



CHANDIGARH
UNIVERSITY

Discover. Learn. Empower.

Major Project Report
on
"STOCK AND SHOP: A FULL-STACK
E-COMMERCE SYSTEM USING THE MERN
STACK"

**Submitted in partial fulfilment for the requirements of the
degree of**

BACHELOR OF COMPUTER APPLICATIONS
CENTER FOR DISTANCE & ONLINE EDUCATION
CHANDIGARH UNIVERSITY, MOHALI
PUNJAB

SUBMITTED TO :

Programme Coordinator – BCA
Center for Distance & Online Education
Chandigarh University, Mohali
Punjab

SUBMITTED BY :

Saanvi Suresh Rao
O23BCA160051
BCA – Batch – Jan 2023



I hereby declare that this Project Report titled "**Stock And Shop: A Full-Stack E-Commerce system using the Mern stack**" is an authentic record of my own work carried out during the academic year **2025-2026**, in partial fulfillment of the requirements for the award of the degree of **Bachelor Of Computer Applications** at **Chandigarh University**.

I further declare that the work presented in this report is original and has not been previously submitted in part or full for any other degree, diploma, or similar award at any other university or institution. All sources of information and help have been duly acknowledged.

Saanvi Suresh Rao

O23BCA160051

Bachelor Of Computer Application

Center for Distance & Online Education

Date: November 29, 2025

Place: Vapi, Gujarat

ACKNOWLEDGEMENT

I want to sincerely thank everyone who helped me finish my project, **“Stock And Shop: A Full-Stack E-Commerce system using the Mern stack”** and for their support.

I am extremely grateful to Chandigarh University for giving me the chance to complete this project as a requirement for my Bachelor Of Computer Application degree. I am grateful to Chandigarh University's academic staff and administration for creating and implementing a thorough program that gave me the skills and information I needed to finish this work.

I also want to thank the technical support staff for helping to make sure that the learning process runs smoothly.

Lastly, I would like to express my sincere gratitude to my family for their unwavering understanding, encouragement, and support during the difficult stages of this project.

Saanvi Suresh Rao

O23BCA160051

Bachelor Of Computer Applications – CU ONLINE

ABSTRACT

This major project introduces "**Stock & Shop**," an e-commerce platform developed to address the process inefficiencies inherent in traditional grocery shopping while providing a secure and scalable online marketplace for groceries, vegetables, and fruits.

The system focuses on establishing a full-stack, dynamic environment for retail, ensuring improved inventory visibility and streamlined transaction processing. The platform is built on the **MERN stack**, utilizing **MongoDB** for flexible product and order data storage, **Express.js** and **Node.js** for a high-performance backend API, and **React** for a dynamic, user-friendly shopping experience.

Key functionalities include secure user authentication, real-time cart management, and a robust **online payment gateway integrated via Stripe**. A critical design element is the focus on **transaction reliability**, ensuring orders are created, and inventory is updated only upon verified, final confirmation of payment receipt.

"Stock & Shop" provides a comprehensive, modern e-commerce solution, significantly enhancing operational proficiency and demonstrating mastery of full-stack development and third-party financial service integration.

TABLE OF CONTENTS

- **Title Page**
- **Declaration / Bonafide Certificate**
- **Acknowledgement**
- **Abstract**
- **Table of Contents**

Chapters

- **Chapter 1: Background**
 - **Project Profile and Objectives**
 - **Scope of the Project**
 - **Existing System Limitations and Proposed Solution**
 - **Technology Used**
- **Chapter 2: System Analysis**
 - **Study of Requirements**
 - **Feasibility Study**
 - **Hardware and Software Requirements**
 - **System Planning**
- **Chapter 3: System Design**
 - **System Architecture**
 - **ER-Diagram**
 - **Data Flow Diagrams**
 - **Database Design and Schema**
 - **Input And Output Screen Design**

- **Chapter 4: Implementation**

- **Process Logic of Authentication Module**
- **Process Logic of Product and Inventory Module**
- **Process Logic of Payment Gateway**
 - **Secure Webhook Handler Implementation**
 - **Order Fulfillment and Cart Clearance Logic**
- **Code implementation Details**

- **Chapter 5: Testing And Maintenance**

- **Testing Methodology Adopted**
- **Test Cases and Expected Results**
- **System Evaluation and Future Scope**
- **Cost and Benefit Analysis**
- **User/Operational Manual**
 - **CONLUSION**
 - **BIBLIOGRAPHY / REFERENCES**
 - **APPENDICES**

CHAPTER – 1

BACKGROUND

PROJECT PROFILE AND OBJECTIVE:

The project, titled "**E-COMMERCE SYSTEM USING MERN STACK**" (named **Stock & Shop** in the application), is a robust, full-stack application developed to serve as a modern online platform for purchasing groceries, fresh vegetables, and fruits. The system is architected around the highly scalable **MERN (MongoDB, Express.js, React, Node.js) technology stack**.

Project Profile

DETAIL	DESCRIPTION
Project Name	Stock & Shop
Domain	E-commerce for Groceries and Inventory Management
Architecture	Full-Stack, Three-Tier (MERN)
Key Technology	MongoDB, React, Node.js/Express, Stripe Payment Gateway
Target Users	Retail Customers and System Administrators

Project Objectives

The core objectives of developing the Stock & Shop E-commerce System are as follows:

- 1. To Implement a Scalable MERN Architecture:** To design and develop a high-performance application utilizing a component-based React frontend and a modular Node.js/Express backend, ensuring the system can handle a growing user base and product catalog.

2. **To Ensure Secure and Controlled Access:** To establish a complete user lifecycle management system, including secure authentication, Role-Based Access Control (RBAC) to differentiate Customer and Administrator privileges, and a secure Forgot Password feature utilizing OTP (One-Time Password) email verification.
3. **To Provide Real-Time Inventory Control:** To implement an efficient Admin Dashboard allowing for CRUD (Create, Read, Update, Delete) operations on categories, sub-categories, and products, ensuring accurate and real-time tracking of stock quantity.
4. **To Integrate a Reliable Payment Solution:** To integrate the Stripe Payment Gateway for secure, third-party processing of all online credit/debit card transactions, maintaining strict security and PCI compliance.
5. **To Guarantee Transaction Reliability:** To design and implement the complex secure order fulfillment logic on the backend to ensure that permanent order records are created and customer carts are cleared only after a verified, successful payment signal is received from Stripe.

Scope of the Project:

The project scope defines the functional boundaries of the "Stock & Shop" E-commerce System, covering the complete lifecycle from inventory management to order fulfillment using the MERN stack and Stripe.

- 1. Authentication and Security:** Includes secure user **Login**, **Register**, and **Forgot Password** with OTP verification. The system implements **Role-Based Access Control (RBAC)** to distinguish between Customer and Administrator privileges.
- 2. Inventory Management:** The Administrator module is fully in scope, enabling **CRUD (Create, Read, Update, Delete)** operations on all **Groceries, Categories, and Sub-Categories**. The system tracks and updates **real-time stock quantity**.
- 3. Customer Shopping Experience:** The scope covers the user interface for product **browsing, searching, cart management**, and the ability to save multiple **shipping addresses**.
- 4. Secure Payment and Fulfillment:** The system integrates the **Stripe API** to initiate online payment sessions. Crucially, the backend implements **secure server-to-server confirmation logic** to ensure orders are created and the customer's cart is cleared **only** upon verified payment success.

Existing System Limitations and Proposed Solution:

This section analyzes the shortcomings of traditional or older retail systems, which the Stock & Shop E-commerce platform is designed to address through its modern technology stack.

A. Limitations of the Existing/Manual System

LIMITATION	DESCRIPTION
Inaccurate Inventory	Manual tracking of grocery and produce stock is prone to human error, leading to inaccurate availability displayed to the customer.
Limited Accessibility	Physical stores or basic websites are not accessible 24/7, restricting sales opportunities and customer convenience.
Payment Insecurity	Systems relying on cash on delivery or non-integrated payment methods lack security and real-time transaction verification, posing fraud risks.
Lack of Data Centralization	Customer data, orders, and sales history are often scattered, making trend analysis and efficient resource planning difficult.

B. The Proposed Solution

The "Stock & Shop" E-commerce System overcomes these limitations by implementing a modern, integrated, and scalable solution:

1. Real-Time Data Management: The MERN Stack (MongoDB) provides a centralized database for all inventory, customer, and order data, enabling the Admin to view and update stock quantities in real-time.

2. 24/7 Accessibility: The React frontend provides a dynamic, accessible online platform that operates continuously, significantly expanding the market reach.
3. Secure Financial Transactions: Integration with the Stripe Payment Gateway offloads sensitive card processing, ensuring PCI compliance and providing instant, verified payment confirmation via a secure server-to-server connection.
4. Automated Fulfillment: The backend logic automatically manages order creation, inventory updates, and cart clearance upon verified payment, maximizing efficiency and minimizing human error.

Technology Used:

The "Stock & Shop" E-commerce System is a Full-Stack application built on the MERN technology stack, supplemented by the Stripe Payment Gateway. This modern architecture was chosen for its flexibility, scalability, and ability to handle high-performance web applications.

COMPONENT	TECHNOLOGY	ROLE IN THE SYSTEM
M	MongoDB	The NoSQL Database used for flexible and scalable data storage. It manages all application data, including user profiles, product catalogs, and order records.
E	Express.js	The Web Application Framework for Node.js. It handles routing, middleware, and API endpoints, forming the robust backbone of the server.
R	React	The JavaScript Library used to build the dynamic, component-based user interface (UI) for both the Customer shopping experience and the Administrator dashboard.
N	Node.js	The JavaScript Runtime Environment that executes the Express.js server logic. It enables fast, non-blocking I/O operations necessary for a high-traffic E-commerce API.
Payment	Stripe API	The integrated third-party service that securely manages all credit/debit card transactions and provides the necessary webhook signals for reliable order fulfillment.

Development Tools and Libraries

The project leverages several critical tools beyond the core stack:

- **JWT (JSON Web Tokens):** Used for secure, stateless user authentication and session management.
- **Bcrypt:** Utilized for secure hashing of user passwords before storage in MongoDB.
- **Stripe CLI:** Essential for local development, enabling the simulation of live webhook events on the local machine for testing the order fulfillment logic.
- **NodeMailer:** Used for sending the **OTP (One-Time Password)** via email during the Forgot Password process.

CHAPTER - 2

SYSTEM ANALYSIS

Study of Requirements

The Study of Requirements defines the functional, non-functional, and operational needs of the “Stock & Shop” E-commerce System. This ensures that the system aligns with user expectations, performance goals, security needs, and platform constraints. The requirements were analyzed from the perspectives of customers, administrators, and system behavior, forming the foundation for all design and implementation activities.

A. Functional Requirements

Functional requirements define *what the system must do*. These are directly derived from the objectives of the Stock & Shop platform.

1. User Authentication and Authorization

- Secure User Registration and Login functionality.
- Role-Based Access Control (RBAC) distinguishing **Customer** and **Admin** users.
- Forgot Password workflow using OTP email verification.
- Authenticated session handling using JWT tokens.

2. Product and Inventory Management (Admin Module)

- CRUD operations for Categories, Sub-categories, and Products.

- Real-time updating of product stock quantities.
- Uploading product images, setting prices, descriptions, and category associations.
- Dashboard displaying inventory status.

3. Customer Shopping Experience (User Module)

- Product browsing, searching, and filtering.
- Add-to-cart, update quantity, and remove-from-cart functionalities.
- Management of multiple shipping addresses.
- Viewing order history and order details.

4. Order Management and Payment Integration

- Creating a temporary order session before payment.
- Integration with Stripe for secure payment processing.
- Receiving secure **webhook events** from Stripe.
- Creating final order records only after verified payment success.
- Auto-clearing the user's cart upon successful payment.

B. Non-Functional Requirements

Non-functional requirements ensure performance, availability, reliability, and ease of use.

1. Performance Requirements

- The system must handle multiple concurrent user sessions.
- API responses must be optimized using asynchronous, non-blocking Node.js operations.
- Database queries must be index-optimized for faster reads.

2. Security Requirements

- Passwords must be encrypted using Bcrypt.
- JWT-based session authentication ensures secure access.
- Stripe ensures PCI-compliant payment processing.
- Sensitive logic (order creation, inventory update) executed **only** on secure server-side webhook confirmation.

3. Reliability Requirements

- Webhook handler must ensure idempotency to prevent duplicate orders.
- MongoDB collections must guarantee atomic updates on inventory count.
- The application must ensure uptime with minimal failure points.

4. Usability Requirements

- Intuitive navigation for both customer and admin dashboards.
- Mobile-responsive UI using React components.
- Clear visual feedback for cart operations and checkout flow.

5. Scalability Requirements

- MERN architecture allows horizontal scaling of backend and database.
- React frontend supports modular expansion for future features.

C. User Requirements

User requirements define what **end users expect** from the system.

Customer Requirements

- Ability to create an account and login securely.

- Browse products easily with clear categories.
- Add items to cart and complete payments online.
- Track order status and view past orders.

Administrator Requirements

- Ability to manage inventory in real time.
- Add, update, or delete categories and products.
- Monitor product availability and update stock accordingly.
- View and manage customer orders.

D. System Requirements

These specify the expected system behavior and constraints.

- The application must be accessible 24/7 through the internet.
- Backend must expose RESTful API endpoints for all operations.
- MongoDB must handle structured and unstructured product data efficiently.
- The system must integrate third-party services (Stripe, NodeMailer) without compromising performance.

Feasibility Study

The feasibility study evaluates whether the “Stock & Shop” E-commerce System can be successfully developed and operated using the MERN stack and Stripe payment integration. It covers technical, operational, and economic aspects to ensure that the proposed solution is practical, achievable, and beneficial.

A. Technical Feasibility

The technologies selected for the project—MongoDB, Express.js, React, Node.js, and Stripe—are modern, well-documented, and widely used for scalable web applications. The MERN stack provides high performance, non-blocking operations, and flexible data handling, making it technically feasible to implement real-time inventory updates, authentication workflows, and payment processing. All required tools and development environments are freely accessible and supported by extensive online resources, ensuring smooth project development.

B. Operational Feasibility

The system addresses the limitations of traditional grocery shopping by providing 24/7 accessibility, real-time stock visibility, and secure online payments. The customer interface is designed for easy browsing, cart management, and checkout, while the admin dashboard simplifies product

and inventory management. The solution is practical for real-world use, supports seamless online ordering, and can be operated by both customers and administrators without specialized training.

C. Economic Feasibility

The project is economically viable as it uses open-source technologies that eliminate licensing costs. Tools such as MongoDB, Node.js, React, and Express are completely free to use. Stripe follows a pay-as-you-go model, which means there are no upfront charges for integration or testing. The overall development and maintenance cost remains low, making the system cost-effective for deployment, scaling, and long-term use.

Hardware And Software Requirements

This section outlines the minimum hardware and software resources required to develop and operate the “Stock & Shop” E-commerce System. Since the application is MERN-based and uses Stripe for payment integration, the requirements are lightweight and suitable for modern development environments.

A. Hardware Requirements

For Development

- **Processor:** Intel i5 or equivalent
- **RAM:** 8 GB (minimum)
- **Storage:** 20 GB free space
- **System:** Windows / macOS / Linux

For Deployment (Server)

- **Processor:** Dual-core CPU
- **RAM:** 4–8 GB
- **Storage:** SSD preferred
- **Network:** Stable broadband connection

B. Software Requirements

Backend

- Node.js
- Express.js
- MongoDB
- Stripe API
- Postman (for API testing)

Frontend

- React
- HTML, CSS, JavaScript
- Axios for API requests

Development Tools

- VS Code
- Git and GitHub
- MongoDB Compass
- Stripe CLI
- npm packages (JWT, Bcrypt, Nodemailer, etc.)

System Planning

System planning involves defining the structure and workflow of the Stock & Shop platform to ensure smooth development and efficient operation.

1. Modular Architecture:

The system is divided into modules such as Authentication, Product Management, Cart, Orders, and Payment Integration for easier development and maintenance.

2. Three-Tier MERN Structure:

- Frontend (React): Handles UI and customer interactions.
- Backend (Node/Express): Manages business logic and API routes.
- Database (MongoDB): Stores users, products, orders, and inventory data.

3. API Planning:

REST-based endpoints were planned for user actions, admin operations, cart updates, and order processing.

4. Database Planning:

Collections were planned for Users, Products, Categories, Orders, and Cart Items.

5. Payment Workflow Planning:

Stripe Checkout and Webhook handling were designed to ensure secure order creation only after valid payment confirmation.

CHAPTER – 3

SYSTEM DESIGN

System Design defines the structural, architectural, and data-oriented blueprint of the “Stock & Shop” E-commerce System. The purpose of this chapter is to illustrate how different components of the application interact, how data flows between modules, and how the system ensures secure and efficient operations.

System Requirements

The “Stock & Shop” application follows a Three-Tier MERN Architecture, ensuring modularity, scalability, and efficient separation of concerns.

A. Presentation Layer (Frontend – React)

- Displays categories, products, cart, checkout, and admin dashboard.
- Manages user interactions through reusable components.
- Communicates with backend APIs using Axios.
- Implements client-side routing for smooth navigation.

B. Application Layer (Backend – Node.js + Express.js)

- Handles all business logic for authentication, products, orders, and payments.

- Validates user requests and enforces role-based permissions.
- Connects to Stripe for payment session creation and webhook verification.
- Generates JWTs and processes encrypted credentials.

C. Data Layer (Database – MongoDB)

- Stores Users, Products, Categories, Orders, and Cart items.
- Maintains real-time stock quantities.
- Ensures consistency during order confirmation via secure server-side logic.

Data Flow Summary:

React → Express API → MongoDB

Stripe Payment → Webhook → Express → MongoDB (Order creation)

ER Diagram (Entity–Relationship Overview)

The ER model of the system identifies key entities and relationships required for managing the e-commerce workflow.

Major Entities

1. User

- Fields: name, email, password, role, addresses
- Relationship: User → Orders (1-to-many)

2. Product

- Fields: name, categoryId, price, stock, image
- Relationship: Category → Products (1-to-many)

3. Category / Sub-category

- Grouping for products
- Helps streamline admin management and product filtering

4. Order

- Fields: userId, items, totalAmount, paymentStatus, timestamp
- Relationship: Order → User (many-to-1)

5. Cart

- Temporary storage of selected products before checkout
- Related to User (1-to-1)

High-Level Relationships

- User places Orders
- User has Cart
- Category contains Products
- Order contains multiple Products

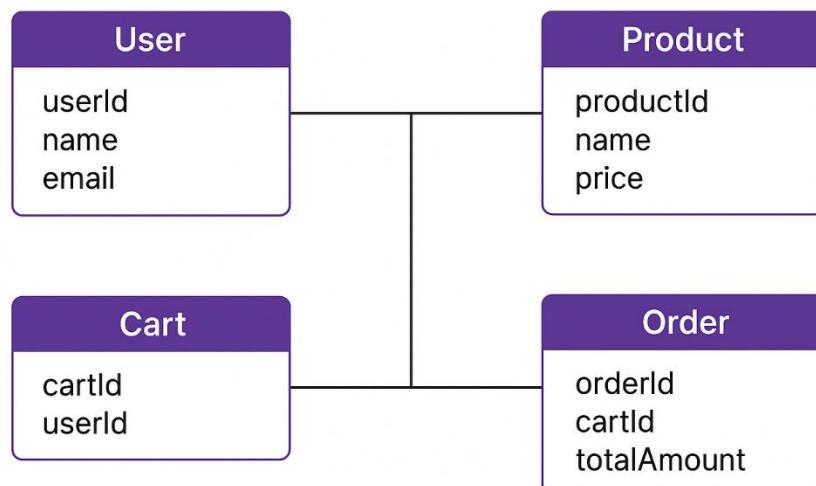


Figure (a) : Entity Relationship Diagram of Stock and Shop

Data Flow Diagrams (DFD)

DFDs illustrate how data moves within the system.

