" });
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: "Server Error" });
    }});
module.exports = router;
```

AdminRoute.js

```
const express = require("express");
const User = require("../models/User");
const { protect , admin} = require("../middleware/authMiddleware");
const router = express.Router();
router.get("/", protect, admin, async(req, res) => {
    try {
        const users = await User.find({});
            res.json(users);
    } catch (error) {
        console.error(error);
        res.status(500).json({message: "Server Error"});
    }});
router.post("/", protect, admin, async (req, res) => {
  const { name, email, password, role } = req.body;
  try {
    let user = await User.findOne({ email });
    if (user) {
      return res.status(400).json({ message: "User already exists" });
    }
    user = new User({
      name,
      email,
```

```
      password,
      role: role || "customer",
    });
    await user.save();
    const { password: _, ...userData } = user.toObject();
    res.status(201).json({
      message: "User created successfully",
      user: userData,
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Server Error" });
  }});
router.put("/:id", protect, admin, async(req , res) => {
    try {
        const user = await User.findById(req.params.id);
        if(user) {
            user.name = req.body.name || user.name;
            user.email = req.body.email || user.email;
            user.role = req.body.role || user.role;
        }
        const updatedUser = await user.save();
        res.json({message: "User updated successfully", user : updatedUser});
    } catch (error) {
        console.error(error);
        res.status(500).json({message: "Server Error"});
    }});
router.delete("/:id", protect,admin, async (req , res) => {
    try {
        const user = await User.findById(req.params.id);
        if(user){
            await user.deleteOne();
            res.json({message:"User deleted successfully"});
```

```
        } else {
            res.status(404).json({message: "User not found"});
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({message: "Server Error"});
    }
});
module.exports = router;
```

6.4 TESTING APPROACH

The testing approach for the **Sweets Shop Web Application** is a structured process that ensures the system fulfills both functional and non-functional requirements, operates reliably, and delivers a smooth user experience. Various testing methods are applied, ranging from unit testing to system-level validation, to ensure that each module works correctly on its own and in combination with others.

Unit Testing

Unit testing is performed to validate the functionality of individual components within the sweets shop application. Features such as user authentication, product browsing, cart management, and order placement are tested independently to ensure they function as intended.

Integration Testing:

Integration testing ensures that different modules—such as the frontend, backend, database, and payment gateway—work together seamlessly. For instance, the interaction between the shopping cart and the Razorpay payment system is verified to ensure correct order processing and stock updates.

Functional Testing:

Functional testing validates whether the application's features meet specified requirements. Core functionalities like browsing sweets, adding products to the cart, applying quantity changes, placing orders, and making secure payments are tested against expected user flows.

Performance Testing:

Performance testing is carried out to assess how the system behaves under different loads. It ensures that the sweets shop can handle multiple concurrent users, process several orders simultaneously, and continue to function efficiently during festive or peak shopping periods.

Security Testing:

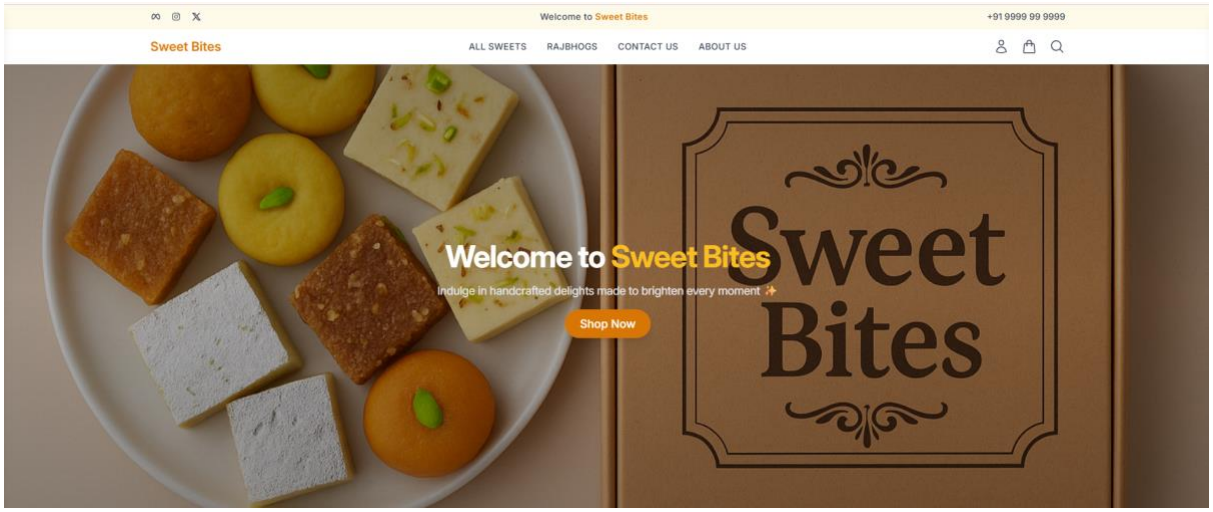
Given the handling of sensitive customer data such as personal information and payment details, the application undergoes rigorous security testing. This includes checking for vulnerabilities, preventing unauthorized access, and validating the security of the Razorpay payment gateway to safeguard user transactions.

