

E-commerce Website

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

Bachelor of Computer Applications (BCA)

To Guru Gobind Singh Indraprastha University

Submitted to:

Ms. Mansi Vats

Assistant Prof.

Submitted By:

Sahil Manav

Roll No. 04611104424



Banarsidas Chandiwal Institute of Information Technology

New Delhi – 110019

Batch (2024-2026)

BCA-307

Acknowledgment

I would like to express my sincere gratitude to all those who contributed to the completion of this project.

Special appreciation goes to my project supervisor, **Ms. Mansi Vats**, for her guidance and mentorship throughout the project. Her insights and encouragement significantly enriched the quality of our work.

I'd also like to extend my thanks to my Director, **Dr. Ravish Saggarr**, for granting me this wonderful opportunity to be part of this project.

My gratitude extends to all our staff members for their invaluable assistance, guidance, and encouragement, which made this project possible.

My appreciation extends to the library staff for their vital support in accessing research materials.

I'd like to express my heartfelt gratitude to Guru Gobind Singh Indraprastha University for providing me the opportunity to collaborate on the "E-commerce Website".

Lastly, I want to acknowledge my family and friends for their unwavering support during the course of this project. Their encouragement kept me motivated, and their understanding during busy times was truly appreciated.

Thank you to everyone who played a part in making this project a success.

Sahil Manav

CERTIFICATE

This is to certify that this project entitled “E-commerce Website” submitted in partial fulfilment of the degree of Master of Computer Applications through Banarsidas Chandiwal Institute of Information Technology affiliated to Guru Gobind Singh Indraprastha University done by **Mr. Sahil Manav**, Roll No. 04611104424 is an authentic work carried out by him under the guidance of Ms. Mansi Vats. The matter embodied in this project work has not been submitted earlier for award of any degree to the best of my knowledge and belief.

Signature of the Student

Sahil Manav

Signature of the Guide

Ms. Mansi Vats

ABSTRACT

This project showcases the development of a dynamic e-commerce shopping website built using the MERN stack, which consists of Mongo DB, Express.js, React, and Node.js. The primary goal is to create a fully functional and user-friendly online store where users can browse products, add items to their cart, make secure purchases, and manage their profiles efficiently. The platform is designed to provide a seamless shopping experience with robust features such as real-time updates, personalized product recommendations, and secure transactions.

- Mongo DB is used as the database system, enabling flexible storage of product information, user data, orders, and transaction records in a No SQL format. This structure supports scalability and performance as the database grows.
- Express.js serves as the backend framework, facilitating the routing, HTTP request handling, and communication between the frontend and the database. It helps in managing the server-side logic efficiently.
- React powers the frontend, providing an interactive and responsive user interface. React's component-based architecture ensures that the website is highly modular, reusable, and fast, with a seamless user experience on both desktop and mobile devices.
- Node.js is the server-side runtime environment that processes client requests, handles business logic, and interfaces with the database. Node.js allows for the handling of multiple requests simultaneously, ensuring the website's performance under heavy traffic.

The e-commerce platform features essential functionalities such as user authentication (sign up, login), product catalog browsing, search filters, cart management, order history, and an integrated payment system for secure online transactions. The website ensures user data protection through secure password hashing and token-based authentication for session management.

This project demonstrates the full potential of the MERN stack in building modern, scalable, and high-performance e-commerce websites. The architecture is modular and can be easily extended with additional features like product reviews, ratings, real-time stock updates, and advanced admin dashboards, making it a versatile solution for online retail businesses.

OBJECTIVE

1. **User-Friendly Interface:**

- To provide a seamless, intuitive, and responsive interface that enhances the user experience, allowing customers to easily browse, search, and purchase products.

2. **Secure User Authentication:**

- To implement a secure and reliable user registration, login, and account management system, ensuring the privacy and safety of customer data through encryption and authentication methods.

3. **Product Catalog Management:**

- To display a wide range of products in a structured and easy-to-navigate format, allowing customers to filter, sort, and view product details (such as price, description, and images).

4. **Shopping Cart Functionality:**

- To develop a functional shopping cart system where users can add, remove, or modify the quantity of items and view the total price before proceeding to checkout.

5. **Secure Payment Gateway Integration:**

- To integrate secure and trusted payment methods (e.g., credit/debit cards, PayPal, or other digital payment systems), ensuring that users can safely complete their transactions.

6. **Order Management:**

- To allow users to track their orders, view purchase history, and manage returns or exchanges.

7. **Scalability and Performance:**

- To ensure that the website can handle a large number of users and products without compromising performance, ensuring fast page load times and minimal downtime.

TABLE OF CONTENTS

CHAPTER NO.	CONTENT	PAGE NO.
1.	Introduction 1.1 Problem Statement 1.2 Proposed Solution	1-3
2.	Analysis and System Requirements 2.1 Existing And Proposed System 2.2 Software and Hardware Requirements	4-6
3.	System Design And Modeling 3.1 Preliminary Design 3.1.1 Use Case Diagram 3.1.2 Data Flow Diagram 3.1.3 Class Diagram 3.1.4 ER Diagram 3.1.5 Activity Diagram	7-13
4.	Implementation 4.1 Implementation of Operations 4.2 Algorithm or Pseudocode of Implementation	14-17
5.	Source Code	18-48

6.	Models of E Commerce Shopping Website 5.1 Features 5.2 Company model 5.3 Shopping model 5.4 User model	49-55
7.	User Interface And Snapshots	56-63
8.	Future Scope	64-66
9.	Feasibility Study	67-69
10.	Limitations	70-72
11.	Conclusion	73-74
12.	Bibliography	75

CHAPTER 1

INTRODUCTION

An e-commerce shopping website is an online platform that facilitates the buying and selling of goods and services over the internet. As a digital storefront, it allows businesses to reach a global audience while providing customers with a convenient and efficient way to shop from anywhere at any time. The rise of internet usage, advancements in payment systems, and a shift in consumer behavior towards online shopping have made e-commerce websites essential in the modern retail landscape.

The primary function of an e-commerce shopping website is to provide a seamless shopping experience. This includes features such as product catalogs, user registration and login, secure payment gateways, and order management systems. Additionally, these websites are designed to showcase products, enable easy navigation, and offer features like product search, categorization, and filtering, helping customers find items quickly and easily.

The development of e-commerce websites using modern technologies like the ****MERN stack**** (MongoDB, Express.js, React, and Node.js) has revolutionized how online stores are built. The MERN stack is particularly well-suited for creating full-stack applications, allowing developers to build scalable and dynamic websites that are responsive and user-friendly. This stack provides robust functionality, from managing databases and handling server-side operations to creating rich, interactive user interfaces.

An e-commerce shopping website offers a wide range of benefits, such as increased customer reach, improved operational efficiency, reduced costs, and the ability to provide personalized shopping experiences. Whether it is a small business or a large corporation, an e-commerce platform enables companies to expand their market presence while providing users with a secure, engaging, and convenient way to shop for products and services.

This project aims to create an e-commerce shopping website with essential features like secure user authentication, product management, a shopping cart system, and an integrated payment gateway, making it an ideal solution for businesses looking to establish an online presence and customers seeking a reliable and efficient online shopping experience.

1.1 PROBLEM STATEMENT

The increasing shift toward online shopping has created a need for businesses to establish a strong digital presence. However, many businesses still face significant challenges in developing efficient, user-friendly, and secure e-commerce platforms. Current e-commerce websites often struggle with issues such as poor user experience, slow loading times, inadequate product management, security vulnerabilities, and complex transaction processes. Additionally, customers often experience difficulties with navigating product catalogs, managing their shopping carts, or making secure payments, leading to abandoned purchases and a loss of sales.

The primary problems that need to be addressed in the development of an e-commerce shopping website are as follows:

1. User Experience Challenges:

Many e-commerce websites are not intuitive or responsive, which makes it difficult for customers to find products, view details, or complete purchases seamlessly across devices (e.g., desktop, tablet, mobile).

2. Security Concerns:

Security breaches, data theft, and payment fraud are significant concerns for both businesses and customers. A lack of robust security measures can lead to loss of trust, compromised user data, and unauthorized transactions.

3. Scalability Issues:

As an e-commerce website grows in terms of products, customers, and traffic, it often faces performance issues, such as slow page load times or system crashes, which can disrupt the shopping experience and hinder business growth.

4. Product Management and Cataloging:

Managing a large inventory of products with detailed descriptions, prices, stock levels, and categories can be complex. Without an efficient way to update and display this information, businesses may struggle to keep their product offerings organized and accessible.

5. Inefficient Checkout Process:

Many online shopping platforms have complicated or lengthy checkout processes, which can cause potential customers to abandon their carts before completing a purchase. A lack of simplified payment methods and account management further contributes to this problem.

6. **Payment Gateway Integration:**

Inadequate or unreliable payment gateway systems can result in failed transactions or poor customer satisfaction. Additionally, some systems may not support multiple payment methods, limiting customer flexibility.

7. **Limited Admin and Order Management:**

Administrators may face challenges in managing user orders, processing refunds, and handling inventory updates, which can lead to delays, errors, and poor customer service.

1.2 **Proposed Solution:**

This e-commerce shopping website aims to address these challenges by creating a scalable, secure, and user-friendly platform. The website will leverage modern technologies such as the MERN stack (Mongo DB, Express.js, React, Node.js) to provide fast performance, intuitive navigation, and an efficient backend for product and order management. The solution will also focus on:

1. **Enhanced User Experience:** A clean, responsive, and easy-to-navigate interface that works across all devices, ensuring a seamless shopping experience.
2. **Robust Security:** Implementation of secure user authentication, data encryption, and trusted payment gateways to safeguard user information and financial transactions.
3. **Scalability:** A well-structured architecture that ensures the website performs well even with high traffic and large product inventories.
4. **Streamlined Checkout Process:** Simplified, fast, and secure checkout, offering multiple payment options for convenience.
5. **Efficient Admin Panel:** An intuitive dashboard for administrators to manage products, process orders, and maintain inventory with ease.

By addressing these core issues, the e-commerce shopping website will offer a solution that improves customer satisfaction, drives conversions, and supports business growth.

CHAPTER 2

ANALYSIS AND SYSTEM REQUIREMENTS

2.1 Existing and Proposed System

Existing System:

In many traditional and existing e-commerce systems, businesses often face various limitations and challenges. These existing systems are typically built with older technologies or may not be optimized for modern requirements like scalability, performance, or security.