A. Level 0 DFD – Context Diagram

Shows the system as a single process interacting with:

- Customer
- Administrator
- Stripe Payment Gateway

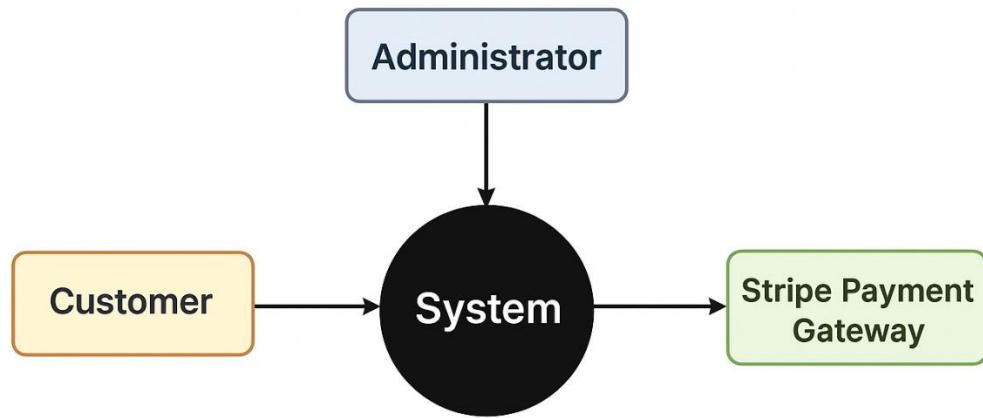


Figure (b) : Level 0 DFD

B. Level 1 DFD – Expanded Processes

1. User Process:

Login → Browse Products → Add to Cart → Checkout

2. Admin Process:

Manage Categories → Manage Products → Update Stock

3. Payment Process:

Initiate Payment → Stripe Checkout → Webhook Response → Order Creation

B. Level 1 DFD – Expanded Processes

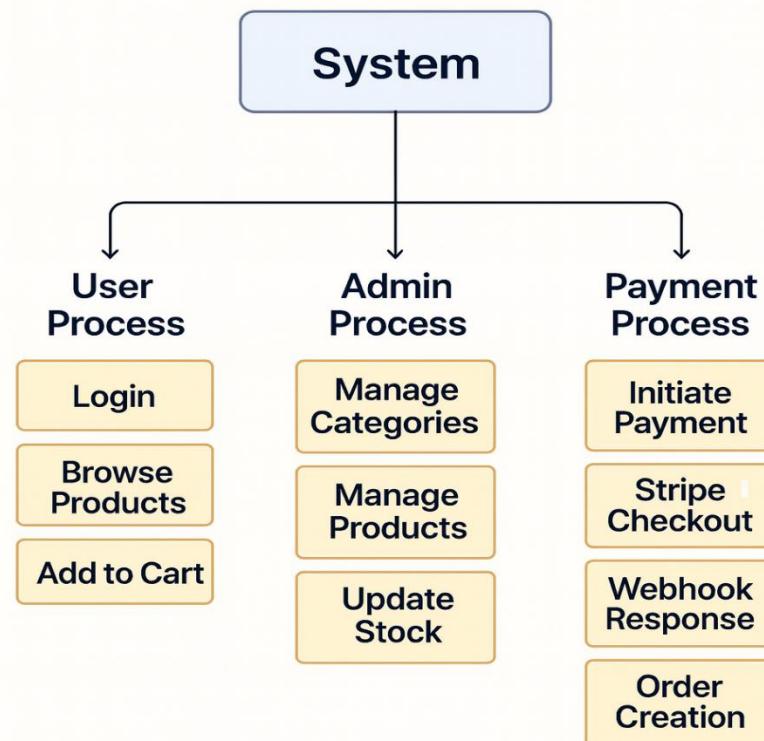


Figure (c) : Level 1 DFD

C. Level 2 DFD – Deep Payment Flow

- Backend creates payment session
- Stripe processes card
- Stripe webhook confirms payment
- Backend updates inventory and creates order
- Cart items are cleared

C. Level 2 DFD Deep Payment Flow

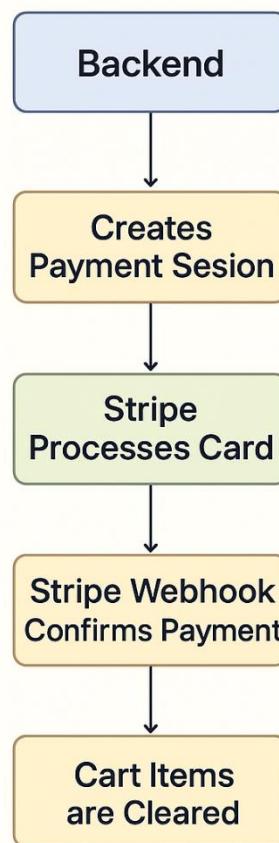


Figure (d) : Level 2 DFD

Database design and schema

The database uses a collection-based schema, ideal for flexible e-commerce data.

Main Collections

1. Users

- name, email, role, hashed password, addresses

2. Products

- title, description, category, stock, price, image URL

3. Categories / Subcategories

- name, parentCategoryId (optional)

4. Orders

- userId, orderItems, totalAmount, paymentIntentId, orderStatus

5. Cart Items

- userId, productId, quantity

Design Principles

- Use of ObjectId references for relationships
- Inventory decremented only after verified payment
- Orders linked to both user and Stripe payment session
- Timestamped records for tracking and sorting

Input and Output Screen Design

The screen designs define the visual interaction points for both customers and administrators.

A. Customer Screens

- Home Page: Categories and product listings
- Product Page: Product details, stock, and Add-to-Cart option
- Cart Page: Update quantity, remove items
- Checkout Page: Payment initiation via Stripe
- Order History: View previous orders

B. Admin Screens

- Dashboard: Summary of products and stock
- Add/Edit Product: Form-based product management
- Category Management: CRUD operations
- View Orders: Complete order details and statuses

C. Output Screens

- Payment Success Screen
- Order Confirmation Screen
- Error or Validation Messages

CHAPTER – 4

IMPLEMENTATION

This chapter describes the practical implementation of the “**Stock & Shop**” E-Commerce System, developed using the MERN stack. The implementation covers authentication, admin inventory management, customer modules, secure payment gateway integration, and backend business logic.

Process Logic of Authentication Module

The Authentication Module handles user registration, login, and secure password reset using OTP. It also identifies whether the user is an Admin or Customer and routes them accordingly._

4.1.1 Registration Logic

Process:

- User enters name, email, and password.
- Backend validates email uniqueness.
- Password is hashed using bcrypt.
- New user is saved with default role “customer”.

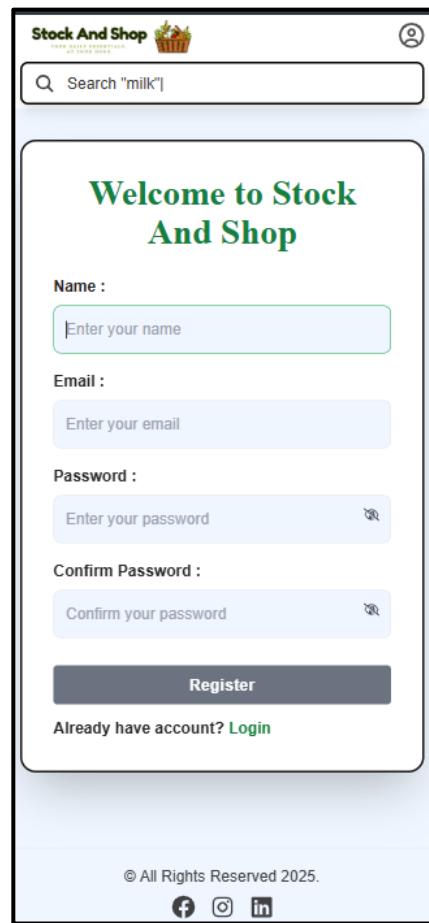
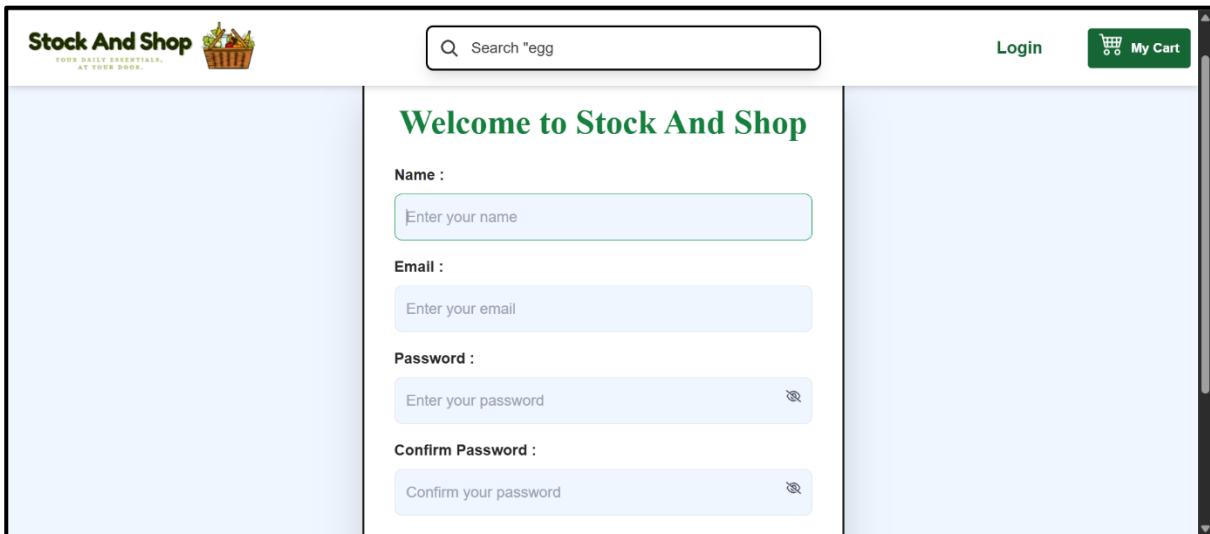
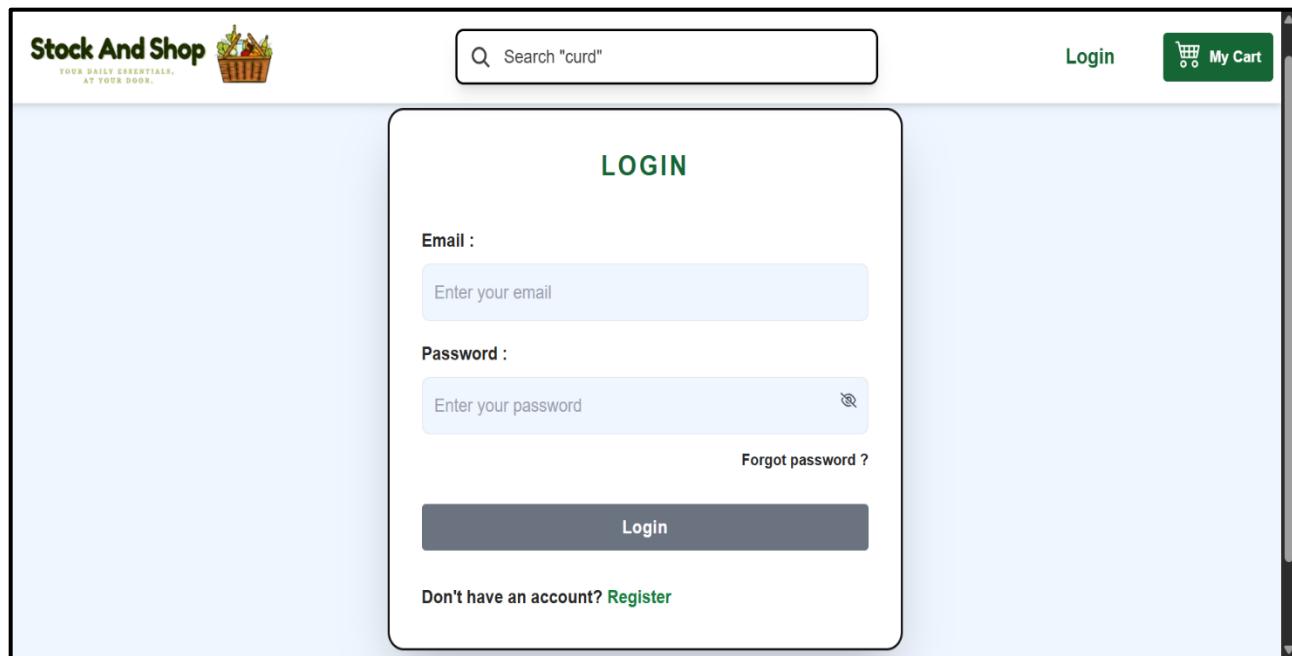


Figure 4.1: User Registration Page

4.1.2 Login Logic

Process:

- User enters credentials.
- Server verifies hashed password.
- JWT token is generated.
- If role = Admin → redirect to Admin Dashboard.
- If role = Customer → redirect to User Homepage



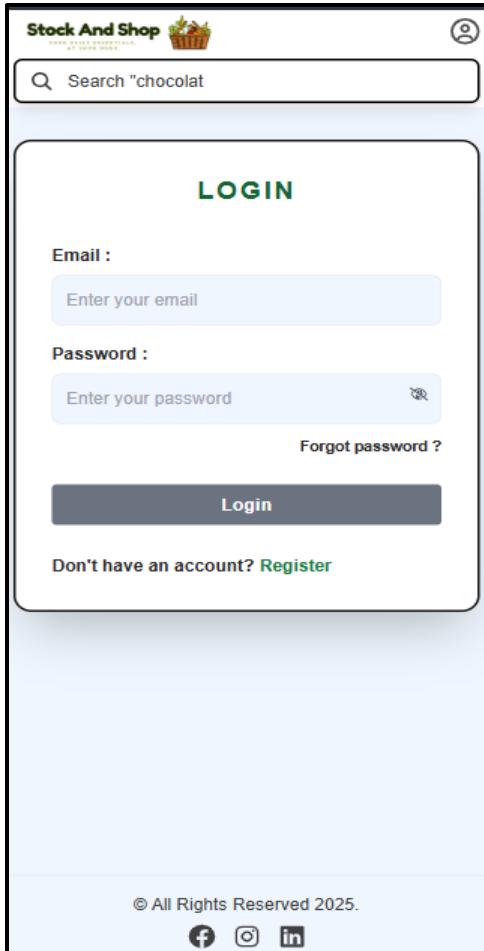


Figure 4.2: User Login Page

4.1.3 Forgot Password (OTP) Logic

Process:

- User enters registered email.
- OTP is generated and emailed using **Nodemailer**.
- User verifies OTP.
- System allows new password creation.

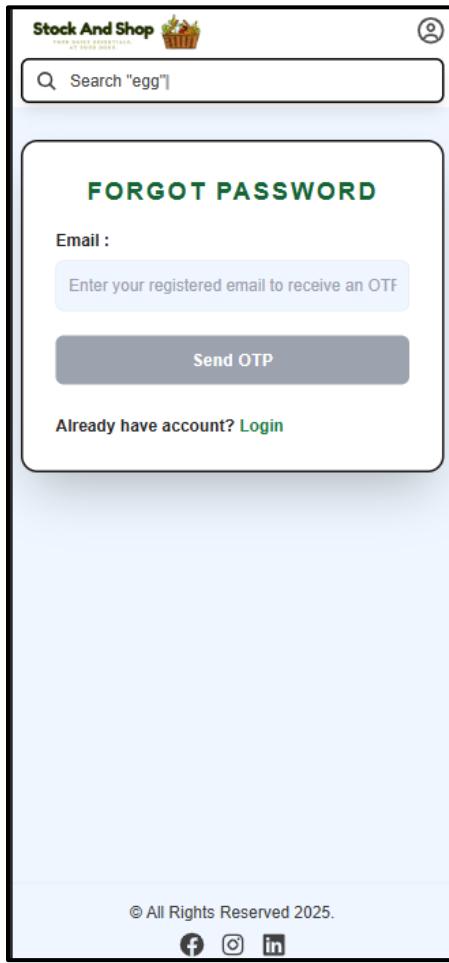
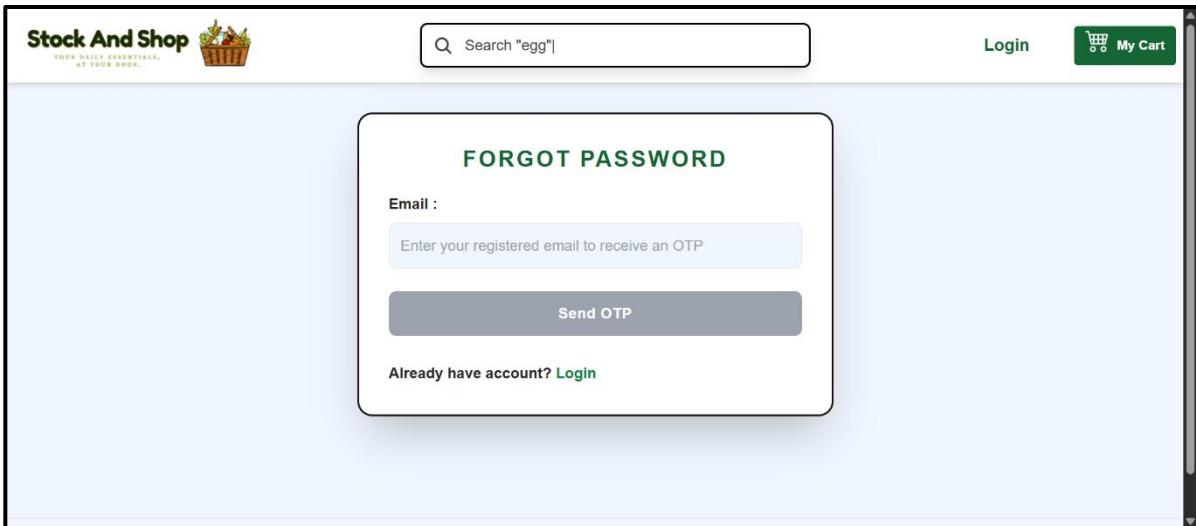


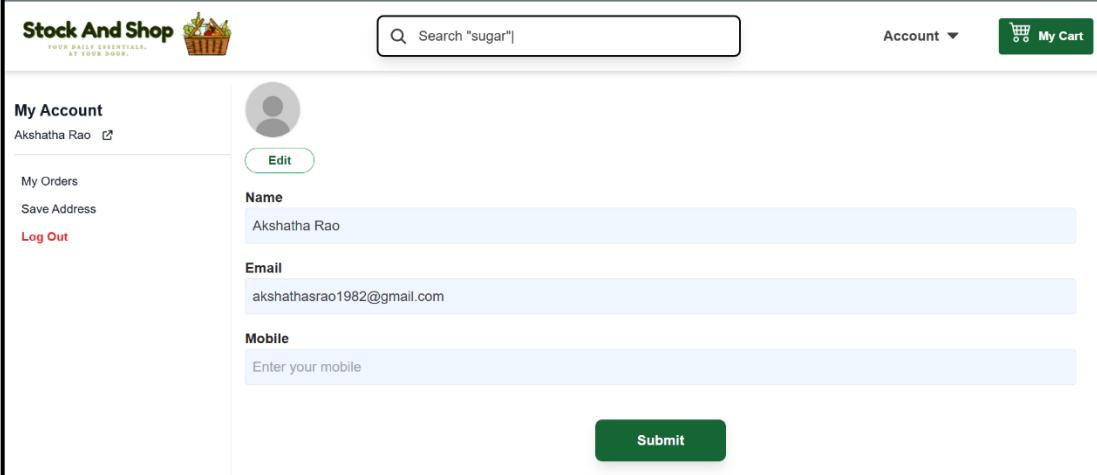
Figure 4.3: Forgot Password

4.1.4 Authenticated User Details Page

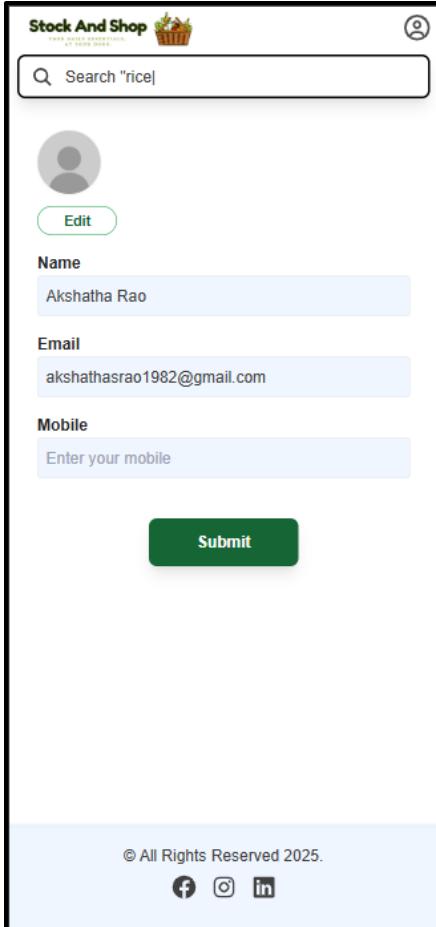
Both Admin and Customer have profile pages displaying personal details.

The image displays two screenshots of the Stock And Shop website, illustrating the Admin Profile Page. Both screenshots show a similar layout with a header containing the logo 'Stock And Shop' and a search bar. The right screenshot shows a green navigation bar with 'Account' and 'My Cart' options. The main content area on both pages includes a sidebar with links like 'Category', 'Sub Category', 'Upload Product', 'Product', 'My Orders', 'Save Address', and 'Log Out'. The central area shows a user profile with a placeholder image, an 'Edit' button, and fields for 'Name' (Saanvi Rao), 'Email' (saanvirao04@gmail.com), and 'Mobile' (7582149680). A large green 'Submit' button is at the bottom. The left screenshot shows a different state where the search bar contains the text 'Search "bread"'. The right screenshot shows a different state where the search bar contains the text 'Search "sugar"'. At the bottom of the right screenshot, there is a footer with copyright information and social media icons for Facebook, Instagram, and LinkedIn.