6.4.1 Test Cases

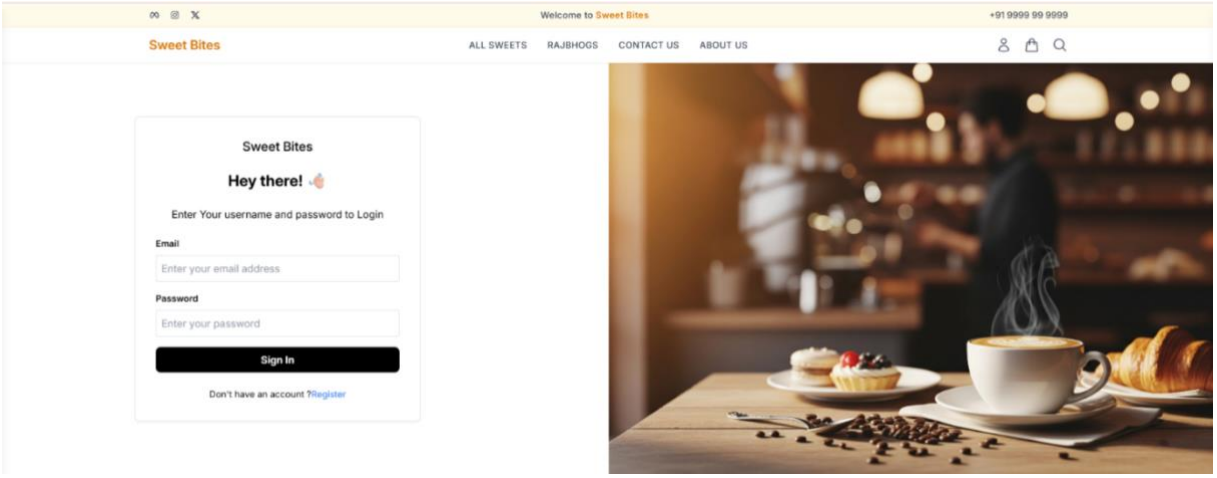
Test Case ID	Case	Action	Expected Result	Actual Result	Status
1	Login	Checking the login credentials of user	Successful login	Successful login	Pass
2	Edit or Delete Sweets	Admin edits or deletes sweets	Sweets successfully added or deleted	Sweets successfully added or deleted	Pass
3	View Sweets	User browses products	User able to browse and view sweets	User able to browse and view sweets	Pass
4	Add to Cart	User adds a sweet to the cart	Sweet added to shopping cart	Sweet added to shopping cart	Pass
5	Remove from Cart	User removes a sweet from the cart	Sweet removed from shopping cart	Sweet removed from shopping cart	Pass
6	Payment Processing	User makes a payment via Razorpay	Payment completed successfully	Payment completed successfully	Pass