Proposed System:

The proposed e-commerce shopping website seeks to address the limitations of existing systems by using modern technologies, ensuring better user experience, security, performance, and administrative control.

Key Features of the proposed system

☐ **Enhanced User Experience (UX):**

- The website will have a **responsive design** built with **React** to ensure a smooth experience on any device—desktop, tablet, or mobile.
- The user interface will be clean, modern, and intuitive, with easy-to-use navigation, personalized product recommendations, and dynamic product filtering and sorting options.

☐ **Robust Security Features:**

- **Token-based authentication** (e.g., JSON Web Tokens) will be used to securely manage user sessions, preventing unauthorized access to accounts.
- **Secure Payment Gateways** like Stripe, PayPal, and credit card integrations will ensure safe and seamless financial transactions.

☐ **Scalability and Performance:**

- The platform will be built using the **MERN stack** (Mongo DB, Express.js, React, Node.js), which is highly scalable and supports fast data processing, even with a large number of users and products.
- **Mongo DB** is designed for handling high volumes of data, while **Node.js** allows for asynchronous operations, ensuring high concurrency without performance degradation.

□ **Streamlined Checkout Process:**

- The **checkout process** will be simplified, with a minimal number of steps to reduce friction and prevent cart abandonment.
- Multiple **payment options** (credit/debit cards, PayPal, wallets, etc.) will be integrated, offering flexibility to users.

□ **Efficient Admin Panel:**

- A **user-friendly admin dashboard** will be created, enabling easy management of products, orders, customer data, and inventory.
- **Real-time updates** will allow administrators to manage product availability, update prices, and launch promotions instantly.

Objectives of the Proposed system:

The objective of the proposed e-commerce shopping website is to create a secure, scalable, and user-friendly platform that provides an enhanced shopping experience for customers while enabling efficient management for administrators. The system aims to offer a seamless, responsive design with personalized features, a simplified and secure checkout process, real-time inventory management, and robust security measures. It will also integrate multiple payment options, real-time product updates, and an intuitive admin dashboard for product and order management. By focusing on performance optimization, secure transactions, and customer engagement, the proposed system strives to increase customer satisfaction, reduce cart abandonment, and support business growth.

2.2 Software and Hardware Requirements

Hardware Requirements:

- **Processor:** Any modern CPU (e.g., Intel Core i3, i5, or equivalent) should be sufficient for development purposes.
- **RAM:** A minimum of **8 GB** of RAM is recommended to handle development tools, databases, and IDEs simultaneously without lag.
- **Storage:** At least **100 GB** of available storage for installing development tools, dependencies, and managing project files. SSD storage is preferred for faster read/write operations.
- **Graphics:** Basic integrated graphics (e.g., Intel UHD Graphics) should suffice, as the development process doesn't require high-performance graphics.
- **Network:** A stable and fast **internet connection** is necessary for development and deployment, especially if using cloud services for hosting and database management.

Software Requirements:

OS: Windows, macOS, or Linux (any of these operating systems can be used for both development and deployment depending on personal preference and project needs).

Code: Visual Studio Code (VS Code) – Popular code editor with extensive support for JavaScript, Node.js, and React.

Backend Development:

- ☐ **Node.js:** The runtime environment for running JavaScript on the server-side. Node.js is required for developing the backend of the application and integrating it with Express.js.
- ☐ **Express.js:** A web application framework for Node.js, used to handle routing, server-side logic, and API endpoints for your e-commerce website.
- ☐ **Mongo DB:** A NoSQL database that will store user data, product information, orders, and transaction details. MongoDB is highly scalable and works well with Node.js applications.
- ☐ **Mongoose:** A MongoDB object modeling tool for Node.js, providing an easy way to interact with the MongoDB database through schemas and models.

Frontend Development:

- **React.js:** A JavaScript library for building the user interface (UI). React allows for the creation of dynamic, interactive, and responsive web pages with fast rendering.
- **React Router:** A library for handling routing in React applications, enabling smooth navigation between pages (e.g., product pages, cart, checkout).
- **Redux** (optional): A state management library for managing the application's state in large-scale React applications, useful for managing user sessions, cart data, and product listings.

CHAPTER 3

SYSTEM DESIGN AND MODELING

3.1 Preliminary Design

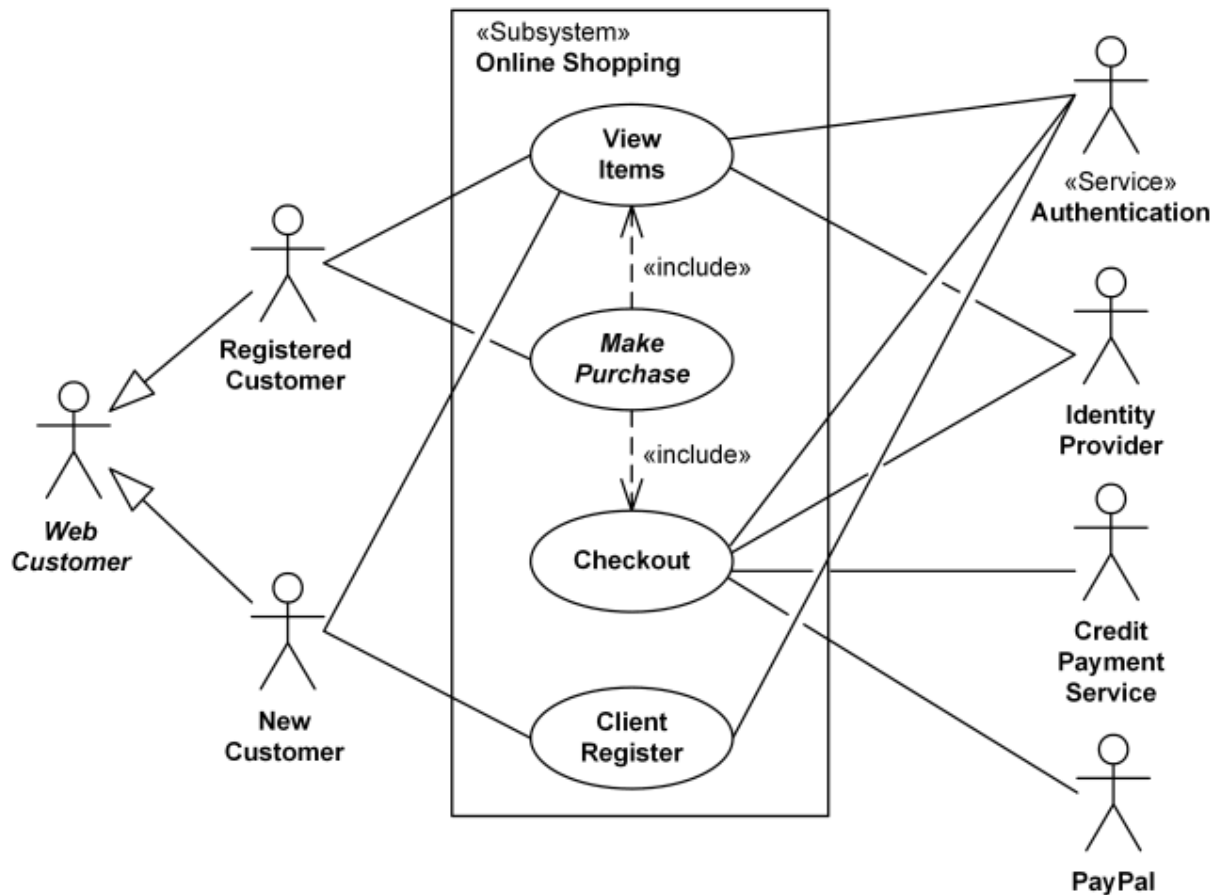
The preliminary design of an e-commerce shopping website using the MERN stack (MongoDB, Express, React, Node.js) focuses on creating a seamless, user-friendly platform that integrates frontend, backend, and database efficiently. The architecture is based on a client-server model, where React handles the frontend, providing dynamic content rendering and a responsive UI. It interacts with a Node.js server running Express.js to manage requests and perform backend operations, such as user authentication, product management, and order processing. MongoDB, a No SQL database, is used to store critical data like user profiles, product listings, order histories, and transaction details, allowing for flexibility and scalability in data handling.

The user interface (UI) design prioritizes ease of use and a visually appealing layout. The homepage features product categories, search functionality, and promotional banners, allowing users to quickly find products and navigate through the store. Product pages display detailed information, including images, descriptions, and pricing, with a simple and intuitive "Add to Cart" button. The website is designed to be fully responsive, ensuring it is accessible and user-friendly across devices like smartphones, tablets, and desktops.

On the backend, the Node.js server with Express.js handles the business logic and API endpoints, providing functionality for user registration, login, product management, and order processing. JWT (JSON Web Tokens) are used for secure user authentication, enabling users to log in and access personalized services like order tracking and wishlists. The backend also integrates with payment gateways like Stripe or PayPal for secure transaction processing, ensuring that all financial data is handled safely.

The admin dashboard allows website administrators to manage products, view customer orders, and update inventory in real-time. Admin users can add new products, edit product details, and track order statuses, providing control over the e-commerce site's operations. This dashboard is designed to be simple yet comprehensive, providing analytics on sales, inventory, and customer behavior. The overall design ensures that the website is scalable, secure, and capable of handling increasing traffic and data, making it suitable for long-term e-commerce growth.