Figure 4.4: Admin Profile Page



The screenshot shows the 'My Account' section of the Stock And Shop website. At the top, there's a logo for 'Stock And Shop' with the tagline 'EVERYDAY ESSENTIALS AT YOUR DOOR.' A search bar contains the placeholder 'Search "sugar"'. On the right, there are 'Account' and 'My Cart' buttons. The main area has a sidebar with 'My Orders', 'Save Address', and a red 'Log Out' link. The profile section shows a placeholder user icon, an 'Edit' button, and fields for 'Name' (Akshatha Rao), 'Email' (akshathasrao1982@gmail.com), and 'Mobile' (placeholder 'Enter your mobile'). A green 'Submit' button is at the bottom.



The screenshot shows the same 'My Account' section as the desktop view, but it's displayed on a mobile device with a vertical orientation. The layout is adjusted to fit the screen, and the overall design is responsive. The bottom of the page includes a footer with the text '© All Rights Reserved 2025.' and social media icons for Facebook, Instagram, and LinkedIn.

Figure 4.5: Customer Profile Page

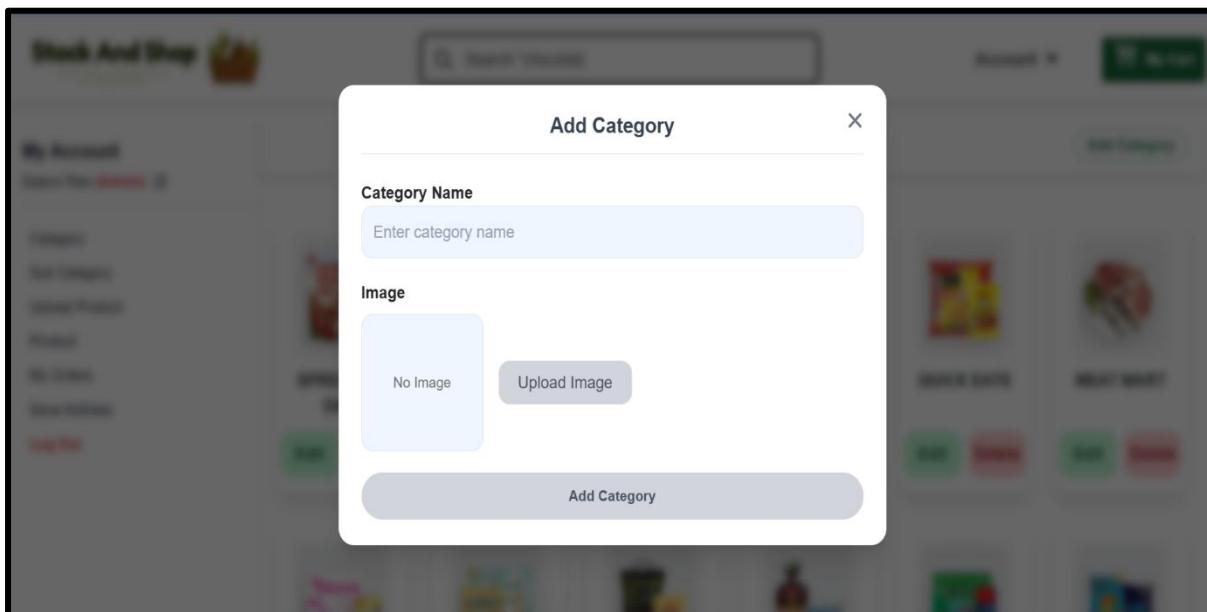
4.2 Process Logic of Product & Inventory Module (Admin)

This module is accessible only to the **Administrator** and provides complete control over categories, sub-categories, products, and inventory stock.

4.2.1 Category Management

Process:

- Admin enters category name and image.
- Category is stored in MongoDB.
- Categories are displayed dynamically during product upload.



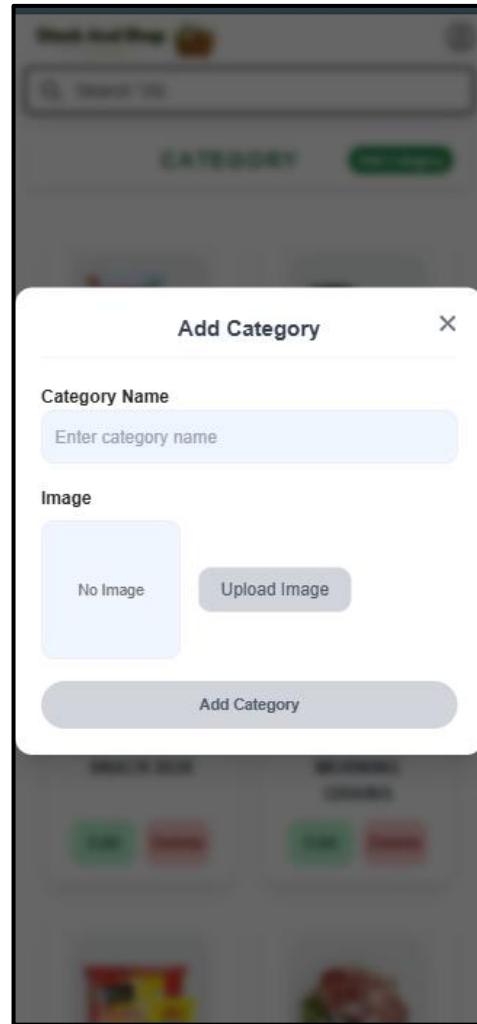


Figure 4.6: Admin – Add Category Page

A screenshot of the 'Stock And Shop' admin dashboard. The top navigation bar includes the logo 'Stock And Shop' with the tagline 'YOUR DAILY ESSENTIALS. AT YOUR DOOR.', a search bar with the placeholder 'Search "egg"', an 'Account' dropdown, and a 'My Cart' button. On the left, there's a sidebar with links for 'My Account' (selected), 'Category', 'Sub Category', 'Upload Product', 'Product', 'My Orders', 'Save Address', and 'Log Out'. The main content area is titled 'CATEGORY' and displays six categories with sub-categories: 'SPREADS & DIPS' (with Nutella and Kissan Jam), 'SPICES & NUTS' (with Chana Dal and Masala), 'SNACK BOX' (with Lays and Doritos), 'MORNING GRAINS' (with Kellogg's Corn Flakes), 'QUICK EATS' (with Maggi and Oreo), and 'MEAT MART' (with Chicken). Each category card has 'Edit' and 'Delete' buttons at the bottom. Below these cards are smaller, partially visible cards for 'ORIGAMI' (with Origami paper), 'Nestle' (with Nestle products), 'LIFEWAY' (with Lifeway yogurt), 'KELLOGGS' (with Kellogg's cereal), and 'Dove' (with Dove chocolates).

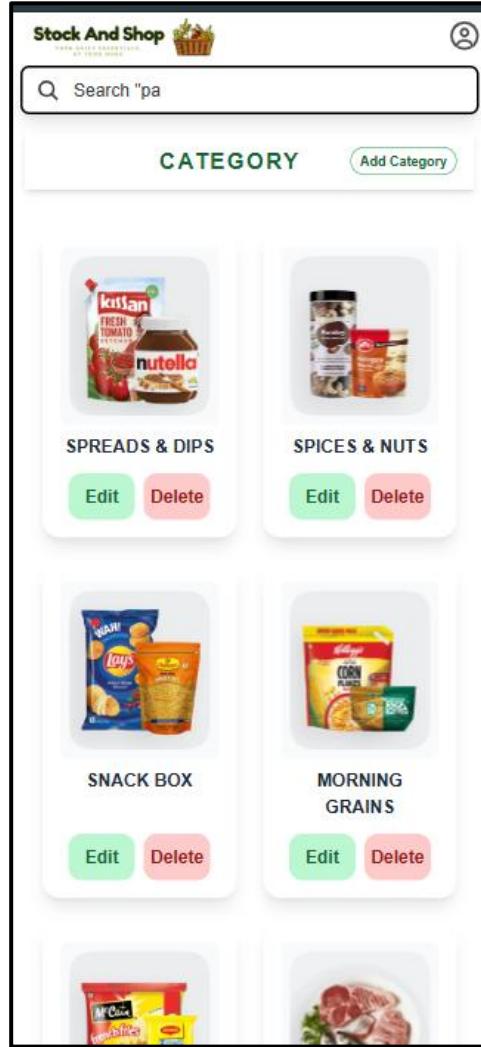


Figure 4.7: Admin – Category List Page

4.2.2 Sub-Category Management

Process:

- Admin selects a category.
- Adds sub-category name and image.
- Sub-category stored with reference to parent category.

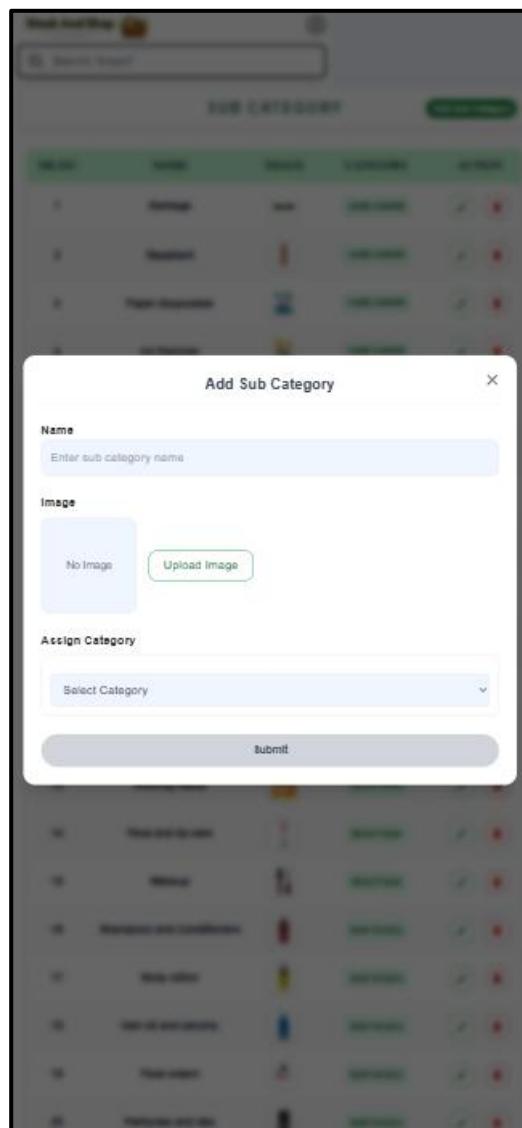
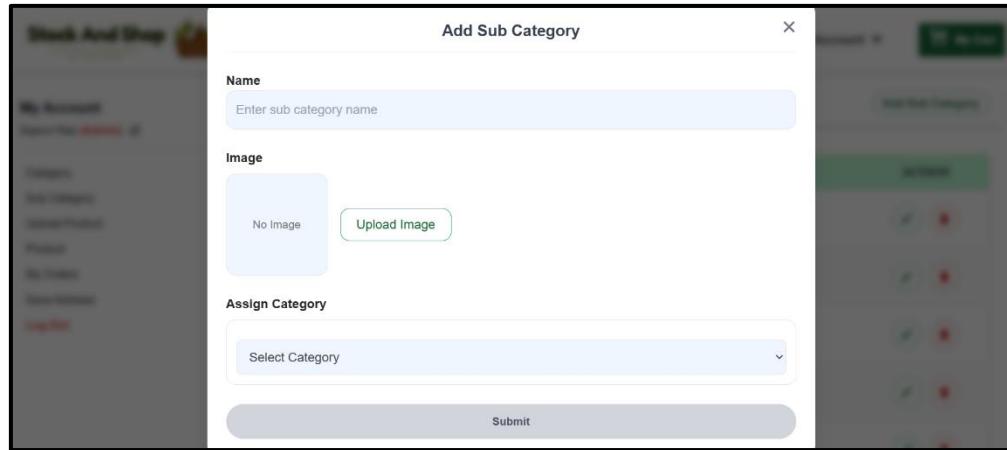


Figure 4.8: Admin – Add Sub-Category Page

My Account		SUB CATEGORY			
		Add Sub Category			
Category	Name	Image	Category	Action	
Sub Category	1 Garbage		HOME CORNER		
Upload Product	2 Repellant		HOME CORNER		
Product	3 Paper disposable		HOME CORNER		
My Orders	4 Air freshner		HOME CORNER		
Save Address	5 Oral care		LITTLE ONES		
Log Out					

SUB CATEGORY		Add Sub Category		
		Add Sub Category		
SR.NO	NAME	IMAGE	CATEGORY	ACTION
1	Garbage		HOME CORNER	
2	Repellant		HOME CORNER	
3	Paper disposable		HOME CORNER	
4	Air freshner		HOME CORNER	
5	Oral care		LITTLE ONES	
6	Skin care		LITTLE ONES	
7	Bath and Hair		LITTLE ONES	
8	Diapers and Wipes		LITTLE ONES	
9	Baby food and formula		LITTLE ONES	
10	Jams		SPREADS & DIPS	
11	Men's grooming		BEAUTY BOX	
12	Hair color		BEAUTY BOX	
13	Shaving needs		BEAUTY BOX	
14	Face and lip care		BEAUTY BOX	
15	Makeup		BEAUTY BOX	
16	Shampoos and Conditioners		BODY BASICS	
17	Body lotion		BODY BASICS	
18	Hair oil and serums		BODY BASICS	
19	Face cream		BODY BASICS	
20	Perfumes and deo		BODY BASICS	

Figure 4.9: Admin – Sub-Category List Page

4.2.3 Product Upload & Listing

Process:

- Admin selects Category → Sub-Category.
- Enters product details.
- Uploads product image.
- Product stored with category and sub-category references.

The screenshot shows a web application interface for 'Stock And Shop'. At the top, there's a logo with the text 'Stock And Shop' and 'YOUR DAILY ESSENTIALS, AT YOUR DOOR.' next to a small icon of a basket with vegetables. To the right of the logo is a search bar containing the placeholder 'Search "egg"' with a magnifying glass icon. Further right are 'Account' and 'My Cart' buttons. On the left, a sidebar titled 'My Account' lists options: 'Category', 'Sub Category', 'Upload Product' (which is currently selected), 'Product', 'My Orders', 'Save Address', and 'Log Out'. The main content area has a title 'UPLOAD PRODUCT'. It contains three input fields: 'Name' with placeholder 'Enter product name', 'Description' with placeholder 'Enter product description', and 'Image' which is a placeholder area with a cloud icon and the text 'Upload Image'.

The screenshot shows the 'UPLOAD PRODUCT' form. It includes fields for Name, Description, Image (with a placeholder 'Upload Image'), Category, Sub Category, Unit, and Number of Stock. Each field has a corresponding input or select dropdown box.

Field	Description
Name	Enter product name
Description	Enter product description
Image	Upload Image
Category	Select Category
Sub Category	Select Sub Category
Unit	Enter product unit
Number of Stock	Enter product stock

Figure 4.10: Admin – Upload Product Page

The screenshot displays the Admin dashboard with a sidebar for 'My Account' (Saanvi Rao (Admin)) and a main area for 'PRODUCT'. The 'PRODUCT' section shows a grid of items with their names, quantities, and edit/delete buttons. A search bar at the top right allows searching for specific products.

Product	Quantity	Action
Vadilal Gourmet...	1	Edit Delete
Vadilal Gourmet -...	1	Edit Delete
Havmor American M...	1	Edit Delete
Amul Vanilla Magic	1	Edit Delete
Amul Butterscot...	1	Edit Delete
Amul King Alphonso...	1	Edit Delete
(partial view)		

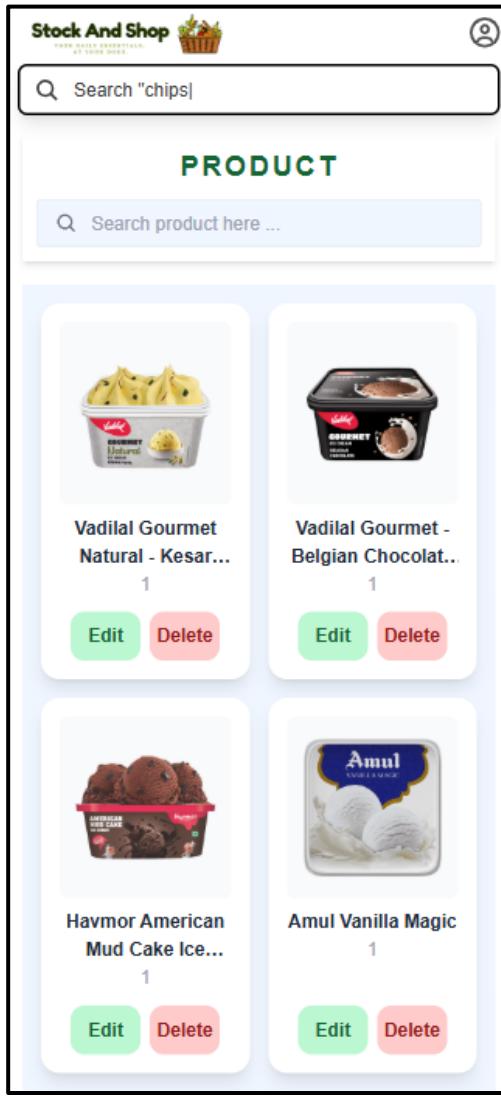


Figure 4.11: Admin – Product Page

4.2.4 Admin – Orders & Addresses

Admin can view:

- All customer orders
- Saved delivery addresses

Stock And Shop 

Search "paneer"

Account ▾ My Cart

My Account
Saanvi Rao (Admin) 

- Category
- Sub Category
- Upload Product
- Product
- My Orders
- Save Address
- [Log Out](#)

MY ORDERS

Order No: ORD-6937b7497946ef11a52ca72f



Britannia Treat
Croissant (Cocoa
Crème)

Order No: ORD-6937b26a7946ef11a52ca5b2



Protinex Health and
Nutritional Drink Mix

Order No: ORD-6937b1617946ef11a52ca522



Lotte Choco Pie

Order No: ORD-6937b0f77946ef11a52ca4b6



Oxylife Natural
Radiance 5 Crème
Bleach

Order No: ORD-6937ad447946ef11a52ca41b



Britannia Brown Bread

Order No: ORD-6937ad447946ef11a52ca41a



Saffola Cold Pressed
Sesame Oil

Stock And Shop 

Search "milk"

MY ORDERS

Order No: ORD-693c15e038e2c5cf0ea62380



Oxylife Natural Radiance 5
Crème Bleach

Order No: ORD-693c15af38e2c5cf0ea6231c



Britannia Treat Croissant (Cocoa
Crème)

Order No: ORD-6937b7497946ef11a52ca72f



Britannia Treat Croissant (Cocoa
Crème)

Order No: ORD-6937b26a7946ef11a52ca5b2



Protinex Health and Nutritional
Drink Mix

Order No: ORD-6937b1617946ef11a52ca522

Figure 4.12: Admin – My Orders Page

The screenshot shows the 'My Account' section of the 'Stock And Shop' website for an admin user named Saanvi Rao. The page title is 'ADDRESS'. It displays three saved addresses in cards:

- Rajmoti**
Vapi
Gujarat
India - 396195
8742198023
[Edit](#) [Delete](#)
- Neelkanth Anant**
Vapi
Gujarat
India - 396191
7582149680
[Edit](#) [Delete](#)
- Shyamal Vihar**
Vapi
Gujarat
India - 396195
7582149680
[Edit](#) [Delete](#)

A green button at the bottom center says 'Add Address'.

The screenshot shows the same 'ADDRESS' page as the desktop version, but it is displayed on a mobile phone screen. The layout is identical, showing the three saved addresses in cards with 'Edit' and 'Delete' buttons.

Figure 4.13: Admin – Saved Addresses Page

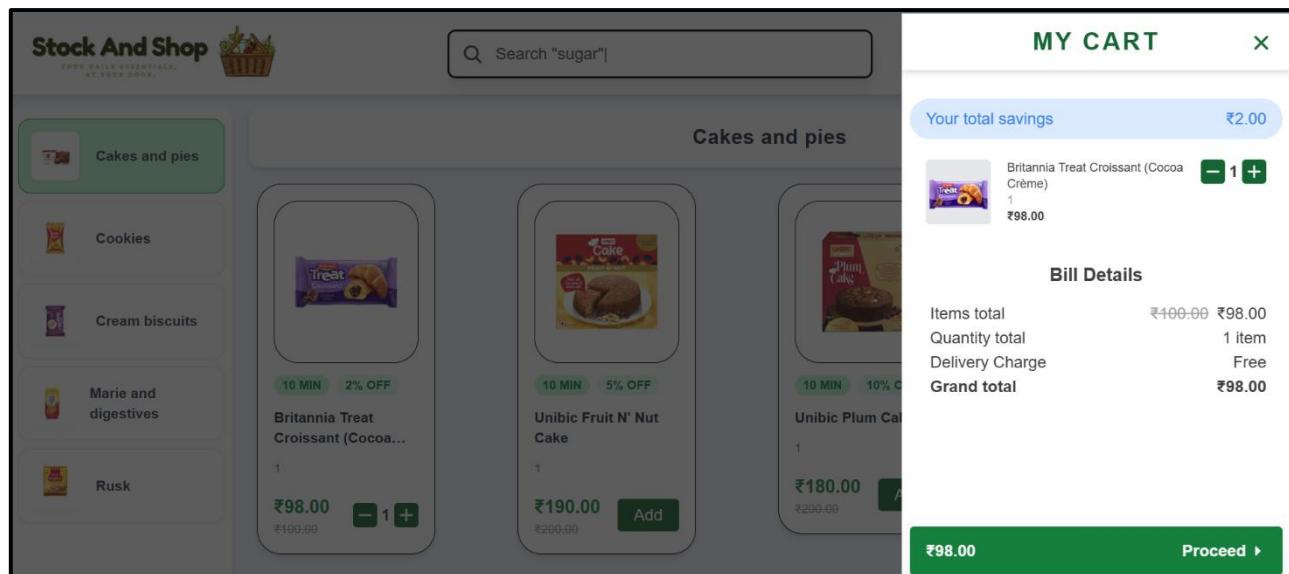
4.3 Process Logic of Payment Gateway (Stripe Integration)

The system integrates **Stripe Payment Gateway** to ensure secure, reliable, and verified online transactions.