7	View Order History	User checks previous orders	Order history displayed correctly	Order history displayed correctly	Pass
8	Logout	Logging out of the system	Successful logout from the web application	Successful logout from the web application	Pass

CHAPTER 7. RESULTS

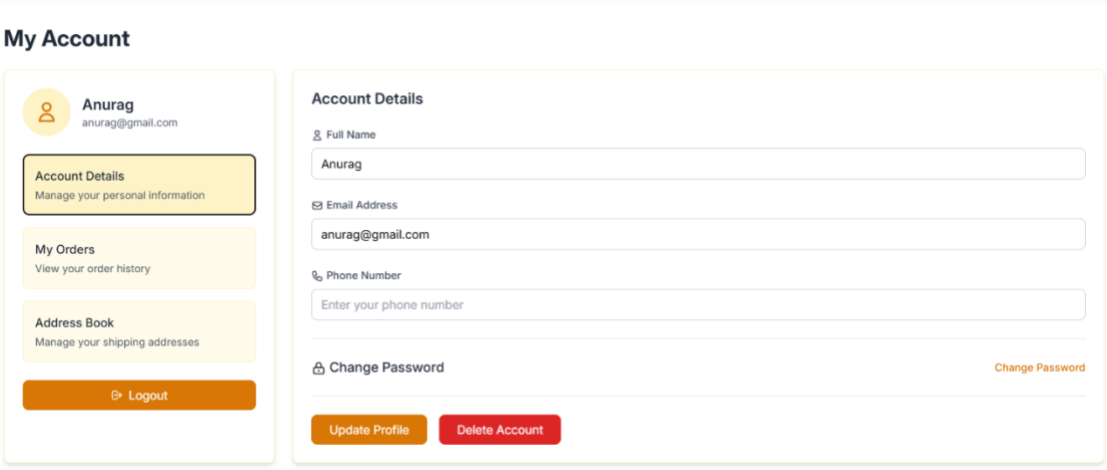
Homepage



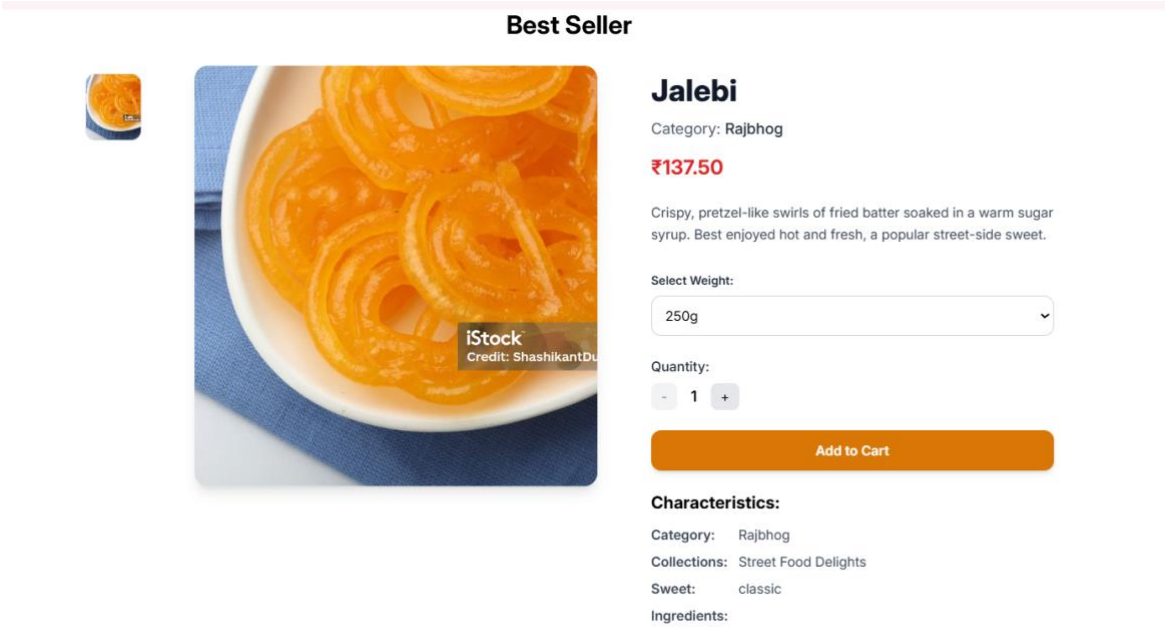
Login Page



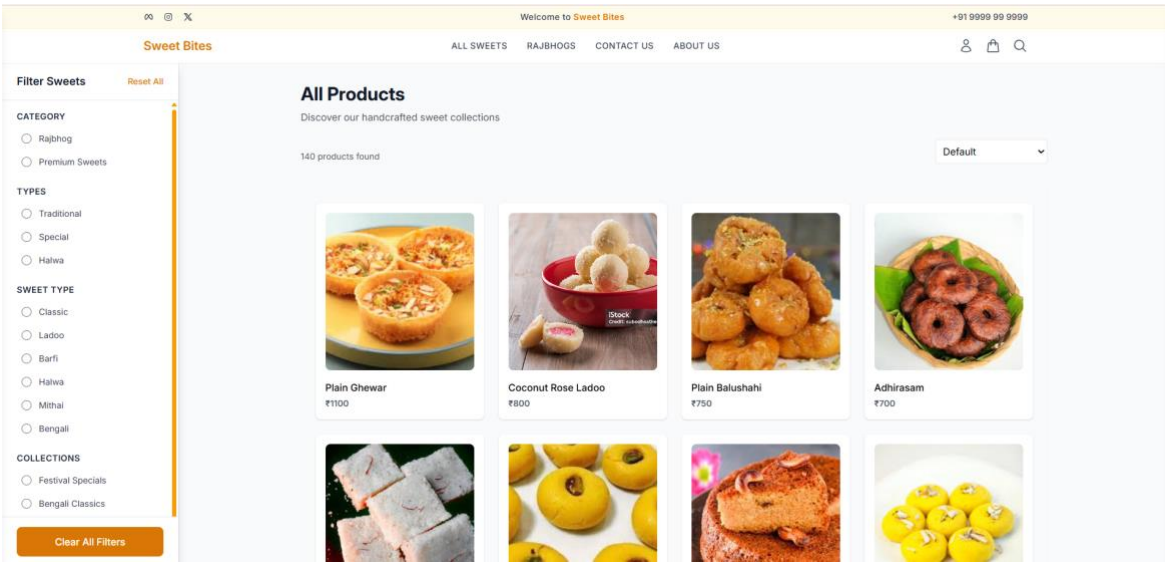
Successfully User Logged In And My Profile Page



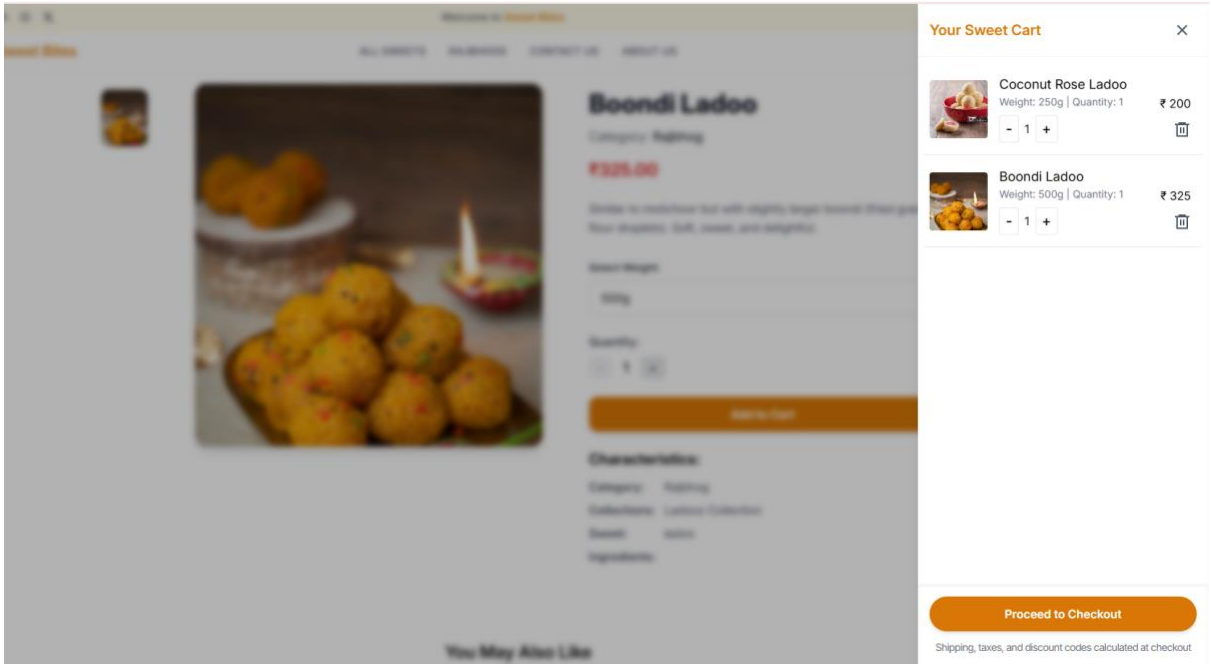
BestSeller Section



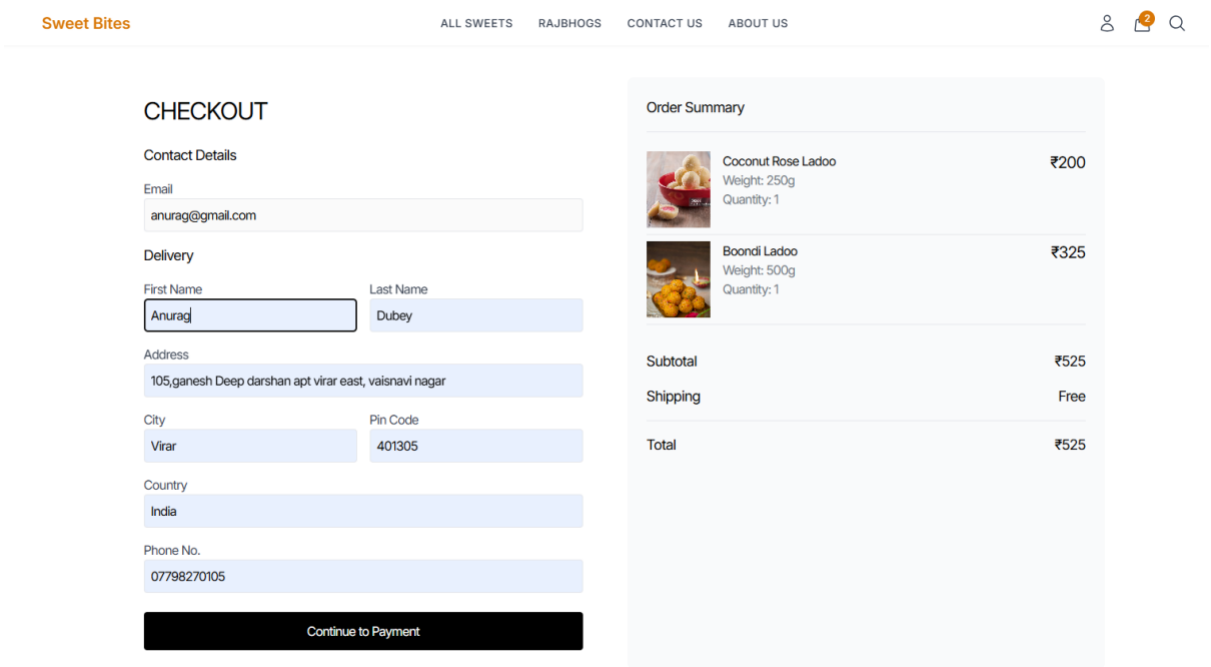
All Sweets Section With FilterSideBar



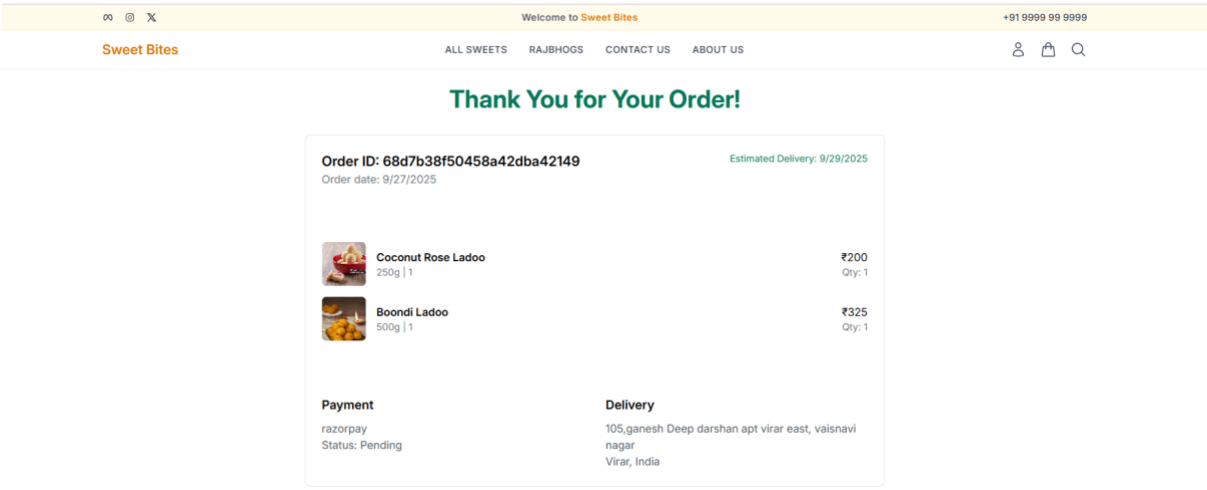
CartDrawer Section



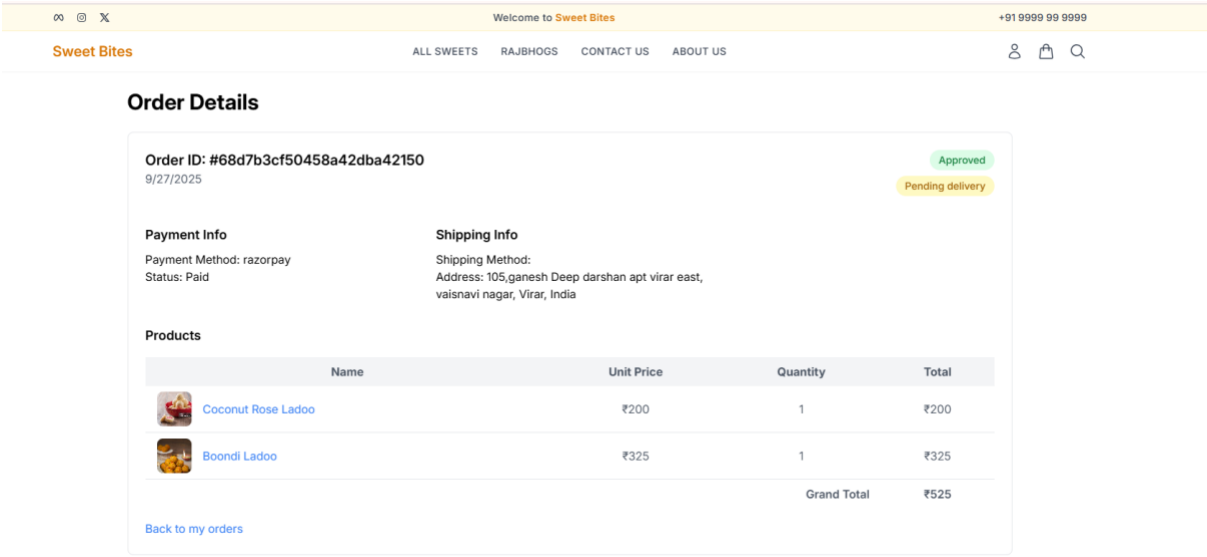
CheckoutPage



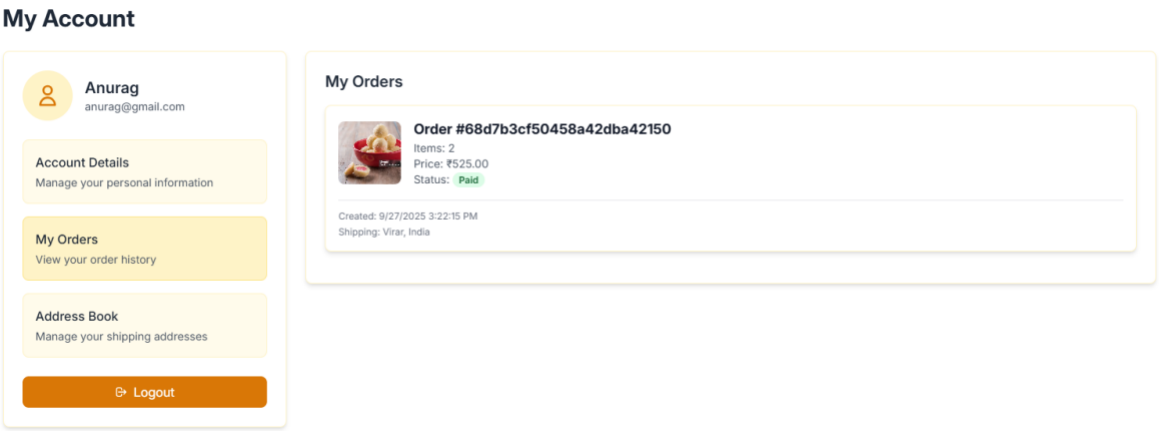
Order Placed Successfully



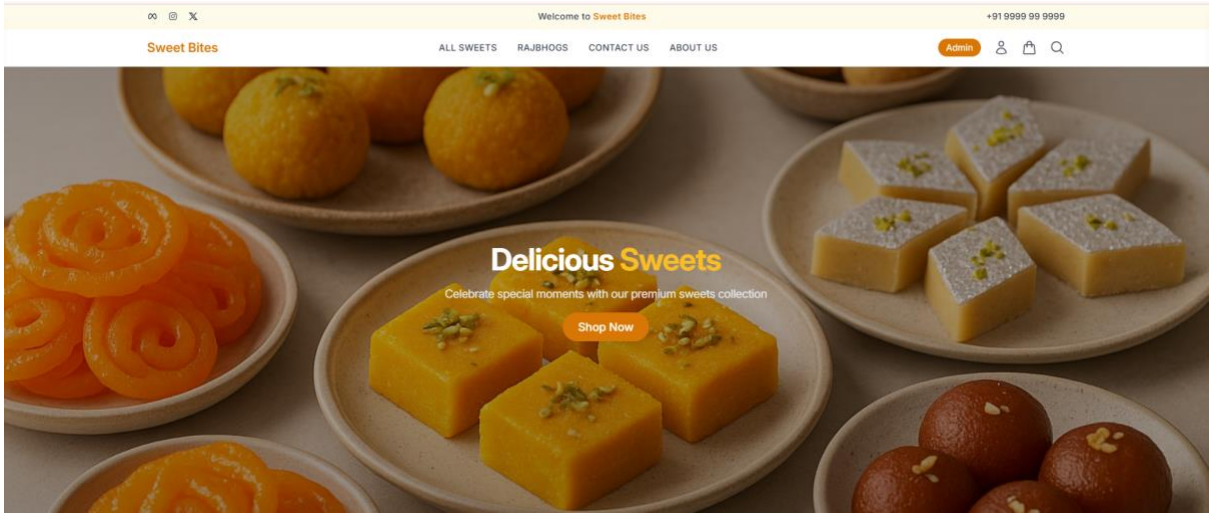
Order Details Page



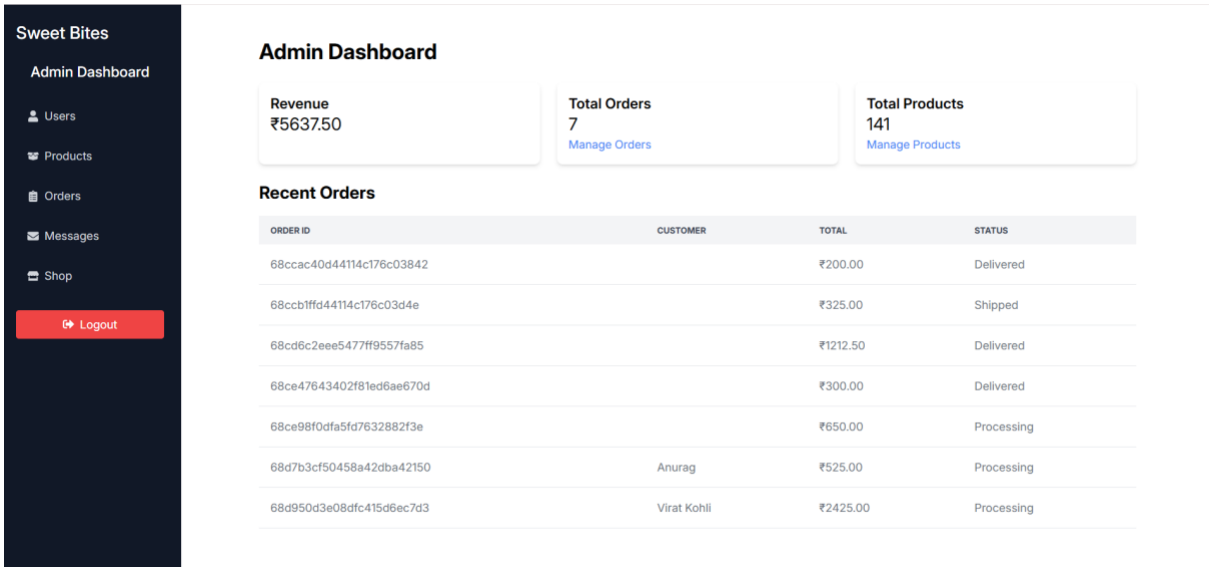