3.1.1 Use Case Diagram



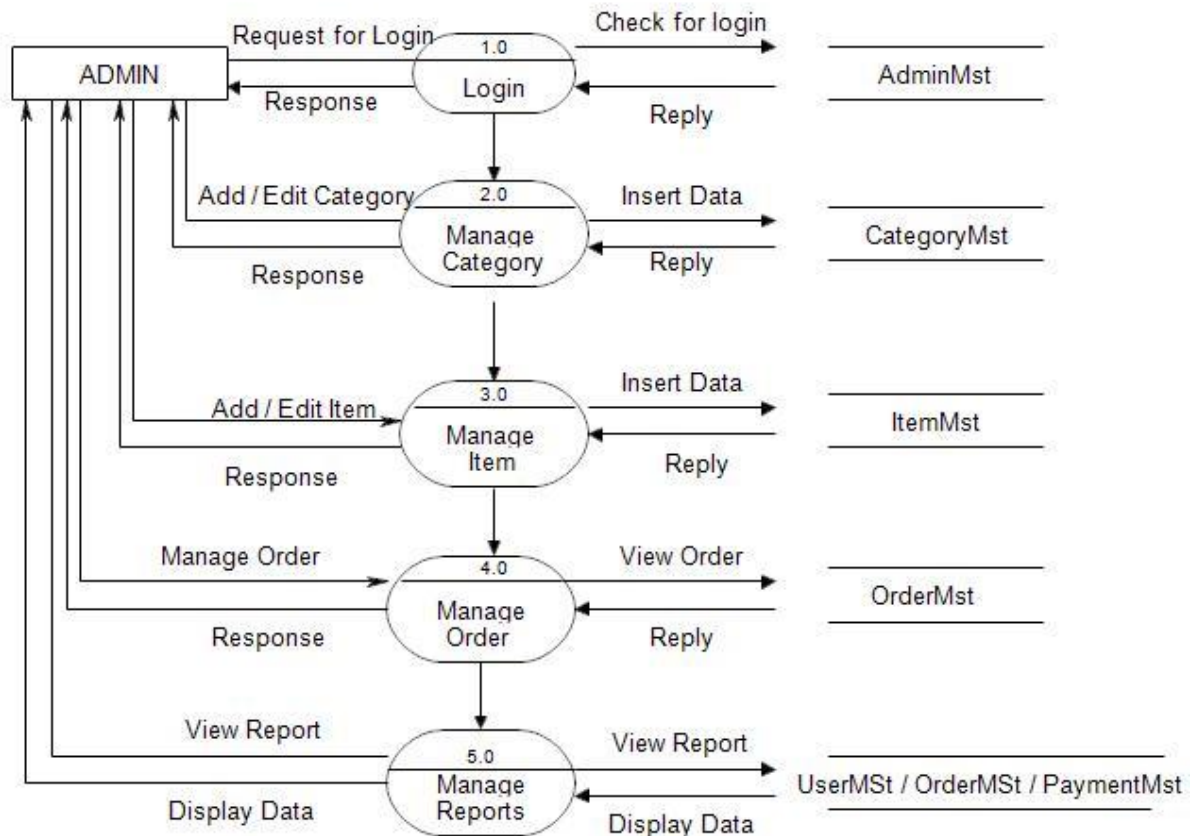
3.1.2 Data Flow Diagram

0-Level DFD

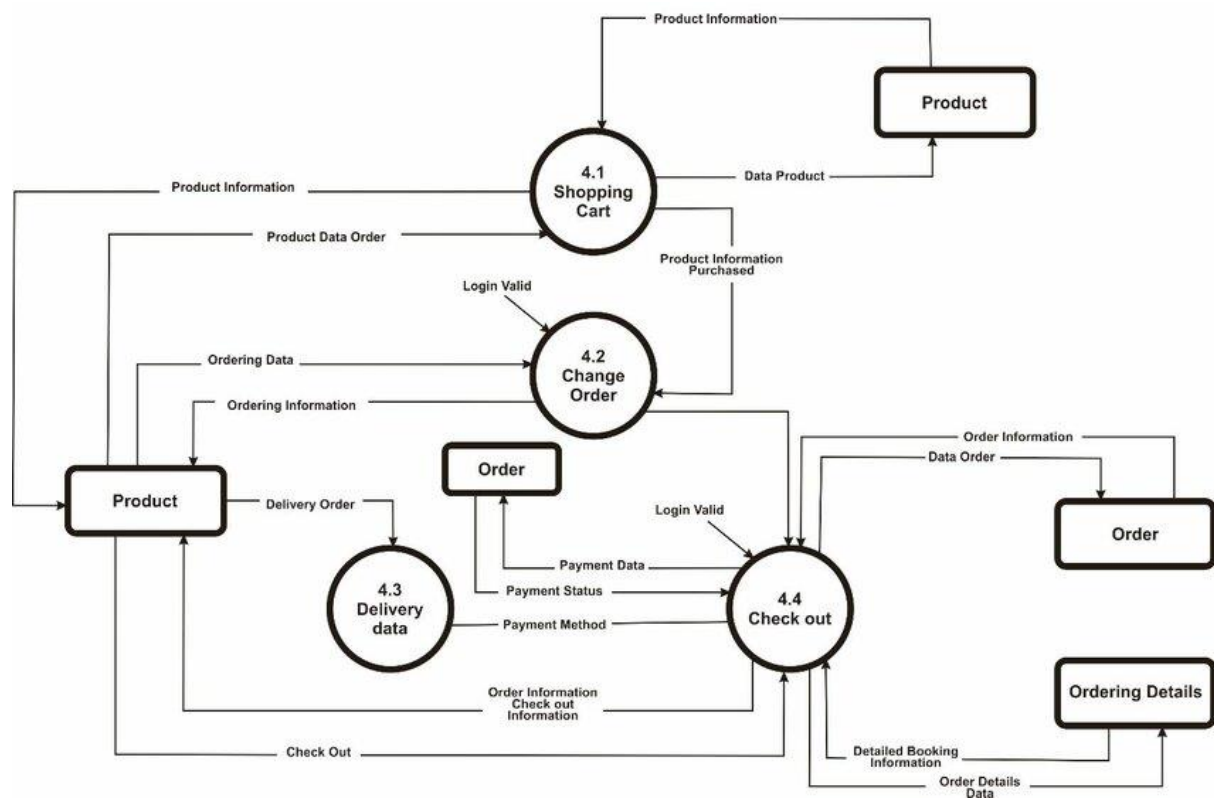


1-Level DFD

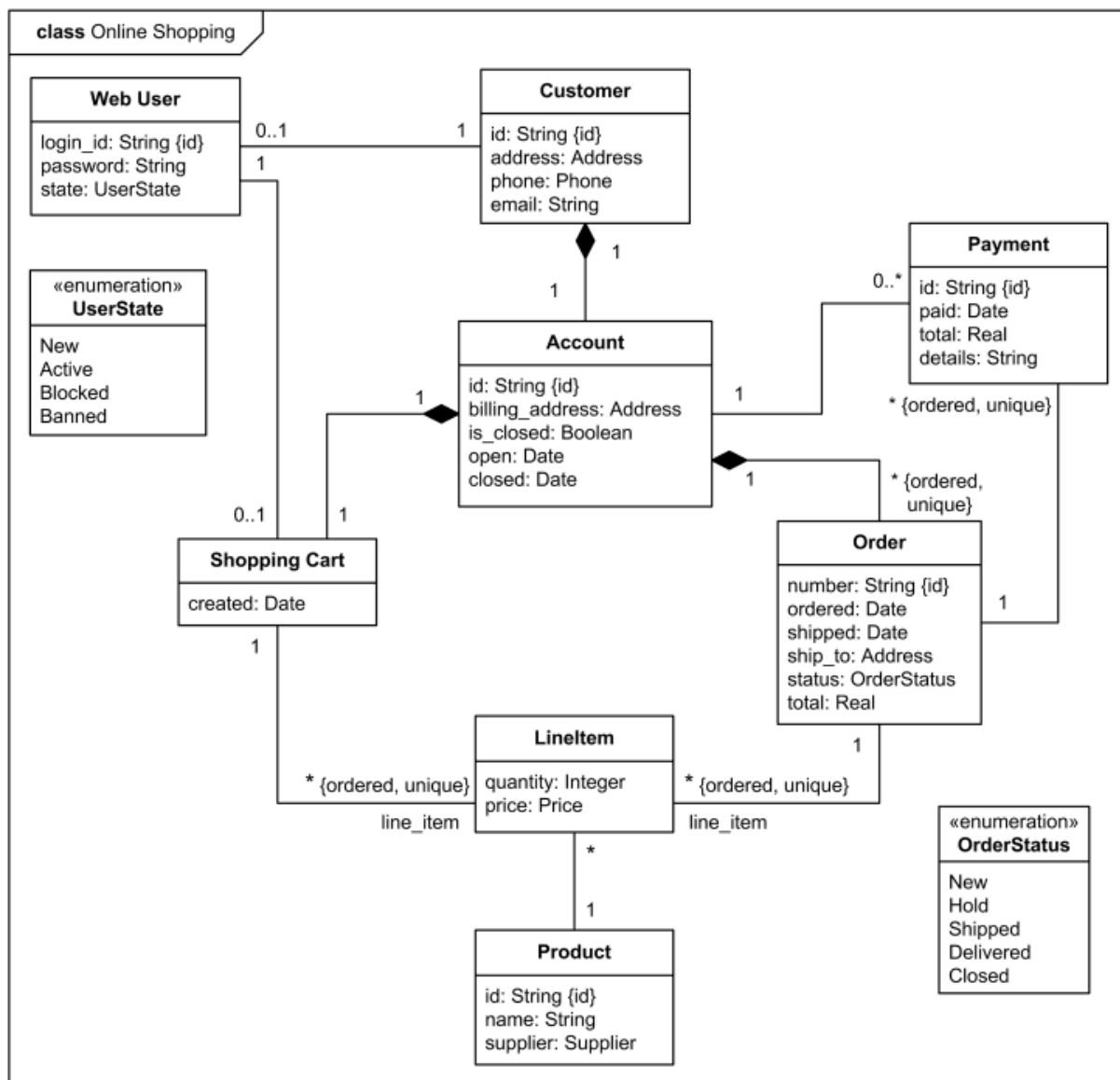
Admin Side DFD - 1st Level



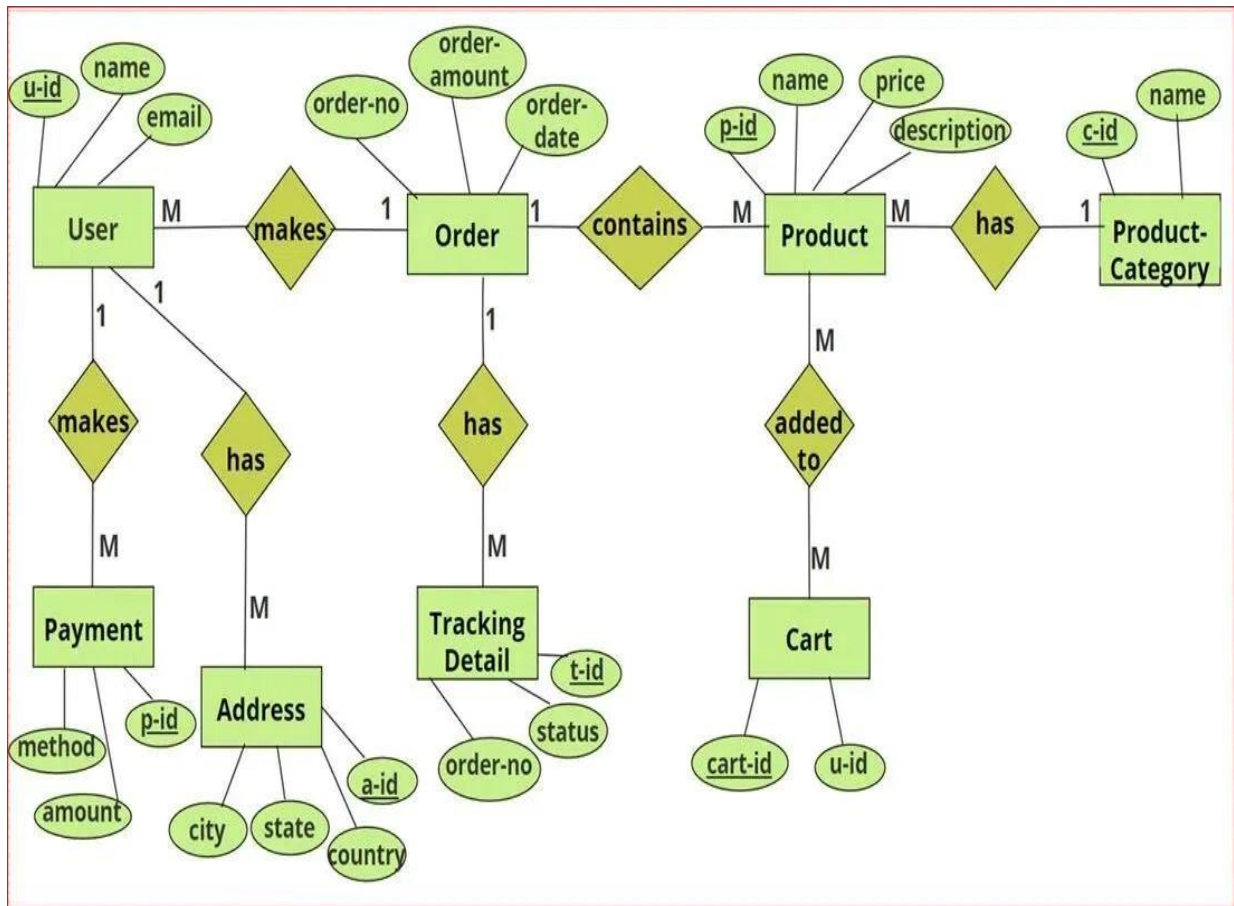
2-Level DFD



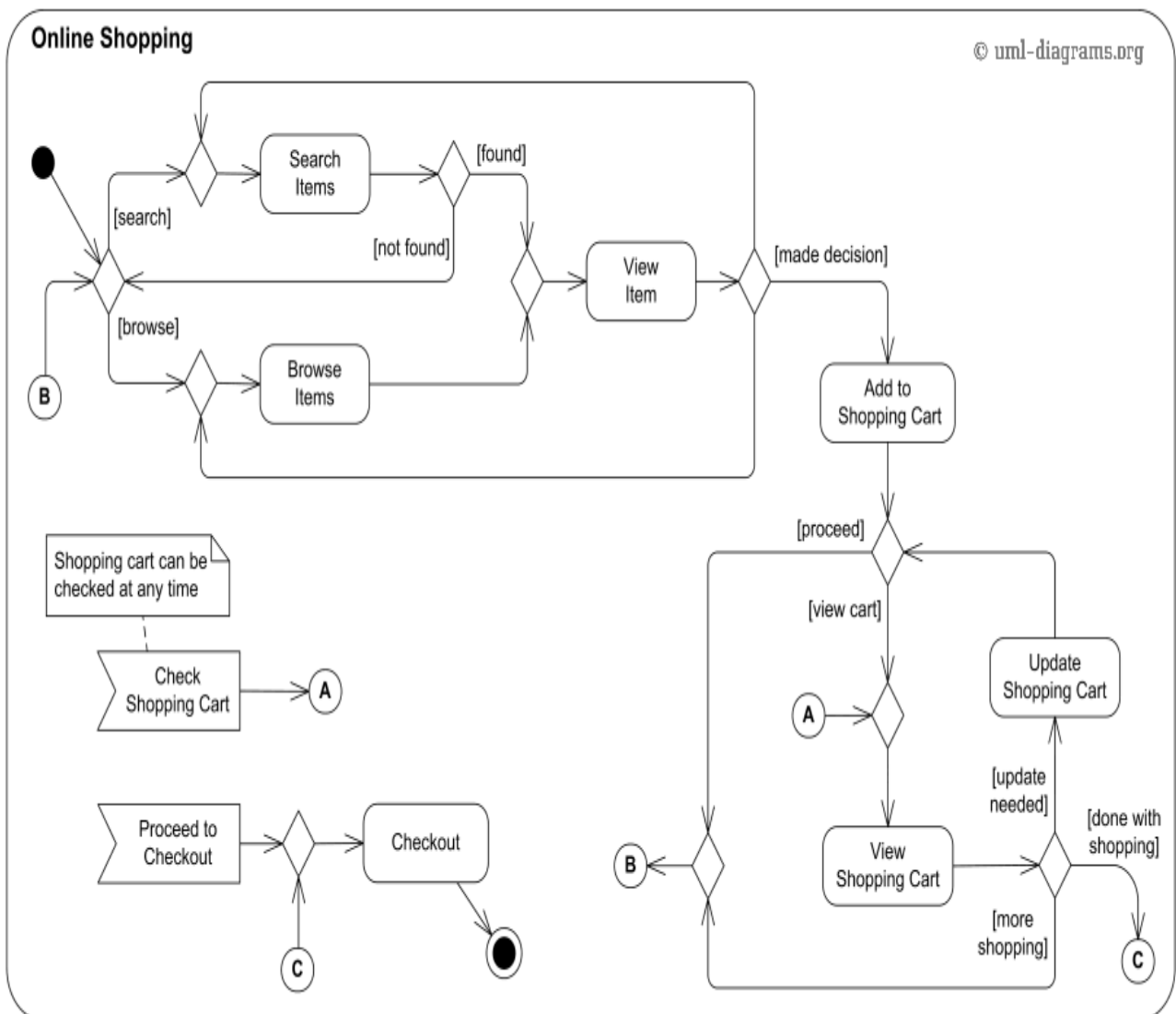
3.1.3 Class Diagram



3.1.4 Entity Relationship Diagram



3.1.5 Activity Diagram



CHAPTER 4

IMPLEMENTATION

4.1 Implementation of Operations

1. User Authentication and Authorization:

User authentication is implemented using JWT (JSON Web Tokens). When users register or log in, their credentials are validated by the backend (Node.js/Express), and a token is generated for session management. This token is stored in the user's browser and used for secure access to protected routes, like profile and order history.

2. Product Management:

Admins can perform CRUD (Create, Read, Update, Delete) operations on products. The backend (Express.js) provides API routes for adding new products, updating existing ones, and deleting products. Product data, including names, prices, descriptions, and images, is stored in MongoDB and fetched by the frontend (React) for display.

3. Shopping Cart Management:

Users can add products to their shopping cart, update quantities, or remove items. The cart is stored either in the session (for guest users) or in the database (for logged-in users). React handles the cart interface, and the backend updates the cart in real time.

4. Order Management:

After checkout, the backend processes the order, stores it in the database, and sends an order confirmation to the user. The order details, including product IDs, quantities, shipping information, and payment status, are stored in MongoDB. Admins can view and update the status of orders (e.g., "shipped", "delivered") through the admin dashboard.

5. Payment Integration:

Payment is integrated with services like Stripe or PayPal. During checkout, the frontend collects payment details, which are sent securely to the payment gateway. After payment confirmation, the backend updates the order status and notifies the user of successful payment.

6. Admin Dashboard:

Admins can manage products, view orders, and monitor website performance via a dashboard. The dashboard provides insights into sales, customer behavior, and order processing, allowing for efficient site management. Admin actions are secured through role-based access control.