4.3.1 Cart Management (Customer)

Process:

- Customer adds products to cart.
- Cart quantity updates dynamically.
- Cart data stored in MongoDB.



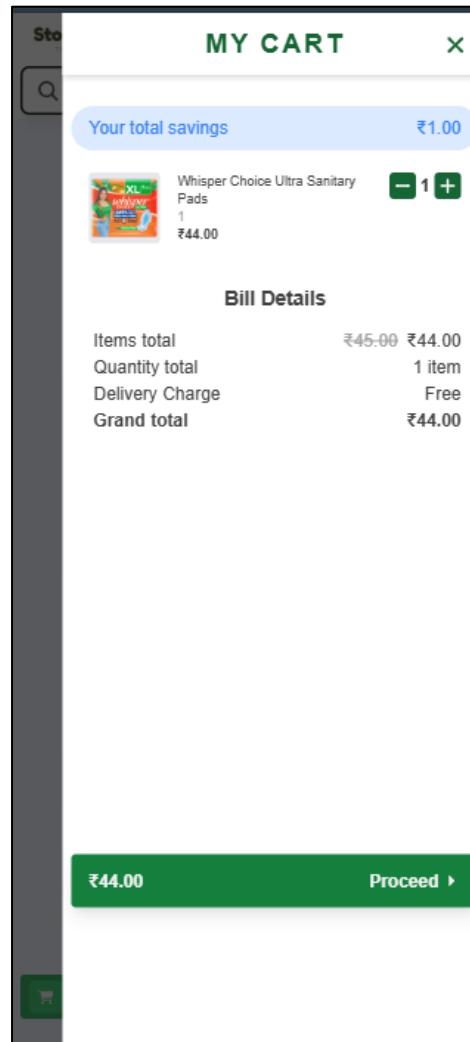


Figure 4.14: Customer – My Cart Page

4.3.2 Checkout Flow

Process:

- User clicks checkout.
- Backend prepares line items.
- Stripe Checkout Session is created.
- User redirected to Stripe payment page.

Stock And Shop YOUR DAILY ESSENTIALS AT YOUR DOOR 

Search "chips"

Account ▾ 1 Items ₹98.00

CHOOSE YOUR ADDRESS

- Rajmoti
Vapi
 Gujarat
India - 396195
8742198023
- Neelkanth Anant
Vapi
 Gujarat
India - 396191
7582149680
- Shyamal Vihar
Vapi

SUMMARY

Items total	₹98.00
Quantity total	1 item
Delivery Charge	Free
Grand total	₹98.00

Online Payment

Cash on Delivery

Stock And Shop YOUR DAILY ESSENTIALS AT YOUR DOOR 

Search "curd"

Vapi
 Gujarat
India - 396191
7582149680

Shyamal Vihar
Vapi
 Gujarat
India - 396195
7582149680

Add address

SUMMARY

Items total	₹44.00
Quantity total	1 item
Delivery Charge	Free
Grand total	₹44.00

Online Payment

Cash on Delivery

© All Rights Reserved 2025.

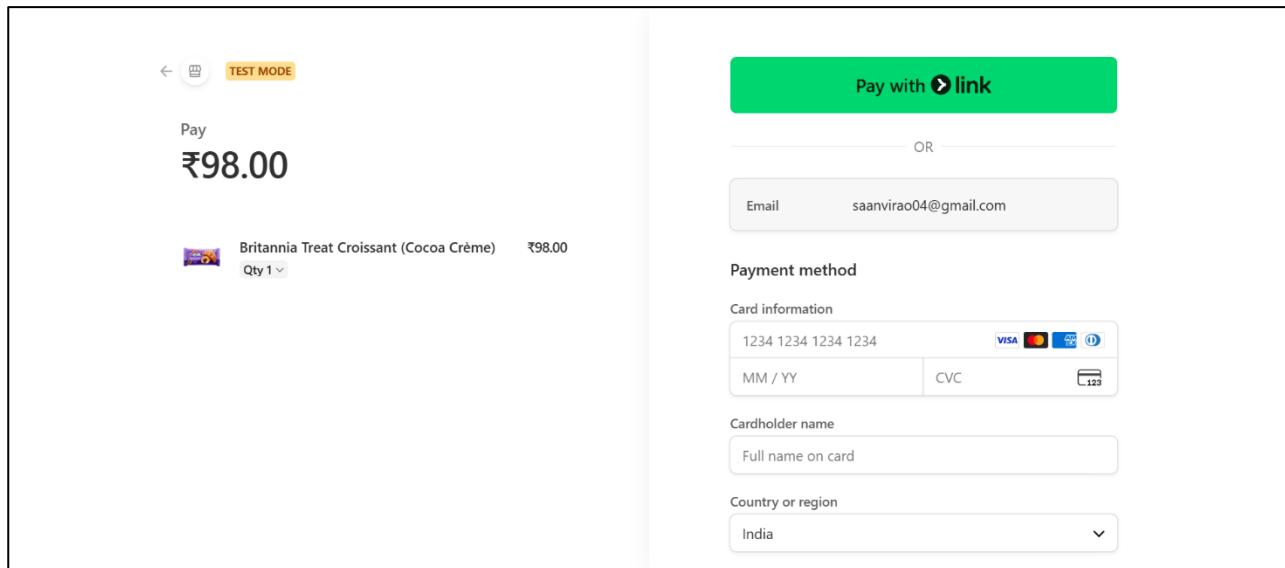
Figure 4.15: Checkout Page

4.3.3 Secure Webhook Handler Implementation

After payment completion, Stripe sends a webhook to the backend.

Process:

- Stripe verifies payment.
- Sends event to /api/webhook/stripe.
- Backend validates webhook signature.
- Metadata (userId, cartId) is extracted.
- Order Fulfillment Logic is triggered.



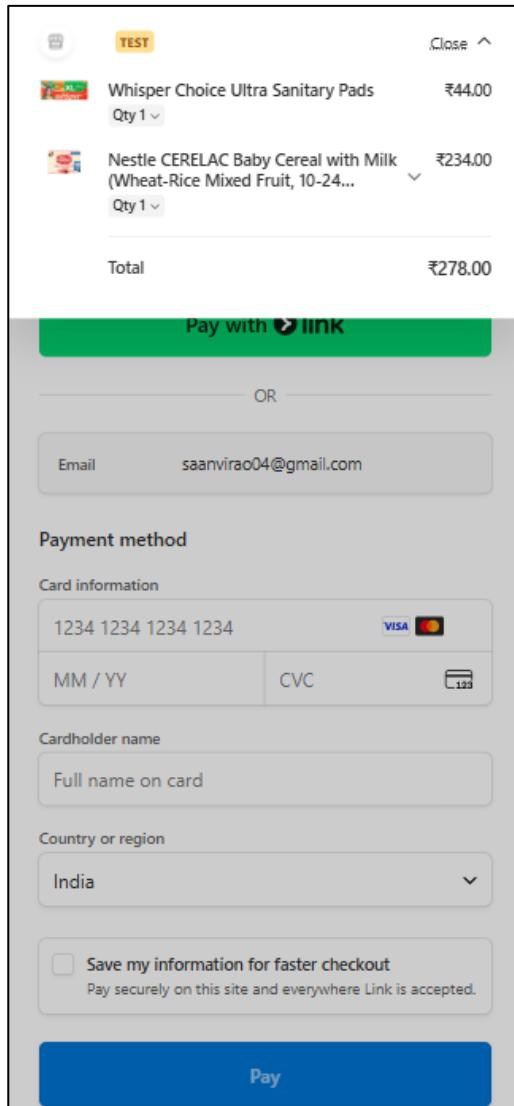


Figure 4.16: Stripe Page

4.3.4 Order Fulfillment & Cart Clearance Logic

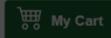
Process:

- Order record created in database.
- Product stock is updated.
- Cart items are cleared.
- Order confirmation response sent to user.



Search "bread"

Account ▾



Payment Successful

[Go To Home](#)

© All Rights Reserved 2025.



Search "bread"

Account ▾



Order Cancelled

[Go To Home](#)

© All Rights Reserved 2025.



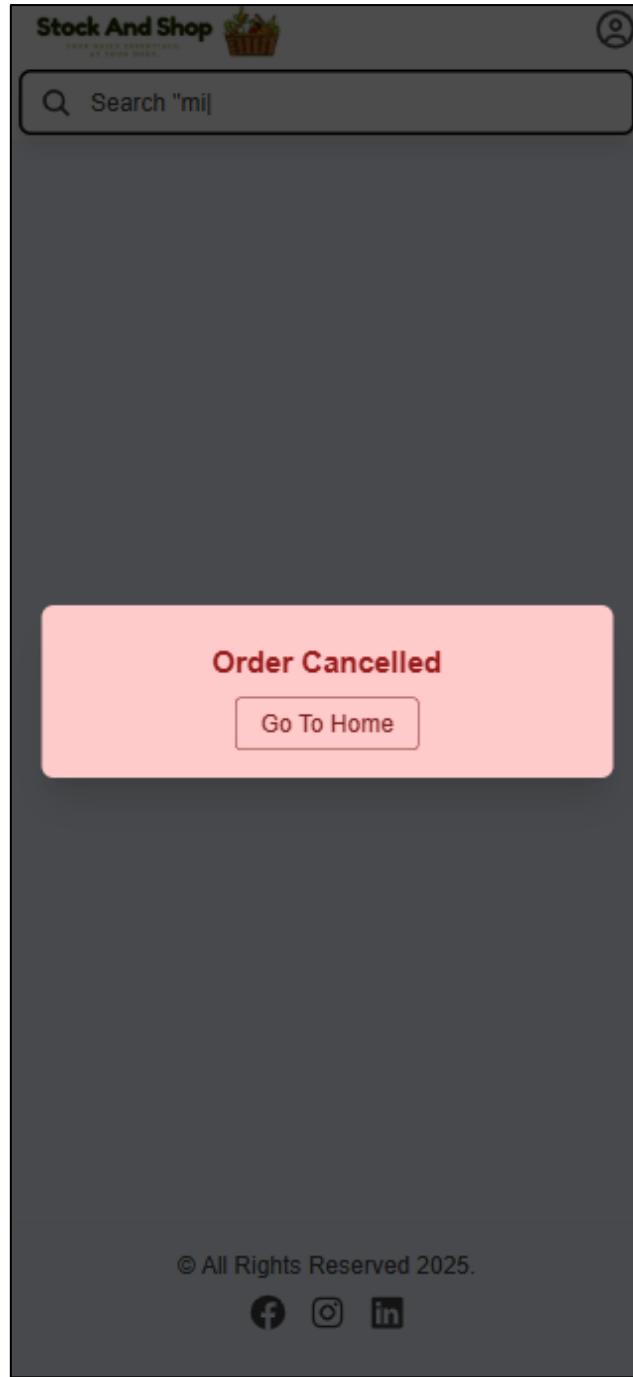
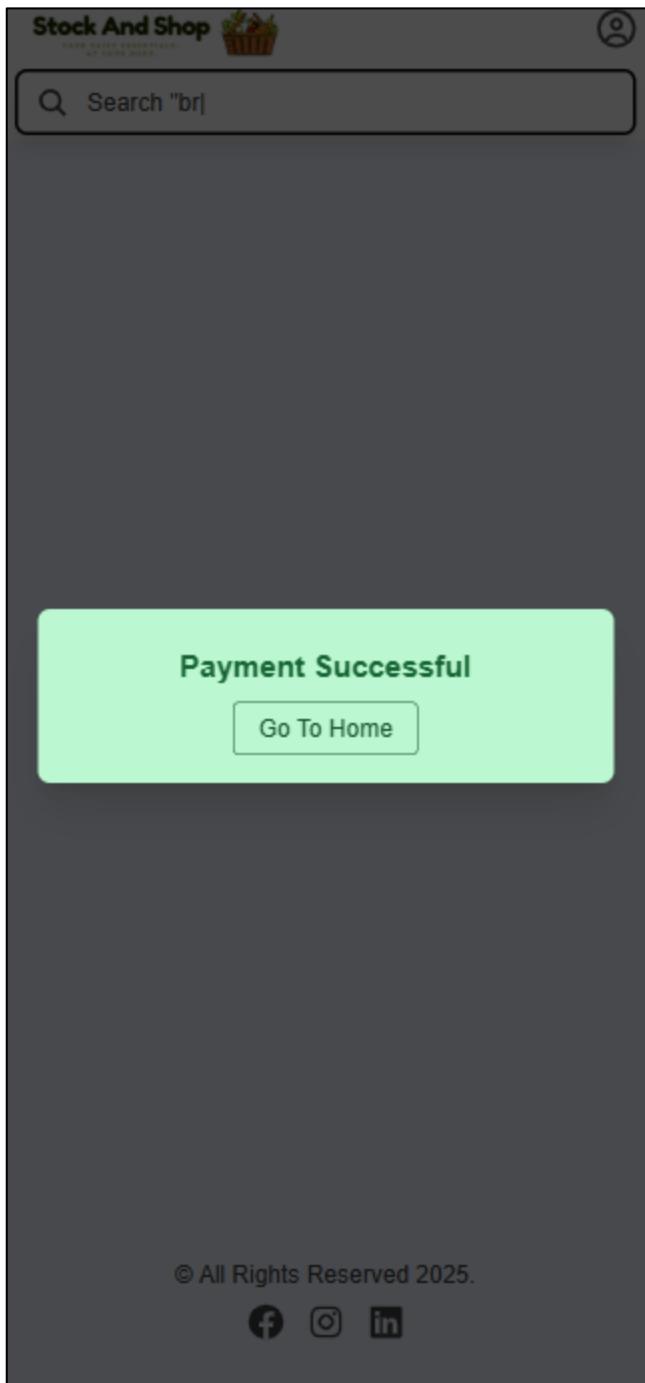


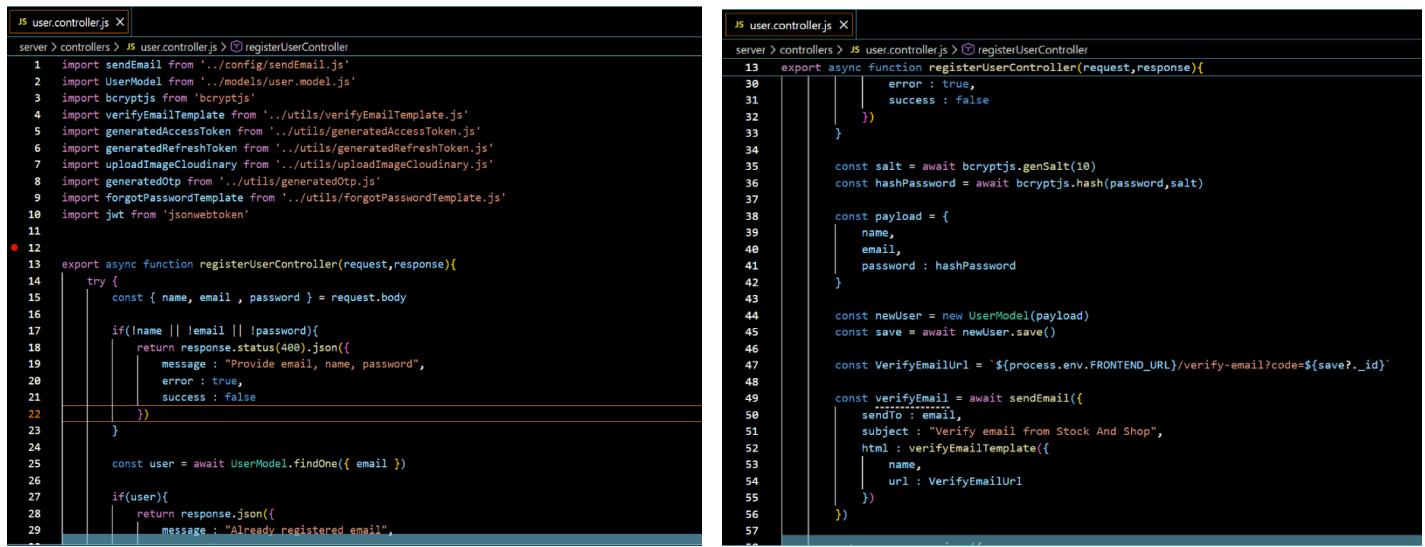
Figure 4.17: Order Confirmation Page

4.4 Code Implementation Details

This section highlights the core code files used in implementing authentication, inventory management, cart operations, and payment processing. Only essential files are included to maintain clarity.

4.4.1 Authentication Code

- **user.controller.js** – Handles user registration, login, and OTP-based password reset
- **auth.js** – JWT authentication middleware



The image shows a code editor with two tabs open: `user.controller.js` and `auth.js`. Both tabs are under the path `server > controllers > JS user.controller.js > registerUserController`.

user.controller.js:

```
JS user.controller.js ×
server > controllers > JS user.controller.js > registerUserController
1 import sendEmail from '../config/sendEmail.js'
2 import UserModel from '../models/user.model.js'
3 import bcryptjs from 'bcryptjs'
4 import verifyEmailTemplate from '../utils/verifyEmailTemplate.js'
5 import generatedAccessToken from '../utils/generatedAccessToken.js'
6 import generatedRefreshToken from '../utils/generatedRefreshToken.js'
7 import uploadImageCloudinary from '../utils/uploadImageCloudinary.js'
8 import generatedOtp from '../utils/generatedOtp.js'
9 import forgotPasswordTemplate from '../utils/forgotPasswordTemplate.js'
10 import jwt from 'jsonwebtoken'

● 11
12
13 export async function registerUserController(request, response){
14     try {
15         const { name, email, password } = request.body
16
17         if(!name || !email || !password){
18             return response.status(400).json({
19                 message : "Provide email, name, password",
20                 error : true,
21                 success : false
22             })
23
24
25         const user = await UserModel.findOne({ email })
26
27         if(user){
28             return response.json({
29                 message : "Already registered email",
30             })
31
32     }
33
34
35     const salt = await bcryptjs.genSalt(10)
36     const hashPassword = await bcryptjs.hash(password,salt)
37
38     const payload = {
39         name,
40         email,
41         password : hashPassword
42     }
43
44     const newUser = new UserModel(payload)
45     const save = await newUser.save()
46
47     const VerifyEmailUrl = `${process.env.FRONTEND_URL}/verify-email?code=${save._id}`
48
49     const verifyEmail = await sendEmail({
50         .....
51         sendto : email,
52         subject : "Verify email from Stock And Shop",
53         html : verifyEmailTemplate({
54             name,
55             url : VerifyEmailUrl
56         })
57     })
58
59 }
```

auth.js:

```
JS user.controller.js ×
server > controllers > JS user.controller.js > registerUserController
13 export async function registerUserController(request, response){
30     try {
31         const { name, email, password } = request.body
32         const salt = await bcryptjs.genSalt(10)
33         const hashPassword = await bcryptjs.hash(password,salt)
34
35         const payload = {
36             name,
37             email,
38             password : hashPassword
39         }
40
41         const newUser = new UserModel(payload)
42         const save = await newUser.save()
43
44         const VerifyEmailUrl = `${process.env.FRONTEND_URL}/verify-email?code=${save._id}`
45
46         const verifyEmail = await sendEmail({
47             .....
48             sendto : email,
49             subject : "Verify email from Stock And Shop",
50             html : verifyEmailTemplate({
51                 name,
52                 url : VerifyEmailUrl
53             })
54         })
55
56     } catch (error) {
57         return response.json({
58             error : true,
59             success : false
60         })
61     }
62
63 }
```

JS user.controller.js X

```
server > controllers > JS user.controller.js > registerUserController
13  export async function registerUserController(request,response){
58      return response.json({
59          message : "User registered successfully",
60          error : false,
61          success : true,
62          data : save
63      })
64
65    } catch (error) {
66      return response.status(500).json({
67          message : error.message || error,
68          error : true,
69          success : false
70      })
71  }
72}
73
74 export async function verifyEmailController(request,response){
75  try {
76      const { code } = request.body
77
78      const user = await UserModel.findOne({ _id : code })
79
80      if(!user){
81          return response.status(400).json({
82              message : "Invalid code",
83              error : true,
84              success : false
85          })
86    }
87
88    const updateUser = await UserModel.updateOne({ _id : code },{
89        verify_email : true
90    })
91
92    return response.json({
93        message : "Email verification successful.",
94        success : true,
95        error : false
96    })
97  } catch (error) {
98    return response.status(500).json({
99        message : error.message || error,
100       error : true,
101       success : true
102    })
103  }
104}
105
106 //login controller
107 export async function loginController(request,response): Promise<any>
108 try {
109     const { email , password } = request.body
110
111     if(!email || !password){
112         return response.status(400).json({
113             message : "Provide email, password",
114             error : true,
115             success : false
116         })
117     }
118
119     const user = await UserModel.findOne({ email })
120
121     if(!user){
122         return response.status(400).json({
123             message : "User not registered",
124             error : true,
125             success : false
126         })
127     }
128
129     if(user.status !== "Active"){
130         return response.status(400).json({
131             message : "Contact to the Admin",
132             error : true,
133             success : false
134         })
135     }
136
137     const checkPassword = await bcryptjs.compare(password,user.password)
138
139 }
```