Updated ProfilePage



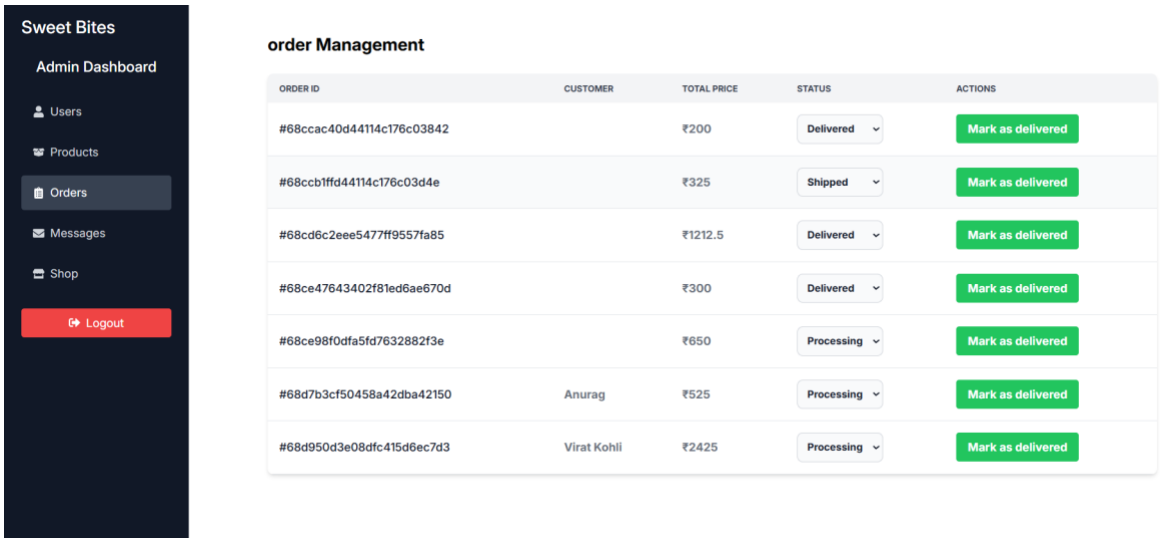
Admin Panel - Admin Succesfully Logged In



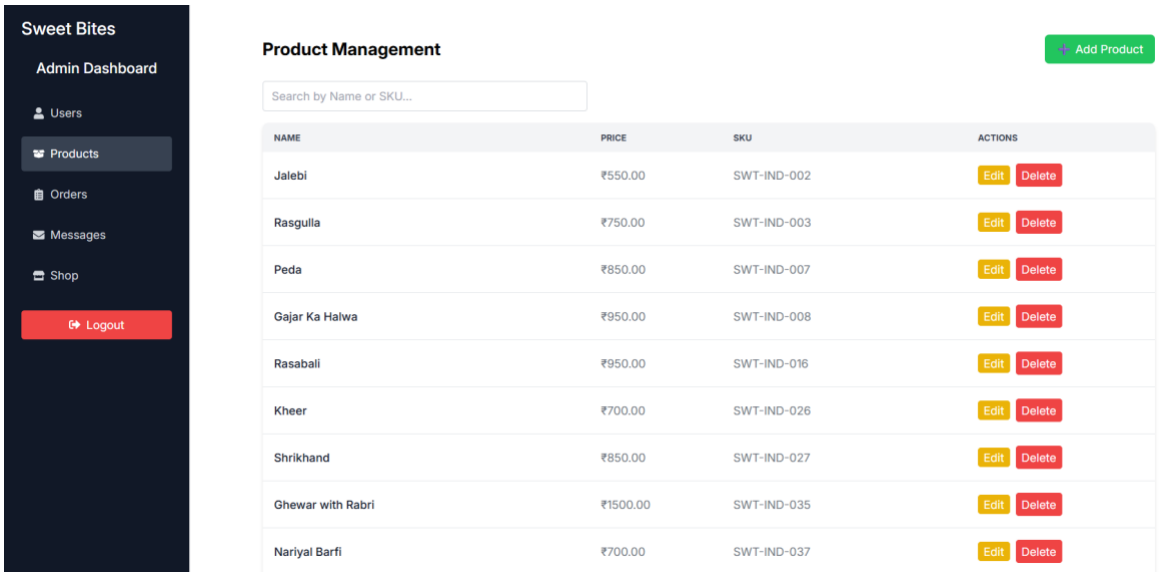
Admin DashBoard



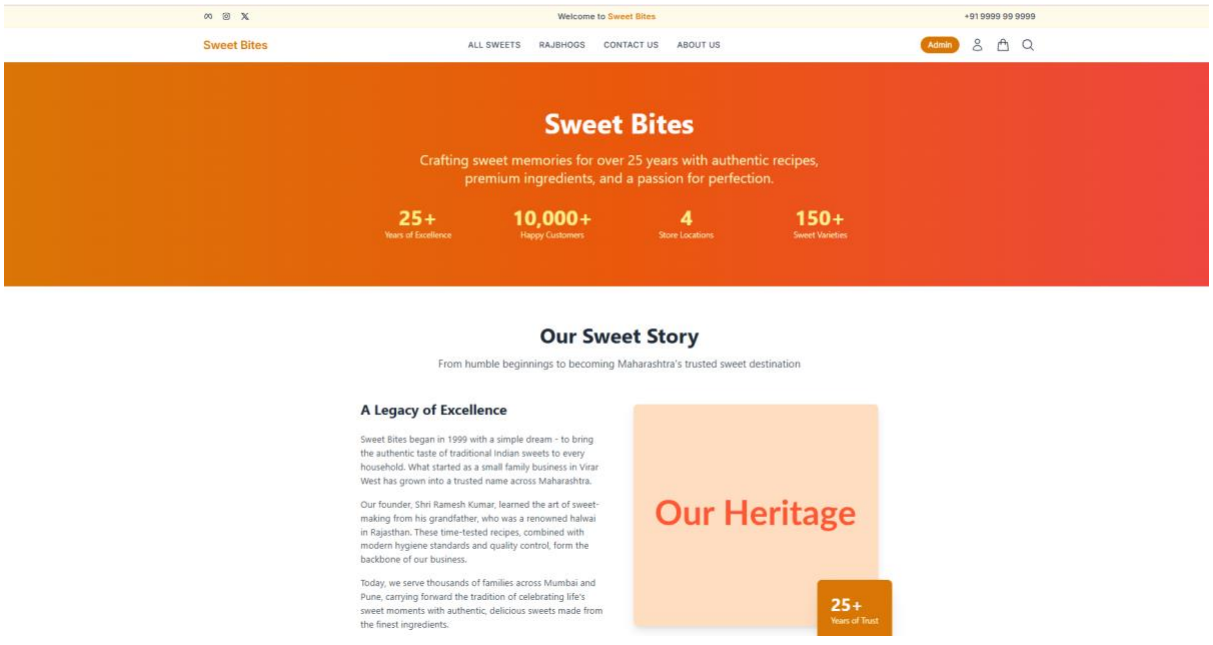
OrderManagement From Admin Section



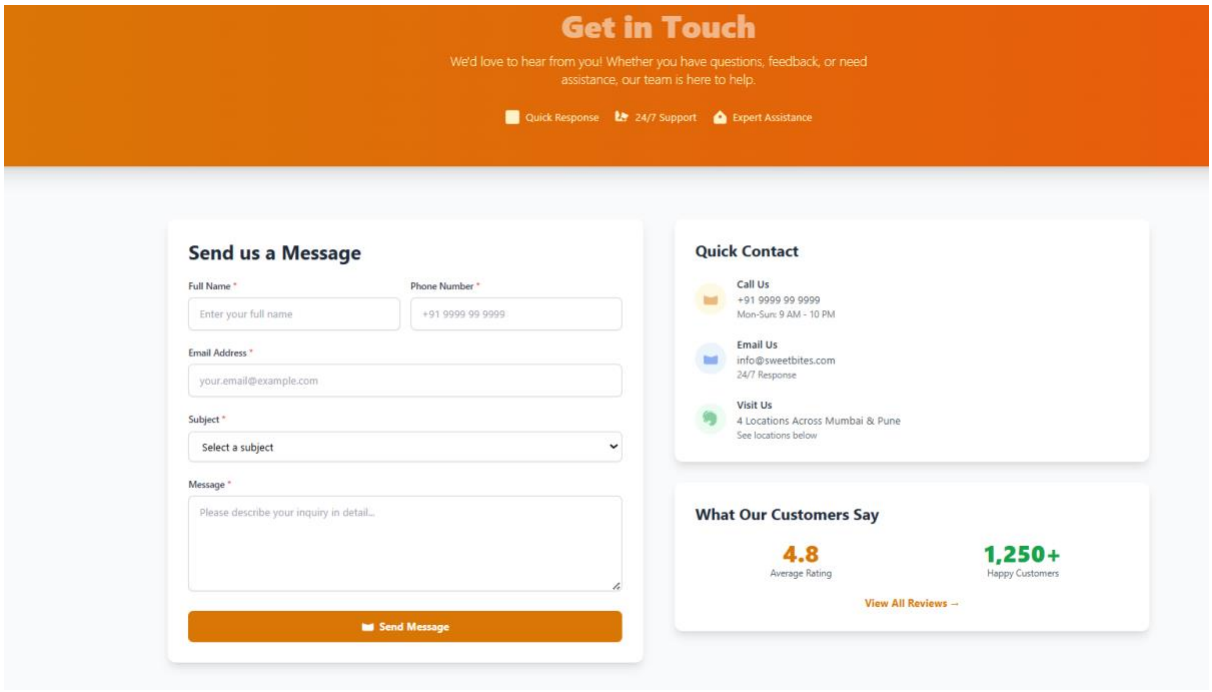
ProductManagement From Admin Section



About Us Page



Contact Us Page



CHAPTER 8. CONCLUSION & FUTURESCOPE

8.1 CONCLUSION

The **Sweets Shop Web Application** successfully addresses the growing demand for a modern, efficient, and user-friendly platform that brings the traditional sweets shopping experience into the digital era. By leveraging advanced technologies such as React, Node.js, and MongoDB, the system ensures scalability, performance, and a seamless user experience.