4.2 Algorithm or Pseudocode of Implementation

1. User Authentication (Login/Registration)

Algorithm:

- 1. User Registration:**
 - Input: User's name, email, password
 - Validate input (check if the email is already registered)
- 2. User Login:**
 - Input: User's email and password
 - Fetch user details from the database using email

2. Product Management (Admin CRUD Operations)

Algorithm:

- 1. Add Product:**
 - Input: Product name, description, price, category, image
 - Validate product data
 - 2. Update Product:**
 - Input: Product ID, updated details (name, description, price, etc.)
 - Find product by ID in the database
 - 3. Delete Product:**
 - Input: Product ID
 - Find and delete the product from MongoDB
 - 4. Fetch Products:**
 - Input: (Optional) Filters like category, price range, etc.
 - Query the database for products based on filters
-

3. Shopping Cart Management (For User)

Algorithm:

1. **Add Product to Cart:**
 - Input: Product ID, quantity
 - Check if user is logged in:
 - If logged in, store cart data in the user's database entry (MongoDB)
 - If not logged in, store cart in session/cookies on the frontend
 2. **Update Cart:**
 - Input: Product ID, new quantity
 - Find the product in the cart
 3. **Remove Product from Cart:**
 - Input: Product ID
 - Remove the product from the cart
 4. **View Cart:**
 - Fetch the cart items (from MongoDB or session)
 - Display product names, prices, quantities, and total cost
-

4. Checkout and Order Processing

Algorithm:

1. **Initiate Checkout:**
 - Input: User's shipping address, payment details
 - Validate input (check for required fields like address, payment method)
 2. **Process Payment:**
 - Input: Payment details (credit card info, PayPal, etc.)
 - Send payment request to the payment gateway (Stripe/PayPal)
 - If payment successful:
 - Create a new order in the database (store product details, shipping address, and payment status)
 - Mark order as 'paid' and return success message
 - If payment fails, return error message
 3. **Generate Order Confirmation:**
 - Display order confirmation details (order ID, products purchased, shipping info)
 - Send confirmation email to the user
 4. **Update Product Inventory:**
 - For each product in the order:
 - Reduce the available stock in MongoDB by the quantity purchased
-

5. Admin Dashboard (Order Management)

Algorithm:

1. **View All Orders:**
 - Fetch all orders from the database
 - Display order details (order ID, customer info, order status)
2. **Update Order Status:**
 - Input: Order ID, new status (e.g., 'Shipped', 'Delivered')
 - Find the order in the database by ID
 - Update the order status
 - Return success message
3. **Track Sales and Analytics:**
 - Query the database for total sales, number of orders, or other performance data
 - Return insights for admin to analyze sales performance

CHAPTER 5

SOURCE CODE

App.jsx

```
import React from 'react'
import { Routes, Route } from 'react-router-dom'
import Home from './pages/Home'
import Collection from './pages/Collection'
import About from './pages/About'
import Contact from './pages/Contact'
import Product from './pages/Product'
import Cart from './pages/Cart'
import Login from './pages/Login'
import PlaceOrder from './pages/PlaceOrder'
import Orders from './pages/Orders'
import Navbar from './components/Navbar'
import Footer from './components/Footer'
import SearchBar from './components/SearchBar'
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import Verify from './pages/Verify'

const App = () => {
  return (
    <div className='px-4 sm:px-[5vw] md:px-[7vw] lg:px-[9vw]'>
      <ToastContainer />
      <Navbar />
      <SearchBar />
```

```

<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/collection" element={<Collection />} />
  <Route path="/about" element={<About />} />
  <Route path="/contact" element={<Contact />} />
  <Route path="/product/:productId" element={<Product />} />
  <Route path="/cart" element={<Cart />} />
  <Route path="/login" element={<Login />} />
  <Route path="/place-order" element={<PlaceOrder />} />
  <Route path="/orders" element={<Orders />} />
  <Route path="/verify" element={<Verify />} />
</Routes>
<Footer />
</div>
)
}

export default App

```

Main.jsx

```
import React from 'react'
```

```
import ReactDOM from 'react-dom/client'
```

```
import App from './App.jsx'
```

```
import './index.css'
```

```
import { BrowserRouter } from 'react-router-dom'
```

```
import ShopContextProvider from './context/ShopContext.jsx'
```

```
ReactDOM.createRoot(document.getElementById('root')).render(
```

```
  <BrowserRouter>
```

```
    <ShopContextProvider>
```

```
      <App />
```

```
    </ShopContextProvider>
```

```
  </BrowserRouter>,
```

```
)
```

PAGES

Login.jsx

```
import axios from 'axios'

import React, { useState } from 'react'

import { backendUrl } from '../App'

import { toast } from 'react-toastify'

const Login = ({setToken}) => {

  const [email,setEmail] = useState("")
  const [password,setPassword] = useState("")

  const onSubmitHandler = async (e) => {
    try {
      e.preventDefault();
      const response = await axios.post(backendUrl + '/api/user/admin',{email,password})
      if (response.data.success) {
        setToken(response.data.token)
      } else {
        toast.error(response.data.message)
      }
    } catch (error) {
      console.log(error);
      toast.error(error.message)
    }
  }

  return (
    <div className='min-h-screen flex items-center justify-center w-full'>
```

```

<div className='bg-white shadow-md rounded-lg px-8 py-6 max-w-md'>

  <h1 className='text-2xl font-bold mb-4'>Admin Panel</h1>

  <form onSubmit={onSubmitHandler}>

    <div className='mb-3 min-w-72'>

      <p className='text-sm font-medium text-gray-700 mb-2'>Email Address</p>

      <input onChange={(e)=>setEmail(e.target.value)} value={email} className='rounded-md
w-full px-3 py-2 border border-gray-300 outline-none' type="email" placeholder='your@email.com'
required />

    </div>

    <div className='mb-3 min-w-72'>

      <p className='text-sm font-medium text-gray-700 mb-2'>Password</p>

      <input onChange={(e)=>setPassword(e.target.value)} value={password}
className='rounded-md w-full px-3 py-2 border border-gray-300 outline-none' type="password"
placeholder='Enter your password' required />

    </div>

    <button className='mt-2 w-full py-2 px-4 rounded-md text-white bg-black' type="submit">
Login </button>

  </form>

</div>

)
}

export default Login

```

About.jsx

```
import React from 'react'

import Title from '../components/Title'

import { assets } from '../assets/assets'

import NewsletterBox from '../components/NewsletterBox'

const About = () => {
  return (
    <div>

      <div className='text-2xl text-center pt-8 border-t'>
        <Title text1={'ABOUT'} text2={'US'} />
      </div>

      <div className='my-10 flex flex-col md:flex-row gap-16'>
        <img className='w-full md:max-w-[450px]' src={assets.about_img} alt="" />
        <div className='flex flex-col justify-center gap-6 md:w-2/4 text-gray-600'>
          <p>Forever was born out of a passion for innovation and a desire to revolutionize the way people shop online. Our journey began with a simple idea: to provide a platform where customers can easily discover, explore, and purchase a wide range of products from the comfort of their homes.</p>
          <p>Since our inception, we've worked tirelessly to curate a diverse selection of high-quality products that cater to every taste and preference. From fashion and beauty to electronics and home essentials, we offer an extensive collection sourced from trusted brands and suppliers.</p>
          <b className='text-gray-800'>Our Mission</b>
          <p>Our mission at Forever is to empower customers with choice, convenience, and confidence. We're dedicated to providing a seamless shopping experience that exceeds expectations, from browsing and ordering to delivery and beyond.</p>
        </div>
      </div>

      <div className=' text-xl py-4'>
        <Title text1={'WHY'} text2={'CHOOSE US'} />
      </div>
    </div>
  )
}
```

</div>

<div className='flex flex-col md:flex-row text-sm mb-20'>

<div className='border px-10 md:px-16 py-8 sm:py-20 flex flex-col gap-5'>

Quality Assurance:

<p className=' text-gray-600'>We meticulously select and vet each product to ensure it meets our stringent quality standards.</p>

</div>

<div className='border px-10 md:px-16 py-8 sm:py-20 flex flex-col gap-5'>

Convenience:

<p className=' text-gray-600'>With our user-friendly interface and hassle-free ordering process, shopping has never been easier.</p>

</div>

<div className='border px-10 md:px-16 py-8 sm:py-20 flex flex-col gap-5'>

Exceptional Customer Service:

<p className=' text-gray-600'>Our team of dedicated professionals is here to assist you the way, ensuring your satisfaction is our top priority.</p>

</div>

</div>

<NewsletterBox/>

</div>

)

}

export default About

Cart.jsx

```
import React, { useContext, useEffect, useState } from 'react'

import { ShopContext } from '../context/ShopContext'

import Title from '../components/Title';

import { assets } from '../assets/assets';

import CartTotal from '../components/CartTotal';


const Cart = () => {

  const { products, currency, cartItems, updateQuantity, navigate } = useContext(ShopContext);

  const [cartData, setCartData] = useState([]);

  useEffect(() => {

    if (products.length > 0) {
      const tempData = [];
      for (const items in cartItems) {
        for (const item in cartItems[items]) {
          if (cartItems[items][item] > 0) {
            tempData.push({
              _id: items,
              size: item,
              quantity: cartItems[items][item]
            })
          }
        }
      }
      setCartData(tempData);
    }
  }, [cartItems, products])
```



```

return (
  <div className='border-t pt-14'>

    <div className=' text-2xl mb-3'>
      <Title text1={'YOUR'} text2={'CART'} />
    </div>

    <div>
      {
        cartData.map((item, index) => {

          const productData = products.find((product) => product._id === item._id);

          return (
            <div key={index} className='py-4 border-t border-b text-gray-700 grid grid-cols-[4fr_0.5fr_0.5fr] sm:grid-cols-[4fr_2fr_0.5fr] items-center gap-4'>
              <div className=' flex items-start gap-6'>
                <img className='w-16 sm:w-20' src={productData.image[0]} alt="" />
                <div>
                  <p className='text-xs sm:text-lg font-medium'>{productData.name}</p>
                  <div className='flex items-center gap-5 mt-2'>
                    <p>{currency}{productData.price}</p>
                    <p className='px-2 sm:px-3 sm:py-1 border bg-slate-50'>{item.size}</p>
                  </div>
                </div>
              </div>

              <input onChange={(e) => e.target.value === '' || e.target.value === '0' ? null :
updateQuantity(item._id, item.size, Number(e.target.value))} className='border max-w-10 sm:max-w-20 px-1 sm:px-2 py-1' type="number" min={1} defaultValue={item.quantity} />

              <img onClick={() => updateQuantity(item._id, item.size, 0)} className='w-4 mr-4 sm:w-5 cursor-pointer' src={assets.bin_icon} alt="" />
            </div>
          )
        })
      }
    </div>
  </div>
)

```

```

        </div>
    )

    })
  }
</div>

<div className='flex justify-end my-20'>
  <div className='w-full sm:w-[450px]'>
    <CartTotal />
    <div className=' w-full text-end'>
      <button onClick={() => navigate('/place-order')} className='bg-black text-white text-sm my-8
px-8 py-3'>PROCEED TO CHECKOUT</button>
    </div>
  </div>
</div>

</div>
)
}

export default Cart