JS user.controller.js X

```
server > controllers > JS user.controller.js > verifyEmailController
74  export async function verifyEmailController(request,response){
86    }
87
88    const updateUser = await UserModel.updateOne({ _id : code },{
89        verify_email : true
90    })
91
92    return response.json({
93        message : "Email verification successful.",
94        success : true,
95        error : false
96    })
97  } catch (error) {
98    return response.status(500).json({
99        message : error.message || error,
100       error : true,
101       success : true
102    })
103  }
104}
105
106 //login controller
107 export async function loginController(request,response): Promise<any>
108 try {
109     const { email , password } = request.body
110
111     if(!email || !password){
112         return response.status(400).json({
113             message : "Provide email, password",
114             error : true,
115             success : false
116         })
117     }
118
119     const user = await UserModel.findByIdAndUpdate(user._id,{
120         last_login_date : new Date()
121     })
122
123     const accesstoken = await generatedAccessToken(user._id)
124     const refreshToken = await generatedRefreshToken(user._id)
125
126     const updateuser = await UserModel.findByIdAndUpdate(user._id,{
127         last_login_date : new Date()
128     })
129
130     const cookiesOption = {
131         httpOnly : true,
132         secure : true,
133         sameSite : "None"
134     }
135
136     response.cookie('accessToken',accesstoken,cookiesOption)
137     response.cookie('refreshToken',refreshToken,cookiesOption)
138
139     return response.json({
140         message : "Login successfully",
141         error : false,
142         success : true,
143         data : {
144             accessToken : accesstoken,
145             refreshToken : refreshToken
146         }
147     })
148
149 }
```

JS user.controller.js X

```
server > controllers > JS user.controller.js > verifyEmailController
107  export async function loginController(request,response){
112    if(!email || !password){
113        return response.status(400).json({
114            message : "Provide email, password",
115            error : true,
116            success : false
117        })
118    }
119
120    const user = await UserModel.findOne({ email })
121
122    if(!user){
123        return response.status(400).json({
124            message : "User not registered",
125            error : true,
126            success : false
127        })
128    }
129
130    if(user.status !== "Active"){
131        return response.status(400).json({
132            message : "Contact to the Admin",
133            error : true,
134            success : false
135        })
136    }
137
138    const checkPassword = await bcryptjs.compare(password,user.password)
139 }
```

JS user.controller.js X

```
server > controllers > JS user.controller.js > verifyEmailController
107  export async function loginController(request,response): Promise<any>
108 try {
109     const { email , password } = request.body
110
111     if(!email || !password){
112         return response.status(400).json({
113             message : "Provide email, password",
114             error : true,
115             success : false
116         })
117     }
118
119     const user = await UserModel.findByIdAndUpdate(user._id,{
120         last_login_date : new Date()
121     })
122
123     const accesstoken = await generatedAccessToken(user._id)
124     const refreshToken = await generatedRefreshToken(user._id)
125
126     const updateuser = await UserModel.findByIdAndUpdate(user._id,{
127         last_login_date : new Date()
128     })
129
130     const cookiesOption = {
131         httpOnly : true,
132         secure : true,
133         sameSite : "None"
134     }
135
136     response.cookie('accessToken',accesstoken,cookiesOption)
137     response.cookie('refreshToken',refreshToken,cookiesOption)
138
139     return response.json({
140         message : "Login successfully",
141         error : false,
142         success : true,
143         data : {
144             accessToken : accesstoken,
145             refreshToken : refreshToken
146         }
147     })
148
149 }
```

JS user.controller.js X

```

server > controllers > JS user.controller.js > ✎ verifyEmailController
187 ✓ export async function loginController(request,response){
188     try {
189         const { accessToken, refreshToken } = JSON.parse(request.body);
190         const user = await UserModel.findByIdAndUpdate(
191             { _id: request.user._id },
192             { $push: { tokens: { type: "refreshToken", token: refreshToken } } }
193         );
194         const token = jwt.sign({ _id: user._id }, process.env.JWT_SECRET);
195         res.cookie("accessToken", token, {
196             httpOnly: true,
197             secure: true,
198             sameSite: "None"
199         });
200         res.cookie("refreshToken", refreshToken, {
201             httpOnly: true,
202             secure: true,
203             sameSite: "None"
204         });
205         return res.json({
206             message: "Login successfully",
207             error: false,
208             success: true
209         });
210     } catch (error) {
211         return res.status(500).json({
212             message: error.message || error,
213             error: true,
214             success: false
215         });
216     }
217 }
218
219 //logout controller
220 export async function logoutController(request,response){
221     try {
222         const userId = request.userId //middleware
223
224         const cookiesOption = {
225             httpOnly: true,
226             secure: true,
227             sameSite: "None"
228         }
229
230         response.clearCookie("accessToken",cookiesOption)
231         response.clearCookie("refreshToken",cookiesOption)
232     } catch (error) {
233         return res.status(500).json({
234             message: error.message || error,
235             error: true,
236             success: false
237         });
238     }
239 }
240
241 //upload user avatar
242 export async function uploadAvatar(request,response){
243     try {
244         const userId = request.userId // auth middleware
245         const image = request.file // multer middleware
246
247         const upload = await uploadImageCloudinary(image)
248
249         const updateUser = await UserModel.findByIdAndUpdate(userId,{
250             ...request.body,
251             avatar: upload.url
252         })
253
254         if(updateUser){
```

JS user.controller.js X

```

server > controllers > JS user.controller.js > ✎ verifyEmailController
183 ✓ export async function logoutController(request,response){
184     try {
185         const removeRefreshToken = await UserModel.findByIdAndUpdate(
186             { _id: request.user._id },
187             { $pull: { tokens: { type: "refreshToken", token: "" } } }
188         );
189
190         return res.json({
191             message: "Logout successfully",
192             error: false,
193             success: true
194         });
195     } catch (error) {
196         return res.status(500).json({
197             message: error.message || error,
198             error: true,
199             success: false
200         });
201     }
202 }
203
204 //upload user avatar
205 export async function uploadAvatar(request,response){
206     try {
207         const userId = request.userId // auth middleware
208         const image = request.file // multer middleware
209
210         const upload = await uploadImageCloudinary(image)
211
212         const updateUser = await UserModel.findByIdAndUpdate(userId,{
213             ...request.body,
214             avatar: upload.url
215         })
216
217         if(updateUser){
```

JS user.controller.js X

```

server > controllers > JS user.controller.js > ✎ verifyEmailController
215 ✓ export async function uploadAvatar(request,response){
216     try {
217         const { _id, avatar } = JSON.parse(request.body);
218
219         const updateAvatar = await UserModel.findByIdAndUpdate(
220             { _id },
221             { avatar }
222         );
223
224         if(updateAvatar){
```

JS user.controller.js X

```

server > controllers > JS user.controller.js > ✎ verifyEmailController
246 ✓ export async function updateUserDetails(request,response){
247     try {
248         const { name, email, mobile, password } = request.body;
249
250         let hashPassword = "";
251
252         if(password){
```

```
JS user.controller.js X
server > controllers > JS user.controller.js > ⏎ verifyEmailController
283   export async function forgotPasswordController(request, response) {
284     try {
285       const { email } = request.body
286
287       const user = await UserModel.findOne({ email })
288
289       if(!user){
290         return response.status(400).json({
291           message : "Email doesn't exist",
292           error : true,
293           success : false
294         })
295       }
296
297       const otp = generatedOtp()
298       const expireTime = new Date() + 60 * 60 * 1000 // 1hr
299
300       const update = await UserModel.findByIdAndUpdate(user._id, {
301         forgot_password_otp : otp,
302         forgot_password_expiry : new Date(expireTime).toISOString()
303       })
304
305       await sendEmail({
306         sendTo : email,
307         subject : "Stock And Shop Password Reset Request",
308         html : forgotPasswordTemplate({
309           name : user.name,
310           otp : otp
311         })
312       })
313     }
314   }
315 }
```

```
JS user.controller.js X
server > controllers > JS user.controller.js > ✎ verifyEmailController
283   export async function forgotPasswordController(request,response) {
312     })
313
314     return response.json({
315       message : "Please check your email",
316       error : false,
317       success : true
318     })
319
320   } catch (error) {
321     return response.status(500).json({
322       message : error.message || error,
323       error : true,
324       success : false
325     })
326   }
327 }
328
329 //verify forgot password otp
330 export async function verifyForgotPasswordOtp(request,response){
331   try {
332     const { email , otp } = request.body
333
334     if(!email || !otp){
335       return response.status(400).json({
336         message : "Required fields: Email, OTP",
337         error : true,
338         success : false
339       })
340     }
341   }
342 }
```

```
JS user.controller.js X
server > controllers > JS user.controller.js > verifyEmailController
330  export async function verifyForgotPasswordOtp(request, response){
340      }
341
342      const user = await UserModel.findOne({ email })
343
344      if(!user){
345          return response.status(400).json({
346              message : "Email doesn't exist",
347              error : true,
348              success : false
349          })
350      }
351
352      const currentTime = new Date().toISOString()
353
354      if(user.forgot_password_expiry < currentTime ){
355          return response.status(400).json({
356              message : "The OTP has expired.",
357              error : true,
358              success : false
359          })
360      }
361
362      if(otp !== user.forgot_password_otp){
363          return response.status(400).json({
364              message : "Invalid OTP",
365              error : true,
366              success : false
367          })
368      }
369
370      const token = jwt.sign({ email }, process.env.JWT_SECRET, { expiresIn: '15m' })
371
372      const options = {
373          method: 'PUT',
374          url: `http://localhost:3001/api/v1/users/${user._id}/verify-forgot-password`,
375          headers: {
376              'Content-Type': 'application/json'
377          },
378          data: {
379              token
380          }
381      }
382
383      axios(options)
384          .then((res) => {
385              if(res.data.error) {
386                  return response.status(400).json(res.data)
387              }
388
389              response.status(200).json(res.data)
390          })
391
392      user.forgot_password_otp = null
393      user.forgot_password_expiry = null
394
395      await user.save()
396
397      response.status(200).json({
398          message : "OTP verified successfully",
399          error : false,
400          success : true
401      })
402  }
```

```
JS user.controller.js X
server > controllers > JS user.controller.js > verifyEmailController
330  export async function verifyForgotPasswordOtp(request,response){
373      const updateUser = await UserModel.findByIdAndUpdate(user?._id,{
374          forgot_password_otp : "",
375          forgot_password_expiry : ""
376      })
377
378      return response.json({
379          message : "OTP Verified Successfully",
380          error : false,
381          success : true
382      })
383
384  } catch (error) {
385      return response.status(500).json({
386          message : error.message || error,
387          error : true,
388          success : false
389      })
390  }
391 }
392
393 //reset the password
394 export async function resetpassword(request,response){
395     try {
396         const { email , newPassword, confirmPassword } = request.body
397
398         if(!email || !newPassword || !confirmPassword){
399             return response.status(400).json({
400                 message : "Please provide all required fields: Email, New Password, and Confirm Password."
401             })
402         }
403
404         const user = await UserModel.findOne({ email })
405
406         if(!user){
407             return response.status(400).json({
408                 message : "Email doesn't exist",
409                 error : true,
410                 success : false
411             })
412         }
413
414         if(newPassword !== confirmPassword){
415             return response.status(400).json({
416                 message : "New Password and Confirm Password must match.",
417                 error : true,
418                 success : false,
419             })
420         }
421
422         const salt = await bcryptjs.genSalt(10)
423         const hashPassword = await bcryptjs.hash(newPassword,salt)
424
425         const update = await UserModel.findOneAndUpdate(user._id,{
426             password : hashPassword
427         })
428     }
429 }
```

```
JS user.controller.js X
server > controllers > JS user.controller.js > verifyEmailController
394  export async function resetpassword(request,response){
401      })
402
403
404     const user = await UserModel.findOne({ email })
405
406     if(!user){
407         return response.status(400).json({
408             message : "Email doesn't exist",
409             error : true,
410             success : false
411         })
412     }
413
414     if(newPassword !== confirmPassword){
415         return response.status(400).json({
416             message : "New Password and Confirm Password must match.",
417             error : true,
418             success : false,
419         })
420     }
421
422     const salt = await bcryptjs.genSalt(10)
423     const hashPassword = await bcryptjs.hash(newPassword,salt)
424
425     const update = await UserModel.findOneAndUpdate(user._id,{
426         password : hashPassword
427     })
428 }
```

```
JS user.controller.js X
server > controllers > JS user.controller.js > verifyEmailController
394  export async function resetpassword(request,response){
429      return response.json({
430          message : "Password updated successfully.",
431          error : false,
432          success : true
433      })
434
435  } catch (error) {
436      return response.status(500).json({
437          message : error.message || error,
438          error : true,
439          success : false
440      })
441  }
442 }
443
444 //refresh token controller
445 export async function refreshToken(request,response){
446     try {
447         const refreshToken = request.cookies.refreshToken || request?.headers?.authorization?.split(" ")[1]
448
449         if(!refreshToken){
450             return response.status(401).json({
451                 message : "Invalid token",
452                 error : true,
453                 success : false
454             })
455         }
456     }
457 }
```

```

JS user.controller.js X
server > controllers > JS user.controller.js > verifyEmailController
445  export async function refreshToken(request,response){
457    const verifyToken = await jwt.verify(refreshToken,process.env.SECRET_KEY_REFRESH_TOKEN)
458
459    if(!verifyToken){
460      return response.status(401).json({
461        message : "Token has expired",
462        error : true,
463        success : false
464      })
465
466
467    const userId = verifyToken._id
468
469    const newAccessToken = await generatedAccessToken(userId)
470
471    const cookiesOption = {
472      httpOnly : true,
473      secure : true,
474      sameSite : "None"
475    }
476
477    response.cookie('accessToken',newAccessToken,cookiesOption)
478
479    return response.json({
480      message : "New Access token generated",
481      error : false,
482      success : true,
483      data : {
484        accessToken : newAccessToken
485      }
486    })
487
488
489  } catch (error) {
490    return response.status(500).json({
491      message : error.message || error,
492      error : true,
493      success : false
494    })
495  }
496
497
498 //get login user details
499 export async function userDetails(request,response){
500   try {
501     const userId = request.userId
502
503     console.log(userId)
504
505     const user = await UserModel.findById(userId).select('-password -refresh_token')
506
507     return response.json({
508       message : 'user details',
509       data : user,
510       error : false,
511       success : true
512     })
513   } catch (error) {
514
515   }
516
517
518
519
520 }

```

```

JS user.controller.js X
server > controllers > JS user.controller.js > verifyEmailController
499  export async function userDetails(request,response){
513    } catch (error) {
514      return response.status(500).json({
515        message : "Something is wrong",
516        error : true,
517        success : false
518      })
519    }
520  }

```

```

JS auth.js X
server > middleware > JS auth.js > ...
1  import jwt from 'jsonwebtoken'
2  const auth = async(request,response,next)=>{
3    try {
4      const token = request.cookies.accessToken || request?.headers?.authorization?.split(" ")[1]
5
6      if(!token){
7        return response.status(401).json({
8          message : "Provide token"
9        })
10
11      const decode = await jwt.verify(token,process.env.SECRET_KEY_ACCESS_TOKEN)
12      if(!decode){
13        return response.status(401).json({
14          message : "unauthorized access",
15          error : true,
16          success : false
17        })
18      }
19      request.userId = decode.id
20      next()
21    } catch (error) {
22      return response.status(500).json({
23        message : "You have not logged in",//error.message || error,
24        error : true,
25        success : false
26      })
27    }
28  }
29  export default auth