The application benefits customers by providing them with an intuitive interface to browse a variety of sweets, add them to a cart, place secure orders, and complete payments easily. At the same time, it empowers shop owners and administrators to manage products, update prices, handle orders, and maintain stock efficiently.

Throughout the development process, emphasis was placed on secure payment handling, responsive design, and a reliable backend to ensure smooth interaction between users and the system. The inclusion of features such as order tracking, admin controls, and mobile responsiveness further enhances its usability.

The **Sweets Shop Web Application** is a significant step forward in modernizing the way sweets are marketed and purchased, bridging the gap between traditional sweet stores and digital commerce. This project demonstrates the potential for digital transformation in local businesses and lays the foundation for future innovations and improvements.

8.2 FUTURE SCOPE AND ENHANCEMENTS

While the **Sweets Shop Web Application** already provides a strong foundation, there are several opportunities for enhancement and expansion as user expectations and technologies evolve:

1. **Mobile Application Development:**
   Creating a dedicated Android and iOS mobile app will enhance customer convenience by offering features like push notifications for new arrivals, festive offers, and real-time delivery updates.
2. **AI-Powered Personalization:**
   Artificial intelligence and machine learning can be used to provide personalized sweet recommendations based on user preferences, past orders, and festive occasions. This can also enable dynamic pricing and special offers.
3. **Voice-Enabled Ordering:**