```

Collection.jsx

```
import React, { useContext, useEffect, useState } from 'react'

import { ShopContext } from '../context/ShopContext'

import { assets } from '../assets/assets';

import Title from '../components/Title';

import ProductItem from '../components/ProductItem';


const Collection = () => {

  const { products , search , showSearch } = useContext(ShopContext);

  const [showFilter,setShowFilter] = useState(false);

  const [filterProducts,setFilterProducts] = useState([]);

  const [category,setCategory] = useState([]);

  const [subCategory,setSubCategory] = useState([]);

  const [sortType,setSortType] = useState('relavent')


  const toggleCategory = (e) => {

    if (category.includes(e.target.value)) {

      setCategory(prev=> prev.filter(item => item !== e.target.value))

    }

    else{

      setCategory(prev => [...prev,e.target.value])

    }

  }


  const toggleSubCategory = (e) => {

    if (subCategory.includes(e.target.value)) {

      setSubCategory(prev=> prev.filter(item => item !== e.target.value))

    }

    else{

      setSubCategory(prev => [...prev,e.target.value])

    }

  }

}
```

```
}
```

```
const applyFilter = () => {
```

```
  let productsCopy = products.slice();
```

```
  if (showSearch && search) {
```

```
    productsCopy = productsCopy.filter(item =>  
item.name.toLowerCase().includes(search.toLowerCase()))
```

```
  }
```

```
  if (category.length > 0) {
```

```
    productsCopy = productsCopy.filter(item => category.includes(item.category));
```

```
  }
```

```
  if (subCategory.length > 0 ) {
```

```
    productsCopy = productsCopy.filter(item => subCategory.includes(item.subCategory))
```

```
  }
```

```
  setFilterProducts(productsCopy)
```

```
}
```

```
const sortProduct = () => {
```

```
  let fpCopy = filterProducts.slice();
```

```
  switch (sortType) {
```

```
    case 'low-high':
```

```
      setFilterProducts(fpCopy.sort((a,b)=>(a.price - b.price)));
```

```
      break;
```

```
    case 'high-low':
```

```
      setFilterProducts(fpCopy.sort((a,b)=>(b.price - a.price)));
```

```
      break;
```

```
    default:
```

```
      applyFilter();
```

```
      break;
```

```
  }
```

```

}

useEffect(()=>{
  applyFilter();
},[category,subCategory,search,showSearch,products])

useEffect(()=>{
  sortProduct();
},[sortType])

return (
  <div className='flex flex-col sm:flex-row gap-1 sm:gap-10 pt-10 border-t'>

    {/* Filter Options */}

    <div className='min-w-60'>
      <p onClick={()=>setShowFilter(!showFilter)} className='my-2 text-xl flex items-center cursor-pointer gap-2'>FILTERS
        <img className={`h-3 sm:hidden ${showFilter ? 'rotate-90' : ''}`} src={assets.dropdown_icon} alt="" />
      </p>
      {/* Category Filter */}
      <div className={`border border-gray-300 pl-5 py-3 mt-6 ${showFilter ? '' : 'hidden'} sm:block`}>
        <p className='mb-3 text-sm font-medium'>CATEGORIES</p>
        <div className='flex flex-col gap-2 text-sm font-light text-gray-700'>
          <p className='flex gap-2'>
            <input className='w-3' type="checkbox" value={'Men'} onChange={toggleCategory}/> Men
          </p>
          <p className='flex gap-2'>
            <input className='w-3' type="checkbox" value={'Women'} onChange={toggleCategory}/>
Women
          </p>
          <p className='flex gap-2'>

```



```

      <option value="relavent">Sort by: Relavent</option>
      <option value="low-high">Sort by: Low to High</option>
      <option value="high-low">Sort by: High to Low</option>
    </select>
  </div>

  { /* Map Products */}

  <div className='grid grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-4 gap-y-6'>
    {
      filterProducts.map((item,index)=>(
        <ProductItem key={index} name={item.name} id={item._id} price={item.price}
image={item.image} />
      ))
    }
  </div>
</div>

)
}

export default Collection

```

Contact.jsx

```
import React from 'react'

import Title from '../components/Title'

import { assets } from '../assets/assets'

import NewsletterBox from '../components/NewsletterBox'

const Contact = () => {
  return (
    <div>

      <div className='text-center text-2xl pt-10 border-t'>
        <Title text1={'CONTACT'} text2={'US'} />
      </div>

      <div className='my-10 flex flex-col justify-center md:flex-row gap-10 mb-28'>
        <img className='w-full md:max-w-[480px]' src={assets.contact_img} alt="" />
        <div className='flex flex-col justify-center items-start gap-6'>
          <p className='font-semibold text-xl text-gray-600'>Our Store</p>
          <p className='text-gray-500'>M/S Ciarr Industries pvt ltd. <br /> Nangal Raya, New Delhi-110046, India</p>
          <p className='text-gray-500'>Tel: (+91 8287710862) <br /> Email: care@forever.com</p>
          <p className='font-semibold text-xl text-gray-600'>Careers at Forever</p>
          <p className='text-gray-500'>Learn more about our teams and job openings.</p>
          <button className='border border-black px-8 py-4 text-sm hover:bg-black hover:text-white transition-all duration-500'>Explore Jobs</button>
        </div>
      </div>

      <NewsletterBox />
    </div>
  )
}

export default Contact
```


Home.jsx

```
import React from 'react'

import Hero from '../components/Hero'

import LatestCollection from '../components/LatestCollection'

import BestSeller from '../components/BestSeller'

import OurPolicy from '../components/OurPolicy'

import NewsletterBox from '../components/NewsletterBox'


const Home = () => {

  return (

    <div>

      <Hero />

      <LatestCollection/>

      <BestSeller/>

      <OurPolicy/>

      <NewsletterBox/>

    </div>

  )

}

export default Home
```

Orders.jsx

```
import React, { useContext, useEffect, useState } from 'react'

import { ShopContext } from '../context/ShopContext';

import axios from 'axios';

import { toast } from 'react-toastify';

const Login = () => {

  const [currentState, setCurrentState] = useState('Login');

  const { token, setToken, navigate, backendUrl } = useContext(ShopContext)

  const [name, setName] = useState("")

  const [password, setPasword] = useState("")

  const [email, setEmail] = useState("")

  const onSubmitHandler = async (event) => {

    event.preventDefault();

    try {

      if (currentState === 'Sign Up') {

        const response = await axios.post(backendUrl + '/api/user/register', {name, email, password})

        if (response.data.success) {

          setToken(response.data.token)

          localStorage.setItem('token', response.data.token)

        } else {

          toast.error(response.data.message)

        }

      } else {

        const response = await axios.post(backendUrl + '/api/user/login', {email, password})

        if (response.data.success) {

          setToken(response.data.token)

          localStorage.setItem('token', response.data.token)

        } else {

          toast.error(response.data.message)

        }

      }

    }

  }

}
```

```

    } catch (error) {
      console.log(error)
      toast.error(error.message)
    }
  }
  useEffect(()=>{
    if (token) {
      navigate('/')
    }
  },[token])
  return (
    <form onSubmit={onSubmitHandler} className='flex flex-col items-center w-[90%] sm:max-w-96 m-auto mt-14 gap-4 text-gray-800'>
      <div className='inline-flex items-center gap-2 mb-2 mt-10'>
        <p className='prata-regular text-3xl'>{currentState}</p>
        <hr className='border-none h-[1.5px] w-8 bg-gray-800' />
      </div> {currentState === 'Login' ? " : <input onChange={e=>setName(e.target.value)} value={name} type='text' className='w-full px-3 py-2 border border-gray-800' placeholder='Name' required/>}
      <input onChange={e=>setEmail(e.target.value)} value={email} type="email" className='w-full px-3 py-2 border border-gray-800' placeholder='Email' required/>
      <input onChange={e=>setPasword(e.target.value)} value={password} type="password" className='w-full px-3 py-2 border border-gray-800' placeholder='Password' required/>
      <div className='w-full flex justify-between text-sm mt-[-8px]'>
        <p className='cursor-pointer'>Forgot your password?</p>
        {
          currentState === 'Login'
          ? <p onClick={()=>setCurrentState('Sign Up')} className='cursor-pointer'>Create account</p>
          : <p onClick={()=>setCurrentState('Login')} className='cursor-pointer'>Login Here</p>
        }
      </div>
    </form>
  )

```

```

    <button className='bg-black text-white font-light px-8 py-2 mt-4'>{currentState === 'Login' ?
'Sign In' : 'Sign Up'}</button>

  </form>

)
}

export default Login

```

PlaceOrders.jsx

```

import React, { useContext, useState } from 'react'

import Title from '../components/Title'

import CartTotal from '../components/CartTotal'

import { assets } from '../assets/assets'

import { ShopContext } from '../context/ShopContext'

import axios from 'axios'

import { toast } from 'react-toastify'

const PlaceOrder = () => {

  const [method, setMethod] = useState('cod');

  const { navigate, backendUrl, token, cartItems, setCartItems, getCartAmount, delivery_fee,
products } = useContext(ShopContext);

  const [formData, setFormData] = useState({

    firstName: "",

    lastName: "",

    email: "",

    street: "",

    city: "",

    state: "",

    zipcode: "",

    country: "",

    phone: ""

  })

  const onChangeHandler = (event) => {

    const name = event.target.name

```

```

const value = event.target.value

setFormData(data => ({ ...data, [name]: value })))
}

const initPay = (order) => {
  const options = {
    key: import.meta.env.VITE_RAZORPAY_KEY_ID,
    amount: order.amount,
    currency: order.currency,
    name: 'Order Payment',
    description: 'Order Payment',
    order_id: order.id,
    receipt: order.receipt,
    handler: async (response) => {
      console.log(response)
      try {
        const { data } = await axios.post(backendUrl +
'/api/order/verifyRazorpay', response, {headers: {token}})
        if (data.success) {
          navigate('/orders')
          setCartItems({})
        }
      } catch (error) {
        console.log(error)
        toast.error(error)
      }
    }
  }

  const rzp = new window.Razorpay(options)
  rzp.open()
}

```

```

const onSubmitHandler = async (event) => {
  event.preventDefault()
  try {
    let orderItems = []
    for (const items in cartItems) {
      for (const item in cartItems[items]) {
        if (cartItems[items][item] > 0) {
          const itemInfo = structuredClone(products.find(product => product._id === items))
          if (itemInfo) {
            itemInfo.size = item
            itemInfo.quantity = cartItems[items][item]
            orderItems.push(itemInfo)
          }
        }
      }
    }
    let orderData = {
      address: formData,
      items: orderItems,
      amount: getCartAmount() + delivery_fee
    }
    switch (method) {
      // API Calls for COD
      case 'cod':
        const response = await axios.post(backendUrl +
'/api/order/place',orderData,{headers:{token}})
        if (response.data.success) {
          setCartItems({})
          navigate('/orders')
        } else {
          toast.error(response.data.message)

```

```

    }
    break;
    case 'stripe':
        const responseStripe = await axios.post(backendUrl +
        '/api/order/stripe',orderData,{headers:{token}})
        if (responseStripe.data.success) {
            const {session_url} = responseStripe.data
            window.location.replace(session_url)
        } else {
            toast.error(responseStripe.data.message)
        }
        break;
    case 'razorpay':
        const responseRazorpay = await axios.post(backendUrl + '/api/order/razorpay',
        orderData, {headers:{token}})
        if (responseRazorpay.data.success) {
            initPay(responseRazorpay.data.order)
        }
        break;
    default:
        break;
}
} catch (error) {
    console.log(error)
    toast.error(error.message)
}
}
return (
    <form onSubmit={onSubmitHandler} className='flex flex-col sm:flex-row justify-between gap-4
    pt-5 sm:pt-14 min-h-[80vh] border-t'>
        { /* ----- Left Side ----- */ }
        <div className='flex flex-col gap-4 w-full sm:max-w-[480px]'>