```

Figure 4.18: Authentication Controller and JWT Middleware

4.4.2 Product & Inventory Code

- **category.controller.js** – Category CRUD operations
- **subCategory.controller.js** – Sub-category management
- **product.controller.js** – Product upload and stock updates

```
JS category.controller.js X
server > controllers > JS category.controller.js > [?] deleteCategoryController
1 import CategoryModel from "../models/category.model.js";
2 import SubCategoryModel from "../models/subCategory.model.js";
3 import ProductModel from "../models/product.model.js";
4 export const AddCategoryController = async(request,response)=>{
5     try {
6         const { name , image } = request.body
7
8         if(!name || !image){
9             return response.status(400).json({
10                 message : "Enter required fields",
11                 error : true,
12                 success : false
13             })
14         }
15         const addCategory = new CategoryModel({
16             name,
17             image
18         })
19         const saveCategory = await addCategory.save()
20         if(!saveCategory){
21             return response.status(500).json({
22                 message : "Not Created",
23                 error : true,
24                 success : false
25             })
26         }
27         return response.json({
28             message : "Add Category",
29             data : saveCategory,
30         })
31     }
32     catch (error) {
33         return response.status(500).json({
34             message : error.message || error,
35             error : true,
36             success : false
37         })
38     }
39 }
40 }
```

```
JS category.controller.js X
server > controllers > JS category.controller.js > [?] deleteCategoryController
4 export const AddCategoryController = async(request,response)=>{
5     try {
6         const data : saveCategory,
7             success : true,
8             error : false
9     }
10    catch (error) {
11        return response.status(500).json({
12            message : error.message || error,
13            error : true,
14            success : false
15        })
16    }
17 }
18 export const getCategoryController = async(request,response)=>{
19     try {
20         const data = await CategoryModel.find().sort({ createdAt : -1 })
21         return response.json({
22             data : data,
23             error : false,
24             success : true
25         })
26     }
27     catch (error) {
28         return response.status(500).json({
29             message : error.message || error,
30             error : true,
31             success : false
32         })
33     }
34 }
```

```
JS category.controller.js X
server > controllers > JS category.controller.js > [?] deleteCategoryController
57 export const updateCategoryController = async(request,response)=>{
58     try {
59         const { _id ,name , image } = request.body
60         const update = await CategoryModel.updateOne({
61             _id : _id
62         },{
63             name,
64             image
65         })
66         return response.json({
67             message : "Category updated",
68             success : true,
69             error : false,
70             data : update
71         })
72     }
73     catch (error) {
74         return response.status(500).json({
75             message : error.message || error,
76             error : true,
77             success : false
78         })
79     }
80 }
```

```
JS category.controller.js X
server > controllers > JS category.controller.js > [?] deleteCategoryController
80 export const deleteCategoryController = async(request,response)=>{
81     try {
82         const { _id } = request.body
83         const checkSubCategory = await SubCategoryModel.find({
84             category : {
85                 "$in" : [ _id ]
86             }
87         }).countDocuments()
88         const checkProduct = await ProductModel.find({
89             category : {
90                 "$in" : [ _id ]
91             }
92         }).countDocuments()
93         if(checkSubCategory > 0 || checkProduct > 0){
94             return response.status(400).json({
95                 message : "Cannot delete. The category is currently in use",
96                 error : true,
97                 success : false
98             })
99         }
100        const deleteCategory = await CategoryModel.deleteOne({ _id : _id })
101        return response.json({
102             message : "Category deleted successfully",
103             data : deleteCategory,
104             error : false,
105             success : true
106         })
107     }
108     catch (error) {
109         return response.status(500).json({
110             message : error.message || error,
111             error : true,
112             success : false
113         })
114     }
115 }
```

```
108     return response.status(500).json({
109         message : error.message || error,
110         success : false,
111         error : true
112     })
113 }
114 }
```

```
JS product.controller.js X
server > controllers > JS product.controller.js > ...
1 import ProductModel from "./models/product.model.js";
2
3 export const createProductController = async (request, response) => {
4   try {
5     const {
6       name,
7       image,
8       category,
9       subCategory,
10      unit,
11      stock,
12      price,
13      discount,
14      description,
15      more_details,
16    } = request.body
17
18    if (!name || !image[0] || !category[0] || !subCategory[0] || !unit || !price || !description) {
19      return response.status(400).json({
20        message: "Enter required fields",
21        error: true,
22        success: false
23      })
24    }
25
26    const product = new ProductModel({
27      name,
28      image,
29      category,
```

```
JS product.controller.js X
server > controllers > JS product.controller.js > ...
  3  export const createProductController = async(request,response)=>{
  4    const product = new ProductModel({
  5      subCategory,
  6      unit,
  7      stock,
  8      price,
  9      discount,
 10      description,
 11      more_details,
 12    })
 13    const saveProduct = await product.save()
 14
 15    return response.json({
 16      message : "Product Created Successfully",
 17      data : saveProduct,
 18      error : false,
 19      success : true
 20    })
 21
 22  } catch (error) {
 23    return response.status(500).json({
 24      message : error.message || error,
 25      error : true,
 26      success : false
 27    })
 28  }
 29
 30}
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56  export const getProductsController = async(request,response)=>{
```

```
js product.controller.js x
server > controllers > JS productcontroller.js > ...
56 export const getProductController = async(request,response)=>{
57   try {
58     let { page, limit, search } = request.body
59
60     if(!page){
61       |   page = 1
62     }
63
64     if(!limit){
65       |   limit = 10
66     }
67
68     const query = search ? {
69       $text : {
70         |   $search : search
71       }
72     } : {}
73
74
75     const skip = (page - 1) * limit
76
77     const [data,totalCount] = await Promise.all([
78       ProductModel.find(query).sort({createdAt : -1 }).skip(skip).limit(limit).populate('category subCategory')
79       ProductModel.countDocuments(query)
80     ])
81
82
83     return response.json({
84       message : "Product data",
85       error : false,
```

```
JS product.controller.js ×
server > controllers > JS product.controller.js ...  

56 export const getProductController = async(request,response)=>{
  85   success : true,
  86   totalCount : totalCount,
  87   totalNoPage : Math.ceil( totalCount / limit),
  88   data : data
  89 }
  90 } catch (error) {
  91   return response.status(500).json({
  92     message : error.message || error,
  93     error : true,
  94     success : false
  95   })
  96 }
  97 }
  98
 99 export const getProductByCategory = async(request,response)=>{
100   try {
101     const { id } = request.body
102
103     if(!id){
104       return response.status(400).json({
105         message : "provide category id",
106         error : true,
107         success : false
108       })
109     }
  }
```

```
JS product.controller.js × |  
server > controllers > JS product.controller.js > ...  
99  export const getProductByCategory = async(request,response)=>{  
100    const product = await ProductModel.find({  
101      category : { $in : id }  
102    }).limit(15)  
103  
104    return response.json({  
105      message : "category product list",  
106      data : product,  
107      error : false,  
108      success : true  
109    })  
110  } catch (error) {  
111    return response.status(500).json({  
112      message : error.message || error,  
113      error : true,  
114      success : false  
115    })  
116  }  
117  
118  export const getProductByCategoryAndSubCategory = async(request,response)=>{  
119    try {  
120      const { categoryId,subCategoryId,page,limit } = request.body  
121  
122      if(!categoryId || !subCategoryId){  
123        return response.status(400).json({  
124          message : "Provide categoryId and subCategoryId",  
125          error : true,  
126          success : false  
127        })  
128      }  
129    }  
130  }  
131  
132  export const getSubCategory = async(request,response)=>{  
133    const subCategory = await SubCategoryModel.find({  
134      category : { $in : id }  
135    }).limit(15)  
136  
137    return response.json({  
138      message : "SubCategory list",  
139      data : subCategory,  
140      error : false,  
141      success : true  
142    })  
143  }  
144  
145  export const getSubCategoryById = async(request,response)=>{  
146    const subCategory = await SubCategoryModel.findById(id)  
147  
148    return response.json({  
149      message : "SubCategory details",  
150      data : subCategory,  
151      error : false,  
152      success : true  
153    })  
154  }  
155  
156  export const createSubCategory = async(request,response)=>{  
157    const { name,description,category } = request.body  
158  
159    const subCategory = new SubCategoryModel({  
160      name,  
161      description,  
162      category  
163    })  
164  
165    await subCategory.save()  
166  
167    return response.json({  
168      message : "SubCategory created",  
169      data : subCategory,  
170      error : false,  
171      success : true  
172    })  
173  }  
174  
175  export const updateSubCategory = async(request,response)=>{  
176    const { id } = request.params  
177    const { name,description } = request.body  
178  
179    const subCategory = await SubCategoryModel.findById(id)  
180  
181    subCategory.name = name  
182    subCategory.description = description  
183  
184    await subCategory.save()  
185  
186    return response.json({  
187      message : "SubCategory updated",  
188      data : subCategory,  
189      error : false,  
190      success : true  
191    })  
192  }  
193  
194  export const deleteSubCategory = async(request,response)=>{  
195    const { id } = request.params  
196  
197    await SubCategoryModel.findByIdAndDelete(id)  
198  
199    return response.json({  
200      message : "SubCategory deleted",  
201      data : {},  
202      error : false,  
203      success : true  
204    })  
205  }  
206
```

```
js product.controller.js > |  
server > controllers > JS product.controller.js > ...  
130  export const getProductsByCategoryAndSubCategory = async (request, response) => {  
137      |         error: true,  
138      |         success: false  
139      |     })  
140  }  
141  
142  if (!page){  
143  |     page = 1  
144  }  
145  
146  if (!limit){  
147  |     limit = 10  
148  }  
149  
150  const query = {  
151  |     category: { $in: categoryId },  
152  |     subCategory: { $in: subCategoryId }  
153  }  
154  
155  const skip = (page - 1) * limit  
156  
157  const [data, dataCount] = await Promise.all([  
158  |     ProductModel.find(query).sort({ createdAt: -1 }).skip(skip).limit(limit),  
159  |     ProductModel.countDocuments(query)  
160  ])  
161  
162  return response.json({  
163  |     message: "Product list",  
164  |     data: data,
```

```
JS product.controller.js X
server > controllers > JS product.controller.js > ...
181 export const getProductDetails = async(request,response)=>{
182   try {
183     const { _id } = request.query
184     const product = await ProductModel.findById(_id)
185     if(!product){
186       return response.status(404).json({
187         message : "product not found",
188         error : true,
189         success : false
190       })
191     }
192     return response.json({
193       product,
194       error : false,
195       success : true
196     })
197   } catch (error) {
198     return response.status(500).json({
199       message : error.message || error,
200       error : true,
201       success : false
202     })
203   }
204
205   //update product
206   export const updateProductDetails = async(request,response)=>{
207     try {
208       const { _id } = request.body
209
210       if(!_id){
211         return response.status(400).json({
212           message : "provide product _id",
213           error : true,
214           success : false
215         })
216       }
217
218       const updateProduct = await ProductModel.updateOne({ _id : _id },
219       ...request.body
220     )
221     }
222   }
223 }
```

```

JS product.controller.js ×
server > controllers > JS product.controller.js > ...
205 export const updateProductDetails = async(request,response)=>{
219   }
220   return response.json({
221     message : "Updated successfully",
222     data : updateProduct,
223     error : false,
224     success : true
225   })
226 }
227 } catch (error) {
228   return response.status(500).json({
229     message : error.message || error,
230     error : true,
231     success : false
232   })
233 }
234 }
235 }
236 //delete product
237 export const deleteProductDetails = async(request,response)=>{
238   try {
239     const { _id } = request.body
240     if(!_id){
241       return response.status(400).json({
242         message : "provide _id ",
243         error : true,
244         success : false
245       })
246     }
247   }
248   catch (error) {
249     return response.json({
250       message : "provide _id ",
251       error : true,
252       success : false
253     })
254   }
255   const deleteProduct = await ProductModel.deleteOne({_id : _id })
256   return response.json({
257     message : "Deleted successfully",
258     error : false,
259     success : true,
260     data : deleteProduct
261   })
262 } catch (error) {
263   return response.status(500).json({
264     message : error.message || error,
265     error : true,
266     success : false
267   })
268 }
269 //search product
270 export const searchProduct = async(request,response)=>{
271   try {
272     let { search, page , limit } = request.body
273     if(!page){
274       page = 1
275     }
276     if(!limit){
277       limit = 10
278     }
279     const query = search ? {
280       $text : {
281         $search : search
282       }
283     } : {}
284     const skip = ( page - 1 ) * limit
285     const [data,dataCount] = await Promise.all([
286       ProductModel.find(query).sort({ createdAt : -1 }).skip(skip).limit(limit).populate('category subCategory')
287       ProductModel.countDocuments(query)
288     ])
289     return response.json({
290       message : "Product data",
291       error : false,
292       success : true,
293     })
294   }
295 }
```

```

JS product.controller.js ×
server > controllers > JS product.controller.js > ...
267 //search product
268 export const searchProduct = async(request,response)=>{
269   try {
270     let { search, page , limit } = request.body
271     if(!page){
272       page = 1
273     }
274     if(!limit){
275       limit = 10
276     }
277     const query = search ? {
278       $text : {
279         $search : search
280       }
281     } : {}
282     const skip = ( page - 1 ) * limit
283     const [data,dataCount] = await Promise.all([
284       ProductModel.find(query).sort({ createdAt : -1 }).skip(skip).limit(limit).populate('category subCategory')
285       ProductModel.countDocuments(query)
286     ])
287     return response.json({
288       message : "Product data",
289       error : false,
290       success : true,
291     })
292   }
293 }
```

JS product.controller.js X

```
server > controllers > JS product.controller.js > ...
268  export const searchProduct = async(request,response)=
269    |)
291
292    return response.json({
293      message : "Product data",
294      error : false,
295      success : true,
296      data : data,
297      totalCount :dataCount,
298      totalPage : Math.ceil(dataCount/limit),
299      page : page,
300      limit : limit
301    })
302
303
304  } catch (error) {
305    return response.status(500).json({
306      message : error.message || error,
307      error : true,
308      success : false
309    })
310  }
311 }
```

JS subCategory.controller.js X

```
server > controllers > JS subCategory.controller.js > ...
1  import SubCategoryModel from "../models/subCategory.model.js";
2
3  export const AddSubCategoryController = async(request,response)=>{
4    try {
5      const { name, image, category } = request.body
6
7      if(!name && !image && !category[0] ){
8        return response.status(400).json({
9          message : "Provide name, image, category",
10         error : true,
11         success : false
12       })
13
14
15      const payload = {
16        name,
17        image,
18        category
19      }
20
21      const createSubCategory = new SubCategoryModel(payload)
22      const save = await createSubCategory.save()
23
24      return response.json({
25        message : "Sub Category Created",
26        data : save,
27        error : false,
28        success : true
29      })
30
31  }
```

```

JS subCategory.controller.js X
server > controllers > JS subCategory.controller.js > ...
3  export const AddSubCategoryController = async(request,response)=>{
30 |
31      } catch (error) {
32          return response.status(500).json({
33              message : error.message || error,
34              error : true,
35              success : false
36          })
37      }
38  }

40 export const getSubCategoryController = async(request,response)=>{
41     try {
42         const data = await SubCategoryModel.find().sort({createdAt : -1}).populate('category')
43         return response.json({
44             message : "Sub Category data",
45             data : data,
46             error : false,
47             success : true
48         })
49     } catch (error) {
50         return response.status(500).json({
51             message : error.message || error,
52             error : true,
53             success : false
54         })
55     }
56 }
57

```

```

JS subCategory.controller.js X
server > controllers > JS subCategory.controller.js > ...
58  export const updateSubCategoryController = async(request,response)=>{
59      try {
60          const { _id, name, image,category } = request.body
61          const checkSub = await SubCategoryModel.findById(_id)
62
63          if(!checkSub){
64              return response.status(400).json({
65                  message : "Check your _id",
66                  error : true,
67                  success : false
68              })
69          }
70
71          const updateSubCategory = await SubCategoryModel.findByIdAndUpdate(_id,{
72              name,
73              image,
74              category
75          })
76
77          return response.json({
78              message : 'Updated Successfully',
79              data : updateSubCategory,
80              error : false,
81              success : true
82          })
83
84      } catch (error) {
85          return response.status(500).json({
86              message : error.message || error,
87              error : true,
88              success : false
89          })
90      }
91  }
92 }

93

94  export const deleteSubCategoryController = async(request,response)=>{
95      try {
96          const { _id } = request.body
97          console.log("Id",_id)
98          const deleteSub = await SubCategoryModel.findByIdAndDelete(_id)
99
100         return response.json({
101             message : "Deleted successfully",
102             data : deleteSub,
103             error : false,
104             success : true
105         })
106     } catch (error) {
107         return response.status(500).json({
108             message : error.message || error,
109             error : true,
110             success : false
111         })
112     }
113

```

Figure 4.19: Product and Inventory Controllers

4.4.3 Cart & Order Code

- **cart.controller.js** – Add, update, and remove cart items
- **order.controller.js** – Order creation and order history

```
JS cart.controller.js X
server > controllers > JS cart.controller.js > [e] deleteCartItemQtyController
1 import CartProductModel from "../models/cartproduct.model.js";
2 import UserModel from "../models/user.model.js";
3 export const addToCartItemController = async(request,response)=>{
4     try {
5         const userId = request.userId
6         const { productId } = request.body
7         if(!productId){
8             return response.status(402).json({
9                 message : "Provide productId",
10                error : true,
11                success : false
12            })
13        }
14        const checkItemCart = await CartProductModel.findOne({
15            userId : userId,
16            productId : productId
17        })
18        if(checkItemCart){
19            return response.status(400).json({
20                message : "Item is already in the cart"
21            })
22        }
23        const cartItem = new CartProductModel({
24            quantity : 1,
25            userId : userId,
26            productId : productId
27        })
28        const save = await cartItem.save()
29        const updateCartUser = await UserModel.updateOne({ _id : userId },{
30            shopping_cart : [
31                {
32                    productId : productId
33                }
34            ]
35        })
36        return response.json({
37            data : save,
38            message : "Item added successfully",
39            error : false,
40            success : true
41        })
42    } catch (error) {
43        return response.status(500).json({
44            message : error.message || error,
45            error : true,
46            success : false
47        })
48    }
49}
```

```
JS cart.controller.js X
server > controllers > JS cart.controller.js > [e] deleteCartItemQtyController
3 export const addCartItemController = async(request,response)=>{
4     try {
5         const updateCartUser = await UserModel.updateOne({ _id : userId },{
6             shopping_cart : [
7                 {
8                     productId : productId
9                 }
10            ]
11        })
12        return response.json({
13            data : save,
14            message : "Item added successfully",
15            error : false,
16            success : true
17        })
18    } catch (error) {
19        return response.status(500).json({
20            message : error.message || error,
21            error : true,
22            success : false
23        })
24    }
25}
26 export const getCartItemController = async(request,response)=>{
27    try {
28        const userId = request.userId
29        const cartItem = await CartProductModel.find({
30            userId : userId
31        }).populate('productId')
32        return response.json({
33            data : cartItem,
34            message : "Cart items fetched successfully",
35            error : false,
36            success : true
37        })
38    } catch (error) {
39        return response.status(500).json({
40            message : error.message || error,
41            error : true,
42            success : false
43        })
44    }
45}
```

```
JS cart.controller.js X
server > controllers > JS cart.controller.js > [e] deleteCartItemQtyController
48 export const getCartItemController = async(request,response)=>{
49    try {
50        const userId = request.userId
51        const cartItem = await CartProductModel.find({
52            userId : userId
53        }).populate('productId')
54        return response.json({
55            data : cartItem,
56            message : "Cart items fetched successfully",
57            error : false,
58            success : true
59        })
60    } catch (error) {
61        return response.status(500).json({
62            message : error.message || error,
63            error : true,
64            success : false
65        })
66    }
67}
68
69 export const updateCartItemQtyController = async(request,response)=>{
70    try {
71        const userId = request.userId
72        const { _id,qty } = request.body
73
74        if(!_id || !qty){
75            return response.status(400).json({
76                message : "Provide _id, qty"
77            })
78        }
79        const updateCartItem = await CartProductModel.updateOne({
80            _id : _id,
81            userId : userId
82        },{
83            quantity : qty
84        })
85    } catch (error) {
86        return response.status(500).json({
87            message : error.message || error,
88            error : true,
89            success : false
90        })
91    }
92}
```

```
JS cart.controller.js X
server > controllers > JS cart.controller.js > [e] deleteCartItemQtyController
69 export const updateCartItemQtyController = async(request,response)=>{
70    try {
71        const updateCartUser = await UserModel.updateOne({ _id : userId },{
72            shopping_cart : [
73                {
74                    _id : _id,
75                    quantity : qty
76                }
77            ]
78        })
79        return response.json({
80            message : "Update cart",
81            success : true,
82            error : false,
83            data : updateCartItem
84        })
85    } catch (error) {
86        return response.status(500).json({
87            message : error.message || error,
88            error : true,
89            success : false
90        })
91    }
92}
93 export const deleteCartItemQtyController = async(request,response)=>{
94    try {
95        const userId = request.userId // middleware
96        const { _id } = request.body
97        if(!_id){
98            return response.status(400).json({
99                message : "Provide _id",
100               error : true,
101               success : false
102           })
103       }
104       const deleteCartItem = await CartProductModel.deleteOne({ _id : _id, userId : userId })
105       return response.json({
106           message : "Item removed",
107           error : false,
108           success : true
109       })
110   }
```

JS cart.controller.js

```

server > controllers > JS cart.controller.js > [d] deleteCartItemQtyController
  99  export const deleteCartItemQtyController = async(request,response)=>{
100    const deleteCartItem = await CartProductModel.deleteOne({_id : _id, userId : userId })
101    return response.json({
102      message : "Item removed",
103      error : false,
104      success : true,
105      data : deleteCartItem
106    })
107  } catch (error) {
108    return response.status(500).json({
109      message : error.message || error,
110      error : true,
111      success : false
112    })
113  }
114}
115
116
117
118
119
120
121
122
123
124

```

JS order.controller.js

```

server > controllers > JS order.controller.js ...
  1 import Stripe from "../config/stripe.js";
  2 import CartProductModel from "../models/cartproduct.model.js";
  3 import OrderModel from "../models/order.model.js";
  4 import UserModel from "../models/user.model.js";
  5 import mongoose from "mongoose";
  6
  7 // Helper: calculate price after discount
  8 export const priceWithDiscount = (price, discount = 0) => {
  9   const p = Number(price) || 0;
10   const dis = Number(discount) || 0;
11   const discountAmount = Math.ceil((p * dis) / 100);
12   return p - discountAmount;
13 };
14
15 // --- Cash on Delivery Order ---
16 export async function CashOnDeliveryOrderController(req, res) {
17   try {
18     const userId = req.userId;
19     const { list_items, totalAmt, addressId, subTotalAmt } = req.body;
20
21     const payload = list_items.map(el => ({
22       userId,
23       orderId: `ORD-${new mongoose.Types.ObjectId()}`,
24       productId: el.productId._id,
25       product_details: {
26         name: el.productId.name,
27         image: el.productId.image
28       },
29       paymentId: ""
30     })
31   )
32
33   const generatedOrder = await OrderModel.insertMany(payload);
34
35   await CartProductModel.deleteMany({ userId });
36   await UserModel.updateOne({ _id: userId }, { shopping_cart: [] });
37
38   return res.json({
39     message: "Order successfully placed",
40     success: true,
41     error: false,
42     data: generatedOrder
43   });
44 } catch (err) {
45   return res.status(500).json({
46     message: err.message || err,
47     success: false,
48     error: true
49   });
50 }
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84

```

JS order.controller.js

```

server > controllers > JS order.controller.js ...
  16  export async function CashOnDeliveryOrderController(req, res) {
  17    const payload = list_items.map(el => ({
  18      paymentId: "",
  19      payment_status: "CASH ON DELIVERY",
  20      delivery_address: addressId,
  21      subTotalAmt,
  22      totalAmt
  23    }));
  24
  25    const generatedOrder = await OrderModel.insertMany(payload);
  26
  27    await CartProductModel.deleteMany({ userId });
  28    await UserModel.updateOne({ _id: userId }, { shopping_cart: [] });
  29
  30    return res.json({
  31      message: "Order successfully placed",
  32      success: true,
  33      error: false,
  34      data: generatedOrder
  35    });
  36  } catch (err) {
  37    return res.status(500).json({
  38      message: err.message || err,
  39      success: false,
  40      error: true
  41    });
  42  }
  43
  44
  45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1048
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1068
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1088
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1098
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1108
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1118
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1128
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1138
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1146
  1147
  1148
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1158
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1168
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1178
  1179
  1180
  1181
  1182
  1183
  1184
  1185
  1186
  1187
  1188
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1198
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1208
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1217
  1218
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1227
  1228
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1238
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1246
  1247
  1248
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1257
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1267
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1276
  1277
  1278
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1286
  1287
  1288
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1296
  1297
  1298
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1307
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1316
  1317
  1318
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1326
  1327
  1328
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1337
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1345
  1346
  1347
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1356
  1357
  1358
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1366
  1367
  1368
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1376
  1377
  1378
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1386
  1387
  1388
  1389
  1390
  1391
  1392
  1393
  1394
  1395
  1396
  1397
  1398
  1399
  1400
  1401
  1402
  1403
  1404
  1405
  1406
  1407
  1408
  1409
  1410
  1411
  1412
  1413
  1414
  1415
  1416
  1417
  1418
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1426
  1427
  1428
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1436
  1437
  1438
  1439
  1440
  1441
  1442
  1443
  1444
  1445
  1446
  1447
  1448
  1449
  1450
  1451
  1452
  1453
  1454
  1455
  1456
  1457
  1458
  1459
  1460
  1461
  1462
  1463
  1464
  1465
  1466
  1467
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1475
  1476
  1477
  1478
  1479
  1480
  1481
  1482
  1483
  1484
  1485
  1486
  1487
  1488
  1489
  1490
  1491
  1492
  1493
  1494
  1495
  1496
  1497
  1498
  1499
  1500
  1501
  1502
  1503
  1504
  1505
  1506
  1507
  1508
  1509
  1510
  1511
  1512
  1513
  1514
  1515
  1516
  1517
  1518
  1519
  1520
  1521
  1522
  1523
  1524
  1525
  1526
  1527
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1535
  1536
  1537
  1538
  1539
  1540
  1541
  1542
  1543
  1544
  1545
  1546
  1547
  1548
  1549
  1550
  1551
  1552
  1553
  1554
  1555
  1556
  1557
  1558
  1559
  1560
  1561
  1562
  1563
  1564
  1565
  1566
  1567
  1568
  1569
  1570
  1571
  1572
  1573
  1574
  1575
  1576
  1577
  1578
  1579
  1580
  1581
  1582
  1583
  1584
  1585
  1586
  1587
  1588
  1589
  1590
  1591
  1592
  1593
  1594
  1595
  1596
  1597
  1598
  1599
  1600
  1601
  1602
  1603
  1604
  1605
  1606
  1607
  1608
  1609
  1610
  1611
  1612
  1613
  1614
  1615
  1616
  1617
  1618
  1619
  1620
  1621
  1622
  1623
  1624
  1625
  1626
  1627
  1628
  1629
  1630
  1631
  1632
  1633
  1634
  1635
  1636
  1637
  1638
  1639