   With voice assistants becoming popular, integrating voice-enabled shopping through Alexa or Google Assistant can make the ordering process more interactive and hands-free.

4. **Subscription-Based Sweet Boxes:**

   Customers can subscribe to receive their favorite sweets or special festive sweet boxes on a weekly or monthly basis, ensuring regular engagement and building customer loyalty.

5. **Expansion to Multiple Outlets and Cities:**

   Scaling the platform to serve multiple outlets and expanding into new cities will increase accessibility for more customers and allow local sweet shops to grow digitally.

6. **Eco-Friendly and Customizable Packaging:**

   Offering sustainable packaging options and personalized gift boxes for festivals, weddings, and celebrations can add value and attract environmentally conscious customers.

CHAPTER 9. REFERENCES

1) **React.js Documentation:** https://react.dev/
2) **Tailwind CSS Documentation:** https://tailwindcss.com/docs
3) **Node.js Documentation:** https://nodejs.org/docs/
4) **Express.js Documentation:** https://expressjs.com/
5) **MongoDB Documentation:** https://www.mongodb.com/docs/
6) **Razorpay Documentation:** https://razorpay.com/docs/?preferred-country=IN#home-payments
7) **Plagiarism report:** https://plagiarismdetector.net/
8) **ER Diagram:** https://www.geeksforgeeks.org/dbms/introduction-of-ermodel
9) **Activity Diagram:** https://www.geeksforgeeks.org/system-design/unifiedmodeling-language-uml-activity-diagrams/
10) **Data Flow Diagram:** https://www.geeksforgeeks.org/software-engineering/what-is-dfddata-flow-diagram/
11) **Flowchart Diagram:** https://www.geeksforgeeks.org/computer-science-fundamentals/what-is-a-flowchart-and-its-types/
12) **Gantt chart:** https://www.youtube.com/watch?v=xsxi4qaEnOg&t=21s%20%20