```

```

<div className='text-xl sm:text-2xl my-3'>

  <Title text1={'DELIVERY'} text2={'INFORMATION'} />

</div>

<div className='flex gap-3'>

  <input required onChange={onChangeHandler} name='firstName'
value={formData.firstName} className='border border-gray-300 rounded py-1.5 px-3.5 w-full'
type="text" placeholder='First name' />

  <input required onChange={onChangeHandler} name='lastName'
value={formData.lastName} className='border border-gray-300 rounded py-1.5 px-3.5 w-full'
type="text" placeholder='Last name' />

</div>

  <input required onChange={onChangeHandler} name='email' value={formData.email}
className='border border-gray-300 rounded py-1.5 px-3.5 w-full' type="email" placeholder='Email
address' />

  <input required onChange={onChangeHandler} name='street' value={formData.street}
className='border border-gray-300 rounded py-1.5 px-3.5 w-full' type="text" placeholder='Street'
/>

  <div className='flex gap-3'>

    <input required onChange={onChangeHandler} name='city' value={formData.city}
className='border border-gray-300 rounded py-1.5 px-3.5 w-full' type="text" placeholder='City' />

    <input onChange={onChangeHandler} name='state' value={formData.state}
className='border border-gray-300 rounded py-1.5 px-3.5 w-full' type="text" placeholder='State' />

  </div>

  <div className='flex gap-3'>

    <input required onChange={onChangeHandler} name='zipcode' value={formData.zipcode}
className='border border-gray-300 rounded py-1.5 px-3.5 w-full' type="number"
placeholder='Zipcode' />

    <input required onChange={onChangeHandler} name='country'
value={formData.country} className='border border-gray-300 rounded py-1.5 px-3.5 w-full'
type="text" placeholder='Country' />

  </div>

  <input required onChange={onChangeHandler} name='phone' value={formData.phone}
className='border border-gray-300 rounded py-1.5 px-3.5 w-full' type="number"
placeholder='Phone' />

</div>

```



```

{ /* ----- Right Side ----- */}

<div className='mt-8'>

  <div className='mt-8 min-w-80'>

    <CartTotal />

  </div>

  <div className='mt-12'>

    <Title text1={'PAYMENT'} text2={'METHOD'} />

    { /* ----- Payment Method Selection ----- */}

    <div className='flex gap-3 flex-col lg:flex-row'>

      { /*<div onClick={() => setMethod('stripe')} className='flex items-center gap-3 border p-
2 px-3 cursor-pointer'>

        <p className={`min-w-3.5 h-3.5 border rounded-full ${method === 'stripe' ? 'bg-
green-400' : ''}`}></p>

        <img className='h-5 mx-4' src={assets.stripe_logo} alt="" />

      </div>*/}

      <div onClick={() => setMethod('razorpay')} className='flex items-center gap-3 border
p-2 px-3 cursor-pointer'>

        <p className={`min-w-3.5 h-3.5 border rounded-full ${method === 'razorpay' ? 'bg-
green-400' : ''}`}></p>

        <img className='h-5 mx-4' src={assets.razorpay_logo} alt="" />

      </div>

      <div onClick={() => setMethod('cod')} className='flex items-center gap-3 border p-2
px-3 cursor-pointer'>

        <p className={`min-w-3.5 h-3.5 border rounded-full ${method === 'cod' ? 'bg-green-
400' : ''}`}></p>

        <p className='text-gray-500 text-sm font-medium mx-4'>CASH ON DELIVERY</p>

      </div>

    </div>

    <div className='w-full text-end mt-8'>

      <button type='submit' className='bg-black text-white px-16 py-3 text-sm'>PLACE
ORDER</button>

    </div>

  </div>

```

```

        </div>
      </form>
    )
  }
}

```

```
export default PlaceOrder
```

Product.jsx

```

import React, { useContext, useEffect, useState } from 'react'
import { useParams } from 'react-router-dom'
import { ShopContext } from '../context/ShopContext';
import { assets } from '../assets/assets';
import RelatedProducts from '../components/RelatedProducts';

const Product = () => {

  const { productId } = useParams();
  const { products, currency ,addToCart } = useContext(ShopContext);
  const [productData, setProductData] = useState(false);
  const [image, setImage] = useState("")
  const [size,setSize] = useState("")

  const fetchProductData = async () => {

    products.map((item) => {
      if (item._id === productId) {
        setProductData(item)
        setImage(item.image[0])
        return null;
      }
    })
  }
}

```

```
}
```

```
useEffect(() => {
```

```
  fetchProductData();
```

```
}, [productId,products])
```

```
return productData ? (
```

```
<div className='border-t-2 pt-10 transition-opacity ease-in duration-500 opacity-100'>
```

```
  { /* ----- Product Data ----- */ }
```

```
<div className='flex gap-12 sm:gap-12 flex-col sm:flex-row'>
```

```
  { /* ----- Product Images ----- */ }
```

```
<div className='flex-1 flex flex-col-reverse gap-3 sm:flex-row'>
```

```
  <div className='flex sm:flex-col overflow-x-auto sm:overflow-y-scroll justify-between  
sm:justify-normal sm:w-[18.7%] w-full'>
```

```
    {
```

```
      productData.image.map((item,index)=>{
```

```
        <img onClick={()=>setImage(item)} src={item} key={index} className='w-[24%] sm:w-full  
sm:mb-3 flex-shrink-0 cursor-pointer' alt="" />
```

```
      })
```

```
    }
```

```
</div>
```

```
<div className='w-full sm:w-[80%]'>
```

```
  <img className='w-full h-auto' src={image} alt="" />
```

```
</div>
```

```
</div>
```

```
  { /* ----- Product Info ----- */ }
```

```
<div className='flex-1'>
```

```
<h1 className='font-medium text-2xl mt-2'>{productData.name}</h1>
```

```
<div className=' flex items-center gap-1 mt-2'>
```

```
  <img src={assets.star_icon} alt="" className="w-3 5" />
```

```

<img src={assets.star_icon} alt="" className="w-3 5" />
<img src={assets.star_icon} alt="" className="w-3 5" />
<img src={assets.star_icon} alt="" className="w-3 5" />
<img src={assets.star_dull_icon} alt="" className="w-3 5" />
<p className='pl-2'>(122)</p>
</div>

<p className='mt-5 text-3xl font-medium'>{currency}{productData.price}</p>
<p className='mt-5 text-gray-500 md:w-4/5'>{productData.description}</p>
<div className='flex flex-col gap-4 my-8'>
  <p>Select Size</p>
  <div className='flex gap-2'>
    {productData.sizes.map((item,index)=>(
      <button onClick={()=>setSize(item)} className={`border py-2 px-4 bg-gray-100 ${item ===
size ? 'border-orange-500' : ''}`} key={index}>{item}</button>
    ))}
  </div>
</div>

<button onClick={()=>addToCart(productData._id,size)} className='bg-black text-white px-8
py-3 text-sm active:bg-gray-700'>ADD TO CART</button>

<hr className='mt-8 sm:w-4/5' />
<div className='text-sm text-gray-500 mt-5 flex flex-col gap-1'>
  <p>100% Original product.</p>
  <p>Cash on delivery is available on this product.</p>
  <p>Easy return and exchange policy within 7 days.</p>
</div>
</div>
</div>

{/* ----- Description & Review Section ----- */}
<div className='mt-20'>
  <div className='flex'>
    <b className='border px-5 py-3 text-sm'>Description</b>

```

```

    <p className='border px-5 py-3 text-sm'>Reviews (122)</p>
  </div>

  <div className='flex flex-col gap-4 border px-6 py-6 text-sm text-gray-500'>
    <p>An e-commerce website is an online platform that facilitates the buying and selling of
    products or services over the internet. It serves as a virtual marketplace where businesses and
    individuals can showcase their products, interact with customers, and conduct transactions without
    the need for a physical presence. E-commerce websites have gained immense popularity due to
    their convenience, accessibility, and the global reach they offer.</p>

    <p>E-commerce websites typically display products or services along with detailed
    descriptions, images, prices, and any available variations (e.g., sizes, colors). Each product usually
    has its own dedicated page with relevant information.</p>
  </div>
</div>

  { /* ----- display related products ----- */ }

  <RelatedProducts category={productData.category} subCategory={productData.subCategory} />

</div>
) : <div className=' opacity-0'></div>
}

export default Product

```

Verify.jsx

```
import React from 'react'

import { useContext } from 'react'

import { ShopContext } from '../context/ShopContext'

import { useSearchParams } from 'react-router-dom'

import { useEffect } from 'react'

import { toast } from 'react-toastify'

import axios from 'axios'

const Verify = () => {

  const { navigate, token, setCartItems, backendUrl } = useContext(ShopContext)

  const [searchParams, setSearchParams] = useSearchParams()

  const success = searchParams.get('success')

  const orderId = searchParams.get('orderId')

  const verifyPayment = async () => {

    try {

      if (!token) {

        return null

      }

      const response = await axios.post(backendUrl + '/api/order/verifyStripe', { success, orderId }, {

        headers: { token } })

      if (response.data.success) {

        setCartItems({})

        navigate('/orders')

      } else {
```

```
        navigate('/cart')
      }

    } catch (error) {
      console.log(error)
      toast.error(error.message)
    }
  }

  useEffect(() => {
    verifyPayment()
  }, [token])

  return (
    <div>

    </div>
  )
}

export default Verify
```

CHAPTER 6

MODELS OF E-COMMERCE WEBSITE

6.1 Models

In the context of an e-commerce shopping website, **models** represent the core data structures used to manage and store the information required by the system. These models are typically implemented in the backend using a **NoSQL database** like MongoDB in the MERN stack. Each model corresponds to a collection in the database and defines how data is structured, stored, and related to other pieces of data. Below are the primary models used in an e-commerce shopping website.

1. User Model

The **User model** represents a customer or an admin and stores their personal details, credentials, and other relevant data.

Fields:

- name: String (Customer's name)
- email: String (Unique, used for authentication)
- password: String (Hashed password for secure login)
- role: String (User role, e.g., "customer" or "admin")
- address: Array of objects (Shipping addresses with fields like street, city, zip code, etc.)
- orderHistory: Array of Order IDs (Reference to the user's previous orders)
- wishlist: Array of Product IDs (List of products the user has saved for future purchase)
- createdAt: Date (User registration date)

2. Product Model

The **Product model** defines the data structure for the products available for purchase on the website. Each product has information like name, price, description, category, and more.