```

JS order.controller.js

```

server > controllers > JS order.controller.js > ...
57  export async function paymentController(req, res) {
114  |   return res.status(200).json({ success: true, url: session.url });
115  | }
116  | } catch (err) {
117  |   console.error("X FATAL STRIPE ERROR:", err.message);
118  |   console.error("Full Error Object:", JSON.stringify(err, null, 2));
119  |
120  |   return res.status(500).json({ message: err.message || "Payment processing failed.", success: false, error: true })
121  |
122  |
123  |
124  / Helper: get Stripe line items for order creation
125  const getOrderProductItems = async ({ lineItems, userId, addressId, paymentId, payment_status }) => {
126  |   const productPromises = lineItems.data.map(async item => {
127  |     const stripeProduct = item.price.product;
128  |     const imageUrl = (stripeProduct.images && stripeProduct.images.length > 0)
129  |     || || || || ? stripeProduct.images[0]
130  |     || || || || : '';
131  |
132  |     return {
133  |       userId,
134  |       orderId: `ORD-${new mongoose.Types.ObjectId()}`,
135  |       productId: stripeProduct.metadata.productId,
136  |       product_details: { name: stripeProduct.name, image: imageUrl },
137  |       paymentId,
138  |       payment_status,
139  |       subtotalAmt: (item.amount_subtotal / 100),
140  |       totalAmt: (item.amount_total / 100),
141  |       delivery_address: addressId,
142  |     };
143  |   });
144  |
145  |
146  / --- Stripe Webhook ---
147  export async function webhookStripe(req, res) {
148  |   const sig = req.headers['stripe-signature'];
149  |   const endpointSecret = process.env.STRIPE_WEBHOOK_SECRET_KEY;
150  |
151  |   if (!endpointSecret || typeof endpointSecret !== 'string') {
152  |     console.error("X STRIPE_WEBHOOK_SECRET_KEY not configured.");
153  |     return res.status(400).send('Webhook secret not configured.');
154  |   }
155  |
156  |   let event;
157  |   try {
158  |     event = Stripe.webhooks.constructEvent(req.body, sig, endpointSecret);
159  |   } catch (err) {
160  |     console.log(`X Webhook Signature Error: ${err.message}`);
161  |     return res.status(400).send(`Webhook Error: ${err.message}`);
162  |   }
163  |
164  |   switch (event.type) {
165  |     case "checkout.session.completed":
166  |       const session = event.data.object;
167  |       const lineItems = await Stripe.checkout.sessions.listLineItems(session.id, { expand: ['data.price.product'] });
168  |       const userId = session.metadata.userId;
169  |   }

```

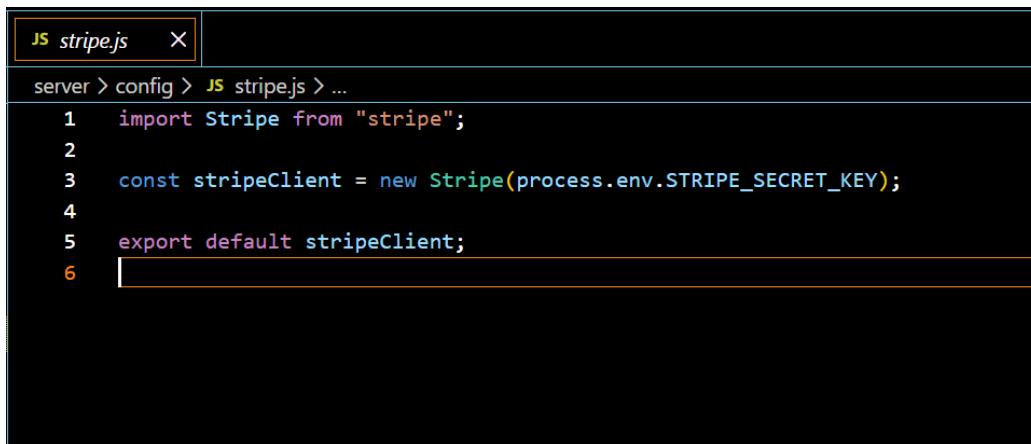
JS order.controller.js

```

server > controllers > JS order.controller.js > ...
125  const getOrderProductItems = async ({ lineItems, userId, addressId, paymentId, payment_status }) => {
126  |   const productPromises = lineItems.data.map(async item => {
127  |     });
128  |   return Promise.all(productPromises);
129  |
130  |
131  |
132  |
133  |
134  |
135  |
136  |
137  |
138  |
139  |
140  |
141  |
142  |
143  |
144  |
145  |
146  |
147  |
148  |
149  |
150  |
151  |
152  |
153  |
154  |
155  |
156  |
157  |
158  |
159  |
160  |
161  |
162  |
163  |
164  |
165  |
166  |
167  |
168  |
169  |
170  |
171  |
172  |
173  |
174  |
175  |
176  |
177  |
178  |
179  |
180  |
181  |
182  |
183  |
184  |
185  |
186  |
187  |
188  |
189  |
190  |
191  |
192  |
193  |
194  |
195  |
196  |
197  |
198  |
199  |
200  |
201  |
202  |
203  |
204  |
205  |
206  |
207  |
208  |
209  |
210  |
211  |
212  |
213  |
214  |
215  |
216  |
217  |
218  |
219  |
220  |
221  |
222  |
223  |
224  |
225  |
226  |
227  |
228  |
229  |
230  |
231  |
232  |
233  |
234  |
235  |
236  |
237  |
238  |
239  |
240  |
241  |
242  |
243  |
244  |
245  |
246  |
247  |
248  |
249  |
250  |
251  |
252  |
253  |
254  |
255  |
256  |
257  |
258  |
259  |
260  |
261  |
262  |
263  |
264  |
265  |
266  |
267  |
268  |
269  |
270  |
271  |
272  |
273  |
274  |
275  |
276  |
277  |
278  |
279  |
280  |
281  |
282  |
283  |
284  |
285  |
286  |
287  |
288  |
289  |
290  |
291  |
292  |
293  |
294  |
295  |
296  |
297  |
298  |
299  |
300  |
301  |
302  |
303  |
304  |
305  |
306  |
307  |
308  |
309  |
310  |
311  |
312  |
313  |
314  |
315  |
316  |
317  |
318  |
319  |
320  |
321  |
322  |
323  |
324  |
325  |
326  |
327  |
328  |
329  |
330  |
331  |
332  |
333  |
334  |
335  |
336  |
337  |
338  |
339  |
340  |
341  |
342  |
343  |
344  |
345  |
346  |
347  |
348  |
349  |
350  |
351  |
352  |
353  |
354  |
355  |
356  |
357  |
358  |
359  |
360  |
361  |
362  |
363  |
364  |
365  |
366  |
367  |
368  |
369  |
370  |
371  |
372  |
373  |
374  |
375  |
376  |
377  |
378  |
379  |
380  |
381  |
382  |
383  |
384  |
385  |
386  |
387  |
388  |
389  |
390  |
391  |
392  |
393  |
394  |
395  |
396  |
397  |
398  |
399  |
400  |
401  |
402  |
403  |
404  |
405  |
406  |
407  |
408  |
409  |
410  |
411  |
412  |
413  |
414  |
415  |
416  |
417  |
418  |
419  |
420  |
421  |
422  |
423  |
424  |
425  |
426  |
427  |
428  |
429  |
430  |
431  |
432  |
433  |
434  |
435  |
436  |
437  |
438  |
439  |
440  |
441  |
442  |
443  |
444  |
445  |
446  |
447  |
448  |
449  |
450  |
451  |
452  |
453  |
454  |
455  |
456  |
457  |
458  |
459  |
460  |
461  |
462  |
463  |
464  |
465  |
466  |
467  |
468  |
469  |
470  |
471  |
472  |
473  |
474  |
475  |
476  |
477  |
478  |
479  |
480  |
481  |
482  |
483  |
484  |
485  |
486  |
487  |
488  |
489  |
490  |
491  |
492  |
493  |
494  |
495  |
496  |
497  |
498  |
499  |
500  |
501  |
502  |
503  |
504  |
505  |
506  |
507  |
508  |
509  |
510  |
511  |
512  |
513  |
514  |
515  |
516  |
517  |
518  |
519  |
520  |
521  |
522  |
523  |
524  |
525  |
526  |
527  |
528  |
529  |
530  |
531  |
532  |
533  |
534  |
535  |
536  |
537  |
538  |
539  |
540  |
541  |
542  |
543  |
544  |
545  |
546  |
547  |
548  |
549  |
550  |
551  |
552  |
553  |
554  |
555  |
556  |
557  |
558  |
559  |
560  |
561  |
562  |
563  |
564  |
565  |
566  |
567  |
568  |
569  |
570  |
571  |
572  |
573  |
574  |
575  |
576  |
577  |
578  |
579  |
580  |
581  |
582  |
583  |
584  |
585  |
586  |
587  |
588  |
589  |
590  |
591  |
592  |
593  |
594  |
595  |
596  |
597  |
598  |
599  |
600  |
601  |
602  |
603  |
604  |
605  |
606  |
607  |
608  |
609  |
610  |
611  |
612  |
613  |
614  |
615  |
616  |
617  |
618  |
619  |
620  |
621  |
622  |
623  |
624  |
625  |
626  |
627  |
628  |
629  |
630  |
631  |
632  |
633  |
634  |
635  |
636  |
637  |
638  |
639  |
640  |
641  |
642  |
643  |
644  |
645  |
646  |
647  |
648  |
649  |
650  |
651  |
652  |
653  |
654  |
655  |
656  |
657  |
658  |
659  |
660  |
661  |
662  |
663  |
664  |
665  |
666  |
667  |
668  |
669  |
670  |
671  |
672  |
673  |
674  |
675  |
676  |
677  |
678  |
679  |
680  |
681  |
682  |
683  |
684  |
685  |
686  |
687  |
688  |
689  |
690  |
691  |
692  |
693  |
694  |
695  |
696  |
697  |
698  |
699  |
700  |
701  |
702  |
703  |
704  |
705  |
706  |
707  |
708  |
709  |
710  |
711  |
712  |
713  |
714  |
715  |
716  |
717  |
718  |
719  |
720  |
721  |
722  |
723  |
724  |
725  |
726  |
727  |
728  |
729  |
730  |
731  |
732  |
733  |
734  |
735  |
736  |
737  |
738  |
739  |
740  |
741  |
742  |
743  |
744  |
745  |
746  |
747  |
748  |
749  |
750  |
751  |
752  |
753  |
754  |
755  |
756  |
757  |
758  |
759  |
760  |
761  |
762  |
763  |
764  |
765  |
766  |
767  |
768  |
769  |
770  |
771  |
772  |
773  |
774  |
775  |
776  |
777  |
778  |
779  |
779  |
780  |
781  |
782  |
783  |
784  |
785  |
786  |
787  |
788  |
789  |
789  |
790  |
791  |
792  |
793  |
794  |
795  |
796  |
797  |
798  |
799  |
799  |
800  |
801  |
802  |
803  |
804  |
805  |
806  |
807  |
808  |
809  |
809  |
810  |
811  |
812  |
813  |
814  |
815  |
816  |
817  |
818  |
819  |
819  |
820  |
821  |
822  |
823  |
824  |
825  |
826  |
827  |
828  |
829  |
829  |
830  |
831  |
832  |
833  |
834  |
835  |
836  |
837  |
838  |
839  |
839  |
840  |
841  |
842  |
843  |
844  |
845  |
846  |
847  |
848  |
849  |
849  |
850  |
851  |
852  |
853  |
854  |
855  |
856  |
857  |
858  |
859  |
859  |
860  |
861  |
862  |
863  |
864  |
865  |
866  |
867  |
868  |
869  |
869  |
870  |
871  |
872  |
873  |
874  |
875  |
876  |
877  |
878  |
879  |
879  |
880  |
881  |
882  |
883  |
884  |
885  |
886  |
887  |
888  |
889  |
889  |
890  |
891  |
892  |
893  |
894  |
895  |
896  |
897  |
898  |
899  |
899  |
900  |
901  |
902  |
903  |
904  |
905  |
906  |
907  |
908  |
909  |
909  |
910  |
911  |
912  |
913  |
914  |
915  |
916  |
917  |
918  |
919  |
919  |
920  |
921  |
922  |
923  |
924  |
925  |
926  |
927  |
928  |
929  |
929  |
930  |
931  |
932  |
933  |
934  |
935  |
936  |
937  |
938  |
939  |
939  |
940  |
941  |
942  |
943  |
944  |
945  |
946  |
947  |
948  |
949  |
949  |
950  |
951  |
952  |
953  |
954  |
955  |
956  |
957  |
958  |
958  |
959  |
960  |
961  |
962  |
963  |
964  |
965  |
966  |
967  |
968  |
969  |
969  |
970  |
971  |
972  |
973  |
974  |
975  |
976  |
977  |
978  |
979  |
979  |
980  |
981  |
982  |
983  |
984  |
985  |
986  |
987  |
988  |
989  |
989  |
990  |
991  |
992  |
993  |
994  |
995  |
996  |
997  |
998  |
999  |
999  |
1000  |
1001  |
1002  |
1003  |
1004  |
1005  |
1006  |
1007  |
1008  |
1009  |
1009  |
1010  |
1011  |
1012  |
1013  |
1014  |
1015  |
1016  |
1017  |
1018  |
1019  |
1019  |
1020  |
1021  |
1022  |
1023  |
1024  |
1025  |
1026  |
1027  |
1028  |
1029  |
1029  |
1030  |
1031  |
1032  |
1033  |
1034  |
1035  |
1036  |
1037  |
1038  |
1039  |
1039  |
1040  |
1041  |
1042  |
1043  |
1044  |
1045  |
1046  |
1047  |
1048  |
1049  |
1049  |
1050  |
1051  |
1052  |
1053  |
1054  |
1055  |
1056  |
1057  |
1058  |
1059  |
1059  |
1060  |
1061  |
1062  |
1063  |
1064  |
1065  |
1066  |
1067  |
1068  |
1069  |
1069  |
1070  |
1071  |
1072  |
1073  |
1074  |
1075  |
1076  |
1077  |
1078  |
1079  |
1079  |
1080  |
1081  |
1082  |
1083  |
1084  |
1085  |
1086  |
1087  |
1088  |
1089  |
1089  |
1090  |
1091  |
1092  |
1093  |
1094  |
1095  |
1096  |
1097  |
1098  |
1099  |
1099  |
1100  |
1101  |
1102  |
1103  |
1104  |
1105  |
1106  |
1107  |
1108  |
1109  |
1109  |
1110  |
1111  |
1112  |
1113  |
1114  |
1115  |
1116  |
1117  |
1118  |
1119  |
1119  |
1120  |
1121  |
1122  |
1123  |
1124  |
1125  |
1126  |
1127  |
1128  |
1129  |
1129  |
1130  |
1131  |
1132  |
1133  |
1134  |
1135  |
1136  |
1137  |
1138  |
1139  |
1139  |
1140  |
1141  |
1142  |
1143  |
1144  |
1145  |
1146  |
1147  |
1148  |
1149  |
1149  |
1150  |
1151  |
1152  |
1153  |
1154  |
1155  |
1156  |
1157  |
1158  |
1159  |
1159  |
1160  |
1161  |
1162  |
1163  |
1164  |
1165  |
1166  |
1167  |
1167  |
1168  |
1169  |
1170  |
1171  |
1172  |
1173  |
1174  |
1175  |
1176  |
1177  |
1178  |
1179  |
1180  |
1181  |
1182  |
1183  |
1184  |
1185  |
1186  |
1187  |
1188  |
1189  |
1189  |
1190  |
1191  |
1192  |
1193  |
1194  |
1195  |
1196  |
1197  |
1198  |
1199  |
1199  |
1200  |
1201  |
1202  |
1203  |
1204  |
1205  |
1206  |
1207  |
1208  |
1209  |
1209  |
1210  |
1211  |
1212  |
1213  |
1214  |
1215  |
1216  |
1217  |
1218  |
1219  |
1219  |
1220  |
1221  |
1222  |
1223  |
1224  |
1225  |
1226  |
1227  |
1228  |
1229  |
1229  |
1230  |
1231  |
1232  |
1233  |
1234  |
1235  |
1236  |
1237  |
1238  |
1239  |
1239  |
1240  |
1241  |
1242  |
1243  |
1244  |
1245  |
1246  |
1247  |
1248  |
1249  |
1249  |
1250  |
1251  |
1252  |
1253  |
1254  |
1255  |
1256  |
1257  |
1258  |
1259  |
1259  |
1260  |
1261  |
1262  |
1263  |
1264  |
1265  |
1266  |
1267  |
1268  |
1269  |
1269  |
1270  |
1271  |
1272  |
1273  |
1274  |
1275  |
1276  |
1277  |
1278  |
1279  |
1279  |
1280  |
1281  |
1282  |
1283  |
1284  |
1285  |
1286  |
1287  |
1288  |
1289  |
1289  |
1290  |
1291  |
1292  |
1293  |
1294  |
1295  |
1296  |
1297  |
1298  |
1299  |
1299  |
1300  |
1301  |
1302  |
1303  |
1304  |
1305  |
1306  |
1307  |
1308  |
1309  |
1309  |
1310  |
1311  |
1312  |
1313  |
1314  |
1315  |
1316  |
1317  |
1318  |
1319  |
1319  |
1320  |
1321  |
1322  |
1323  |
1324  |
1325  |
1326  |
1327  |
1328  |
1329  |
1329  |
1330  |
1331  |
1332  |
1333  |
1334  |
1335  |
1336  |
1337  |
1338  |
1339  |
1339  |
1340  |
1341  |
1342  |
1343  |
1344  |
1345  |
1346  |
1347  |
1348  |
1349  |
1349  |
1350  |
1351  |
1352  |
1353  |
1354  |
1355  |
1356  |
1357  |
1358  |
1359  |
1359  |
1360  |
1361  |
1362  |
1363  |
1364  |
1365  |
1366  |
1367  |
1368  |
1369  |
1369  |
1370  |
1371  |
1372  |
1373  |
1374  |
1375  |
1376  |
1377  |
1378  |
1379  |
1379  |
1380  |
1381  |
1382  |
1383  |
1384  |
1385  |
1386  |
1387  |
1388  |
1389  |
1389  |
1390  |
1391  |
1392  |
1393  |
1394  |
1395  |
1396  |
1397  |
1398  |
1399  |
1399  |
1400  |
1401  |
1402  |
1403  |
1404  |
1405  |
1406  |
1407  |
1408  |
1409  |
1409  |
1410  |
1411  |
1412  |
1413  |
1414  |
1415  |
1416  |
1417  |
1418  |
1419  |
1419  |
1420  |
1421  |
1422  |
1423  |
1424  |
1425  |
1426  |
1427  |
1428  |
1429  |
1429  |
1430  |
1431  |
1432  |
1433  |
1434  |
1435  |
1436  |
1437  |
1438  |
1438  |
1439  |
1440  |
1441  |
1442  |
1443  |
1444  |
1445  |
1446  |
1447  |
1448  |
1449  |
1449  |
1450  |
1451  |
1452  |
1453  |
1454  |
1455  |
1456  |
1457  |
1458  |
1458  |
1459  |
1460  |
1461  |
1462  |
1463  |
1464  |
1465  |
1466  |
1467  |
1468  |
1469  |
1469  |
1470  |
1471  |
1472  |
1473  |
1474  |
1475  |
1476  |
1477  |
1478  |
1479  |
1479  |
1480  |
1481  |
1482  |
1483  |
1484  |
1485  |
1486  |
1487  |
1488  |
1489  |
1489  |
1490  |
1491  |
1492  |
1493  |
1494  |
1495  |
1496  |
1497  |
1498  |
1499  |
1499  |
1500  |
1501  |
1502  |
1503  |
1504  |
1505  |
1506  |
1507  |
1508  |
1509  |
1509  |
1510  |
1511  |
1512  |
1513  |
1514  |
1515  |
1516  |
1517  |
1518  |
1519  |
1519  |
1520  |
1521  |
1522  |
1523  |
1524  |
1525  |
1526  |
1527  |
1528  |
1529  |
1529  |
1530  |
1531  |
1532  |
1533  |
1534  |
1535  |
1536  |
1537  |
1538  |
1538  |
1539  |
1540  |
1541  |
1542  |
1543  |
1544  |
1545  |
1546  |
1547  |
1548  |
1549  |
1549  |
1550  |
1551  |
1552  |
1553  |
1554  |
1555  |
1556  |
1557  |
1558  |
1559  |
1559  |
1560  |
1561  |
1562  |
1563  |
1564  |
1565  |
1566  |
1567  |
1568  |
1569  |
1569  |
1570  |
1571  |
1572  |
1573  |
1574  |
1575  |
1576  |
1577  |
1578  |
1579  |
1579  |
1580  |
1581  |
1582  |
1583  |
1584  |
1585  |
1586  |
1587  |
1588  |
1589  |
1589  |
1590  |
1591  |
1592  |
1593  |
1594  |
1595  |
1596  |
1597  |
1598  |
1599  |
1599  |
1600  |
1601  |
1602  |
1603  |
1604  |
1605  |
1606  |
1607  |
1608  |
1609  |
1609  |
1610  |
1611  |
1612  |
1613  |
1614  |
1615  |
1616  |
1617  |
1618  |
1619  |
1619  |
1620  |
1621  |
1622  |
1623  |
1624  |
1625  |
1626  |
1627  |
1628  |
1629  |
1629  |
1630  |
1631  |
1632  |
1633  |
1634  |
1635  |
1636  |
1637  |
1638  |
1638  |
1639  |
1640  |
1641  |
1642  |
1643  |
1644  |
1645  |
1646  |
1647  |
1648  |
1649  |
1649  |
1650  |
1651  |
1652  |
1653  |
1654  |
1655  |
1656  |
1657  |
1658  |
1659  |
1659  |
1660  |
1661  |
1662  |
1663  |
1664  |
1665  |
1666  |
1667  |
1668  |
1669  |
1669  |
1670  |
1671  |
1672  |
1673  |
1674  |
1675  |
1676  |
1677  |
1678  |
1679  |
1679  |
1680  |
1681  |
1682  |
1683  |
1684  |
1685  |
1686  |
1687  |
1688  |
1689  |
1689  |
1690  |
1691  |
1692  |
1693  |
1694  |
1695  |
1696  |
1697  |
1698  |
1699  |
1699  |
1700  |
1701  |
1702  |
1703  |
1704  |
1705  |
1706  |
1707  |
1708  |
1709  |
1709  |
1710  |
1711  |
1712  |
1713  |
1714  |
1715  |
1716  |
1717  |
1718  |
1719  |
1719  |
1720  |
1721  |
1722  |
1723  |
1724  |
1725  |
1726  |
1727  |
1728  |
1729  |
1729  |
1730  |
1731  |
1732  |
1733  |
1734  |
1735  |
1736  |
1737  |
1738  |
1738  |
1739  |
1740  |
1741  |
1742  |
1743  |
1744  |
1745  |
1746  |
1747  |
1748  |
1749  |
1749  |
1750  |
1751  |
1752  |
1753  |
1754  |
1755  |
1756  |
1757  |
1758  |
1759  |
1759  |
1760  |
1761  |
1762  |
1763  |
1764  |
1765  |
1766  |
1767  |
1768  |
1769  |
1769  |
1770  |
1771  |
1772  |
1773  |
1774  |
1775  |
1776  |
1777  |
1778  |
1779  |
1779  |
1780  |
1781  |
1782  |
1783  |
1784  |
1785  |
1786  |
1787  |
1788  |
1789  |
1789  |
1790  |
1791  |
1792  |
1793  |
1794  |
1795  |
1796  |
1797  |
1798  |
1799  |
1799  |
1800  |
1801  |
1802  |
1803  |
1804  |
1805  |
1806  |
1807  |
1808  |
1809  |
1809  |
1810  |
1811  |
1812  |
1813  |
1814  |
1815  |
1816  |
1817  |
1818  |
1819  |
1819  |
1820  |
1821  |
1822  |
1823  |
1824  |
1825  |
1826  |
1827  |
1828  |
1829  |
1829  |
1830  |
1831  |
1832  |
1833  |
1834  |
1835  |
1836  |
1837  |
1838  |
1839  |
1839  |
1840  |
1841  |
1842  |
1843  |
1844  |
1845  |
1846  |
1847  |
1848  |
1849  |
1849  |
1850  |
1851  |
1852  |
1853  |
1854  |
1855  |
1856  |
1857  |
1858  |
1859  |
1859  |
1860  |
1861  |
1862  |
1863  |
1864  |
1865  |
1866  |
1867  |
1868  |
1869  |
1869  |
1870  |
1871  |
1872  |
1873  |
1874  |
1875  |
1876  |
1877  |
1878  |
1879  |
1879  |
1880  |
1881  |
1882  |
1883  |
1884  |
1885  |
1886  |
1887  |
1888  |
1889  |
1889  |
1890  |
1891  |
1892  |
1893  |
1894  |
1895  |
1896  |
1897  |
1898  |
1899  |
1899  |
1900  |
1901  |
1902  |
1903  |
1904  |
1905  |
1906  |
1907
```

4.4.4 Payment Gateway Code

- **stripe.js** – Stripe configuration and checkout session creation
- **Webhook Logic (order.controller.js)** – Payment verification and order confirmation



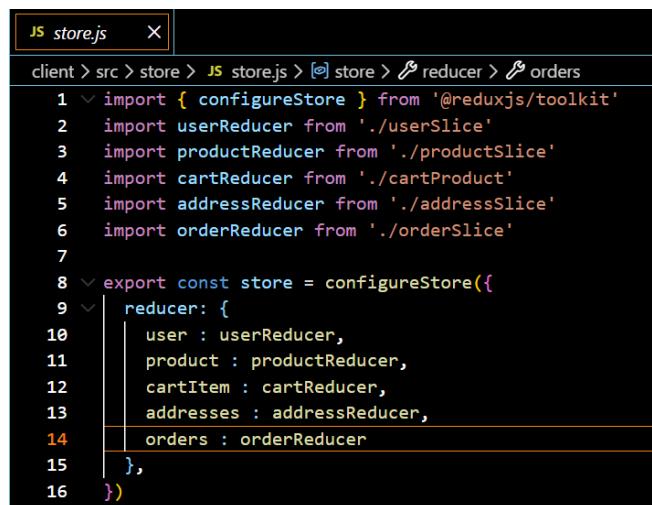
The screenshot shows a code editor window with a dark theme. The title bar says "JS stripe.js X". Below it, the file content is displayed:

```
server > config > JS stripe.js > ...
1 import Stripe from "stripe";
2
3 const stripeClient = new Stripe(process.env.STRIPE_SECRET_KEY);
4
5 export default stripeClient;
6
```

Figure 4.21: Stripe Checkout

4.4.5 Frontend State Management Code

- **store.js** – Redux store configuration
- **userSlice.js** – User state management
- **cartProduct.js** – Cart state management



The screenshot shows a code editor window with a dark theme. The title bar says "JS store.js X". Below it, the file content is displayed:

```
client > src > store > JS store.js > [o] store > ↴ reducer > ↴ orders
1 import { configureStore } from '@reduxjs/toolkit'
2 import userReducer from './userSlice'
3 import productReducer from './productSlice'
4 import cartReducer from './cartProduct'
5 import addressReducer from './addressSlice'
6 import orderReducer from './orderSlice'
7
8 export const store = configureStore({
9   reducer: {
10     user : userReducer,
11     product : productReducer,
12     cartItem : cartReducer,
13     addresses : addressReducer,
14     orders : orderReducer
15   },
16 })
```

```

JS userSlice.js X
client > src > store > JS userSlice.js > [o] default
1 import { createSlice } from "@reduxjs/toolkit";
2 const initialValue = {
3   _id : "",
4   name : "",
5   email : "",
6   avatar : "",
7   mobile : "",
8   verify_email : "",
9   last_login_date : "",
10  status : "",
11  address_details : [],
12  shopping_cart : [],
13  orderHistory : [],
14  role : "",
15 }
16 const userSlice = createSlice({
17   name : 'user',
18   initialState : initialValue,
19   reducers : {
20     setUserDetails : (state,action) =>{
21       state._id = action.payload?._id
22       state.name = action.payload?.name
23       state.email = action.payload?.email
24       state.avatar = action.payload?.avatar
25       state.mobile = action.payload?.mobile
26       state.verify_email = action.payload?.verify_email
27       state.last_login_date = action.payload?.last_login_date
28       state.status = action.payload?.status
29       state.address_details = action.payload?.address_details
30   },
31   reducers : {
32     setUserDetails : (state,action) =>{
33       state.shopping_cart = action.payload?.shopping_cart
34       state.orderHistory = action.payload?.orderHistory
35       state.role = action.payload?.role
36     },
37     updatedAvatar : (state,action)=>{
38       state.avatar = action.payload
39     },
40     logout : (state,action)=>{
41       state._id = ""
42       state.name = ""
43       state.email = ""
44       state.avatar = ""
45       state.mobile = ""
46       state.verify_email = ""
47       state.last_login_date = ""
48       state.status = ""
49       state.address_details = []
50     },
51   },
52 })
53 export const { setUserDetails, logout ,updatedAvatar} = userSlice.actions
54 export default userSlice.reducer

```

```

JS cartProduct.js X
client > src > store > JS cartProduct.js > [o] default
1 import { createSlice } from "@reduxjs/toolkit";
2
3 const initialState = {
4   cart : []
5 }
6
7 const cartSlice = createSlice({
8   name : "cartItem",
9   initialState : initialState,
10  reducers : {
11    handleAddItemCart : (state,action)=>{
12      state.cart = [...action.payload]
13    },
14  },
15 })
16
17 export const { handleAddItemCart } = cartSlice.actions
18
19 export default cartSlice.reducer

```

Figure 4.22: Redux Store and Slice Implementation

4.4.6 Utility Code

- **axios.js** – API request configuration
- **addToCartProduct.js** – Client-side cart handling logic

```
JS Axios.js X
client > src > utils > JS Axios.js > ...
1 import axios from "axios";
2 import SummaryApi , { baseURL } from "../common/SummaryApi";
3
4 const Axios = axios.create({
5   baseURL : baseURL,
6   withCredentials : true
7 })
8
9 //sending access token in the header
10 Axios.interceptors.request.use(
11   async(config)=>{
12     const accessToken = localStorage.getItem('accesstoken')
13
14     if(accessToken){
15       config.headers.Authorization = `Bearer ${accessToken}`
16     }
17
18     return config
19   },
20   (error)=>{
21     return Promise.reject(error)
22   }
23 )
24
25 //extend the life span of access token with
26 // the help refresh
27 Axios.interceptors.request.use(
28   (response)=>{
29     return response
30   }
31 )
```

```
JS Axios.js X
client > src > utils > JS Axios.js > ...
30   },
31   async(error)=>{
32     let originRequest = error.config
33
34     if(error.response.status === 401 && !originRequest.retry){
35       originRequest.retry = true
36
37       const refreshToken = localStorage.getItem("refreshToken")
38
39       if(refreshToken){
40         const newAccessToken = await refreshAccessToken(refreshToken)
41
42         if(newAccessToken){
43           originRequest.headers.Authorization = `Bearer ${newAccessToken}`
44         }
45       }
46     }
47
48     return Promise.reject(error)
49   }
50 )
51
52
53
54 const refreshAccessToken = async(refreshToken)=>{
55   try {
56     const response = await Axios({
57       ...SummaryApi.refreshToken,
58       headers : {
```

```
JS Axios.js X
client > src > utils > JS Axios.js > ...
54   const refreshAccessToken = async(refreshToken)=>{
55     const response = await Axios({
56       headers : {
57         Authorization : `Bearer ${refreshToken}`
58       }
59     )
60
61     const accessToken = response.data.data.accessToken
62     localStorage.setItem('accesstoken',accessToken)
63     return accessToken
64   } catch (error) {
65     console.log(error)
66   }
67 }
68
69 }
70
71 export default Axios
```

```
JS addToCartProduct.js X
client > src > utils > JS addToCartProduct.js > [?] addToCartProduct > [?] response
1  import toast from "react-hot-toast"
2  import SummaryApi from "../common/SummaryApi"
3  import Axios from "./Axios"
4  import AxiosToastError from "./AxiosToastError"
5
6  export const addToCartProduct = async(productId,qty)=>{
7    try {
8      const response = await Axios({
9        ...SummaryApi.addToCart,
10       data : {
11         quantity : qty,
12         productId : productId
13       }
14     })
15
16     const { data : responseData} = response
17
18     console.log(responseData)
19     if(responseData.success){
20       toast.success(responseData.message)
21     }
22     return responseData
23
24   } catch (error) {
25     AxiosToastError(error)
26
27     return {}
28   }
29 }
```

```
JS addToCartProduct.js X
client > src > utils > JS addToCartProduct.js > [?] addToCartProduct > [?] response
30
31  export const getCartItems = async()=>{
32    try {
33      const response = await Axios({
34        ...SummaryApi.getCartItems
35      })
36
37      const { data : responseData } = response
38
39      if(responseData.success){
40        return responseData
41      }
42    } catch (error) {
43      AxiosToastError(error)
44      return error
45    }
46 }
```

Figure 4.23: Frontend Utility Functions

CHAPTER – 5

TESTING AND MAINTENANCE

This chapter describes the testing strategies adopted for the “**Stock & Shop**” E-Commerce System, the test cases executed to verify system functionality, system evaluation, future enhancements, cost–benefit analysis, and a brief user/operational manual.

Testing Methodology Adopted

The **Black Box Testing** methodology was primarily used to test the system. This approach focuses on validating the functionality of the application without considering the internal code structure. Each module was tested based on expected inputs and outputs to ensure correctness, reliability, and usability. In addition, integration testing was carried out to ensure seamless communication between frontend, backend, database, and third-party services such as Stripe.

Test Cases and Expected Results

Test Case ID	Module	Input	Expected Result	Status
TC-01	User Registration	Valid user details	User successfully registered	Pass
TC-02	User Login	Valid credentials	Login successful with JWT token	Pass
TC-03	Forgot Password	Registered email	OTP sent to email	Pass
TC-04	Category Creation	Valid category details	Category added successfully	Pass
TC-05	Product Upload	Valid product data	Product stored in database	Pass
TC-06	Add to Cart	Selected product	Product added to cart	Pass
TC-07	Checkout	Valid cart items	Redirected to Stripe Checkout	Pass
TC-08	Payment Success	Valid card details	Order created and cart cleared	Pass

System Evaluation

The “Stock & Shop” system was evaluated based on performance, security, scalability, and usability.

- **Performance:** The MERN stack ensures fast API responses and dynamic UI rendering.
- **Security:** JWT authentication, bcrypt password hashing, and Stripe’s PCI-compliant payment processing provide strong security.
- **Scalability:** MongoDB and modular backend architecture allow the system to scale with increased users and products.
- **Usability:** The user-friendly interface makes navigation simple for both Admin and Customers.

Overall, the system meets all the defined objectives efficiently.

Future Scope

The system can be further enhanced with the following features:

- Deployment on cloud platforms (AWS / Vercel)
- Mobile application integration
- Advanced analytics for admin dashboard
- Product recommendations using AI
- Multiple payment gateway support
- Real-time order tracking

Cost and Benefit Analysis

Cost Factors

- Development time and effort
- Internet and hardware resources
- Stripe transaction fees (only on successful payments)

Benefits

- 24/7 online availability
- Reduced manual inventory errors
- Secure digital payments
- Improved customer experience
- Efficient order management

The benefits significantly outweigh the costs, making the system economically viable.

User / Operational Manual

For Admin

1. Login using admin credentials.
2. Add categories and sub-categories.
3. Upload and manage products.
4. Monitor orders and inventory levels.
5. View saved customer addresses.

For Customer

1. Register and login.
2. Browse products by category.
3. Add products to cart.
4. Complete checkout using Stripe.
5. View order history and saved addresses.

CONCLUSION

The “**Stock & Shop**” E-Commerce System successfully demonstrates the design and implementation of a full-stack web application using the MERN stack. The project fulfills its primary objective of providing a secure, scalable, and user-friendly platform for online grocery shopping while supporting efficient inventory and order management for administrators.

The system integrates essential features such as role-based authentication, real-time inventory control, cart and order management, and secure online payments using the Stripe payment gateway. The implementation of a secure webhook mechanism ensures reliable order creation and cart clearance only after verified payment confirmation, reflecting real-world e-commerce practices.

Through this project, practical experience was gained in full-stack development, RESTful API design, database modeling, frontend state management using Redux, and third-party service integration. Overall, the project meets all defined requirements and serves as a strong foundation for future enhancements such as cloud deployment, mobile application support, and advanced analytics.

REFERENCES

The development of the “Stock & Shop” E-Commerce System leveraged official documentation and online resources related to its core full-stack technologies, frontend framework, backend runtime environment, database management system, and payment gateway integration. The primary references consulted include:

1. MongoDB Documentation. (n.d.). MongoDB Manual.
Retrieved from <https://www.mongodb.com/docs/>
(Accessed: December 16, 2025).
2. Express.js Documentation. (n.d.). Express.js Official Documentation.
Retrieved from <https://expressjs.com/>
(Accessed: December 16, 2025).
3. React.js Documentation. (n.d.). React – A JavaScript Library for Building User Interfaces.
Retrieved from <https://react.dev/>
(Accessed: December 16, 2025).
4. Node.js Documentation. (n.d.). Node.js Official Documentation.
Retrieved from <https://nodejs.org/en/docs/>
(Accessed: December 16, 2025).
5. Stripe Documentation. (n.d.). Stripe API Documentation.
Retrieved from <https://stripe.com/docs>
(Accessed: December 16, 2025).

6. JSON Web Token Documentation. (n.d.). Introduction to JSON Web Tokens.

Retrieved from <https://jwt.io/introduction>
(Accessed: December 16, 2025).

7. Redux Toolkit Documentation. (n.d.). Redux Toolkit Official Documentation.

Retrieved from <https://redux-toolkit.js.org/>
(Accessed: December 16, 2025).

8. Nodemailer Documentation. (n.d.). Nodemailer Usage and Configuration Guide.

Retrieved from <https://nodemailer.com/about/>
(Accessed: December 16, 2025).

APPENDICES

APPENDIX A

INSTALLATION AND EXECUTION PROCEDURE

The following steps describe the procedure to install and execute the “Stock & Shop” E-Commerce System on a local machine for evaluation and demonstration purposes.

Software Requirements

- Node.js (v16 or above)
- MongoDB
- Web Browser (Google Chrome / Microsoft Edge)
- Stripe CLI

Backend Setup

1. Navigate to the server folder.
2. Install required dependencies:

```
npm install
```

3. Create a .env file and configure the following:
 - MongoDB connection string
 - JWT secret key
 - Stripe secret key

4. Start the backend server:

```
npm run dev
```

The backend server runs on localhost:8080.

Frontend Setup

1. Navigate to the **client** folder.
2. Install required dependencies:

```
npm install
```

3. Start the frontend application:

```
npm run dev
```

Stripe Webhook Setup

To ensure secure and reliable payment confirmation during local development, **Stripe CLI** is used for webhook event forwarding.

1. Login to Stripe CLI:

```
stripe login
```

2. Forward webhook events to the backend server:

```
stripe listen --forward-to localhost:8080/api/webhook/stripe
```

This step is mandatory for successful order creation and cart clearance after payment.

Database Setup

MongoDB is used as the database to store user, product, cart, and order data. Database collections are created automatically when the application is executed.

APPENDIX B

GITHUB REPOSITORY

The complete source code for the “Stock & Shop” E-Commerce System is maintained using GitHub for version control and project management.

- **Repository Name:** Stock & Shop – MERN E-Commerce System
- **GitHub Link:**

<https://github.com/Saanvi-Rao/stock-and-shop-mern-ecommerce>

The repository follows a modular structure with separate frontend and backend folders, including controllers, routes, database schemas, Redux state management files, and utility functions required for system execution.

APPENDIX C

SAMPLE LOGIN CREDENTIALS AND SYSTEM ACCESS

The following login credentials and access instructions are provided strictly for evaluation and demonstration purposes.

Administrator Access

Faculty members can use the following credentials to access the Admin Panel and evaluate administrative functionalities such as category management, product upload, inventory control, and order monitoring.

- **Admin Email:** admin@stockshop.com
- **Admin Password:** Admin@123

Upon successful login, the system automatically redirects the administrator to the Admin Dashboard.

Customer (User) Access

To evaluate the Customer Panel, faculty members may register as a new user using their own email address.

Procedure:

1. Navigate to the Register page.
2. Enter name, email, and password.
3. Complete registration.
4. If the Forgot Password feature is tested, an OTP will be sent to the registered email address.
5. The OTP can be used to reset the password and continue further operations.

This ensures secure verification and demonstrates the OTP-based authentication flow implemented in the system.

Note:

The provided credentials and user registration flow are intended only for project evaluation and do not represent real or permanent user data.