Fields:

- name: String (Name of the product)
- description: String (Description of the product)
- price: Number (Price of the product)
- category: String (Category of the product like electronics, clothing, etc.)
- image: String (URL of the product image)
- stockQuantity: Number (Available quantity in stock)
- ratings: Number (Average rating of the product)
- createdAt: Date (Date the product was added)

3. Order Model

The **Order model** tracks individual orders placed by customers, including product details, shipping address, order status, and payment information.

Fields:

- `userId`: ObjectId (Reference to the User who placed the order)
- `products`: Array of objects (Each product includes Product ID, quantity, and price)
- `totalPrice`: Number (Total price of the order)
- `shippingAddress`: Object (Shipping details like street, city, and postal code)
- `paymentStatus`: String (Payment status, e.g., "Paid", "Pending")
- `orderStatus`: String (Order status, e.g., "Processing", "Shipped", "Delivered")
- `createdAt`: Date (Order creation date)

4. Cart Model

The **Cart model** holds the items that a user has added to their cart but not yet purchased. It stores the user's cart details, including product quantities and prices.

Fields:

- `userId`: ObjectId (Reference to the User who owns the cart)
- `products`: Array of objects (Each product includes Product ID, quantity, and price)
- `totalPrice`: Number (Total price of items in the cart)
- `createdAt`: Date (Date the cart was created/last updated)

5. Payment Model

The **Payment model** stores payment details for each order and tracks payment status, amount, and the payment gateway used (e.g., PayPal, Stripe).

Fields:

- `orderId`: ObjectId (Reference to the Order associated with the payment)
- `paymentGateway`: String (Payment method like "PayPal", "Stripe")
- `paymentStatus`: String (e.g., "Completed", "Failed", "Pending")
- `amount`: Number (Total amount paid)
- `paymentDate`: Date (Date when the payment was made)

6. Admin Model

While the **User model** can also store admin data (via the role field), some e-commerce systems may use a separate **Admin model** to manage admin-specific features, such as product approvals, site analytics, and reporting.

Fields:

- `name`: String (Admin's name)
- `email`: String (Unique identifier for login)

- password: String (Hashed password)
- role: String (Should always be "admin" for admins)
- createdAt: Date (Date of admin account creation)

6.2 Company Model

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

```
const companySchema = new mongoose.Schema({
  companyName: { type: String, required: true },
  companyDescription: { type: String, required: true },
  contactEmail: { type: String, required: true },
  contactPhone: { type: String, required: true },
  address: {
    street: { type: String, required: true },
    city: { type: String, required: true },
    state: { type: String, required: true },
    postalCode: { type: String, required: true },
    country: { type: String, required: true },
  },
  logo: { type: String, required: true }, // URL of the company's logo
  websiteURL: { type: String, required: true }, // Website URL
  socialMediaLinks: {
    facebook: { type: String },
    twitter: { type: String },
    instagram: { type: String },
    linkedin: { type: String },
  },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now },
  returnPolicy: { type: String, required: true }, // Return policy text
  shippingPolicy: { type: String, required: true }, // Shipping policy text
  paymentMethods: [{ type: String, required: true }], // ["Credit Card", "PayPal", etc.]
  taxDetails: {
    taxRate: { type: Number, required: true }, // Tax rate applied to orders
```

```
    taxRegion: { type: String, required: true }, // Region for which tax applies
  },
});
```

```
module.exports = mongoose.model('Company', companySchema);
```

6.3 Shopping Model

1. Product Model

The **Product Model** defines the data structure for products available in the online store.

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

```
const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  description: { type: String, required: true },
  price: { type: Number, required: true },
  category: { type: String, required: true },
  image: { type: String, required: true },
  stockQuantity: { type: Number, required: true },
  ratings: { type: Number, default: 0 },
  createdAt: { type: Date, default: Date.now }
});
```

```
module.exports = mongoose.model('Product', productSchema);
```

2. Cart Model

The **Cart Model** tracks the products added to the user's cart before they proceed to checkout. It holds the products, quantities, and total value of the items.

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

const cartSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  products: [{
    productId: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },
    quantity: { type: Number, required: true },
    price: { type: Number, required: true }
  }],
  totalPrice: { type: Number, required: true },
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Cart', cartSchema);
```

3. Order Model

Once the customer is ready to make a purchase, an order is created. The **Order Model** stores information about the completed order, including the products, shipping details, and payment status.

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

const orderSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  products: [{
```

```

    productId: { type: mongoose.Schema.Types.ObjectId, ref: 'Product', required: true },
    quantity: { type: Number, required: true },
    price: { type: Number, required: true }
  }],
  totalPrice: { type: Number, required: true },
  shippingAddress: {
    street: { type: String, required: true },
    city: { type: String, required: true },
    zip: { type: String, required: true },
    country: { type: String, required: true }
  },
  paymentStatus: { type: String, required: true },
  orderStatus: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});

```

```

module.exports = mongoose.model('Order', orderSchema);

```

4. Payment Model

The **Payment Model** stores information about the payment for an order, including the payment method, status, and amount paid.

```

const mongoose = require('mongoose');

```

```

const paymentSchema = new mongoose.Schema({
  orderId: { type: mongoose.Schema.Types.ObjectId, ref: 'Order', required: true },
  paymentMethod: { type: String, required: true },

```

```
paymentStatus: { type: String, required: true },  
  
amount: { type: Number, required: true },  
  
paymentDate: { type: Date, default: Date.now }  
  
});  
  
module.exports = mongoose.model('Payment', paymentSchema);
```

6.4 User Model

The **User Model** in an e-commerce shopping website represents the customers (or users) who interact with the platform, create accounts, browse products, make purchases, and track their orders. This model typically holds essential user data such as personal information, authentication credentials, address details, and other preferences related to their shopping behavior.

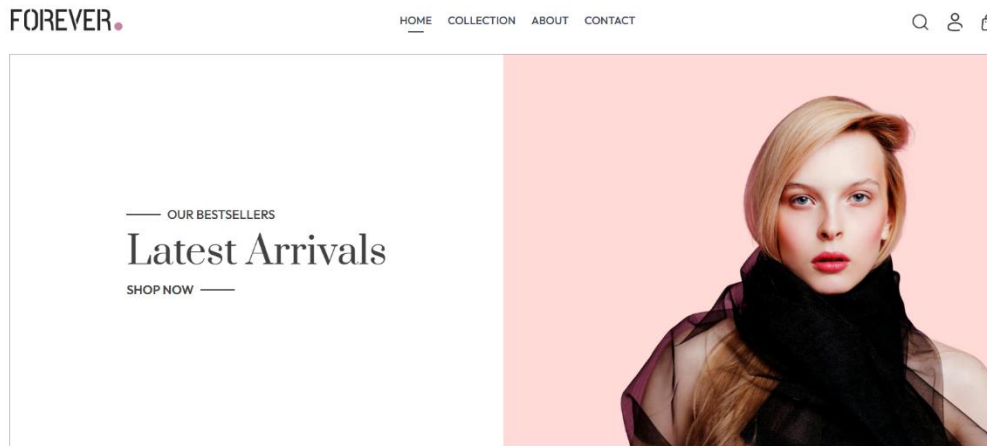
Key Components of the User Model

1. **Personal Information:** Stores basic details like name, email, and phone number.
2. **Authentication Information:** Includes password, email verification status, and roles (if needed).
3. **Address Information:** Stores shipping and billing addresses for order delivery.
4. **Order History:** Tracks past orders made by the user.
5. **Wishlist or Cart:** Stores products the user has saved for later or is currently purchasing.
6. **Payment Information:** Can include payment preferences or transaction history, though sensitive details like credit card numbers are usually stored using a third-party payment service for security.
7. **Preferences and Settings:** Stores the user's notification preferences, language settings, or saved payment methods.

CHAPTER 7

USER INTERFACE AND SNAPSHOTS

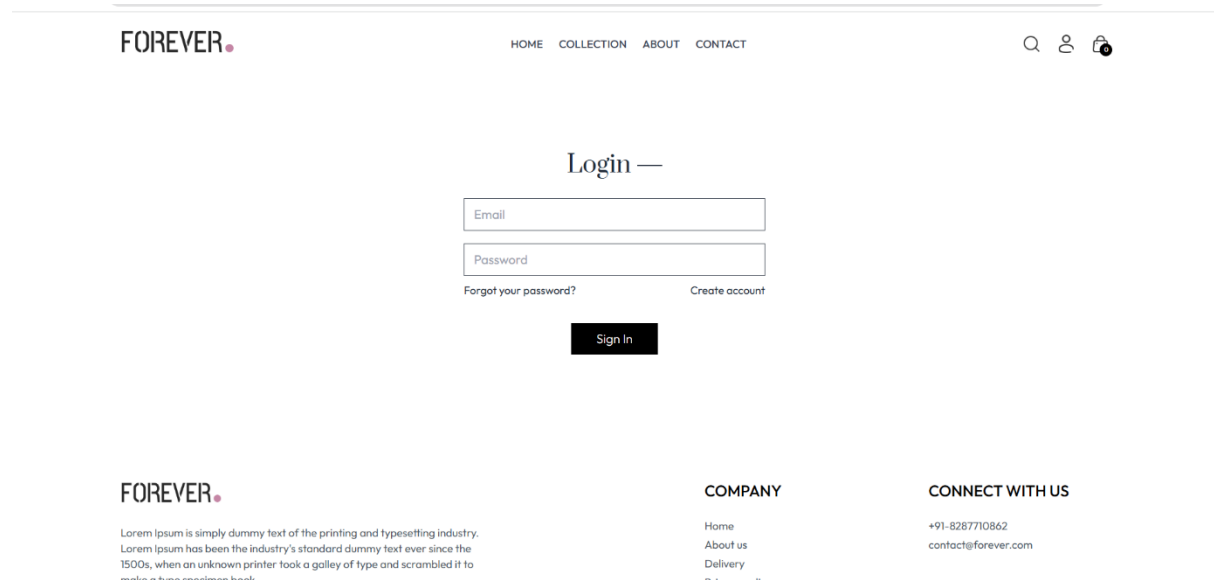
Home Page



LATEST COLLECTIONS —

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the.

Login Page



Signup Page

FOREVER.

HOME COLLECTION ABOUT CONTACT

Q P

Sign Up —

Name

Email

Password

Forgot your password?

Login Here

Sign Up

FOREVER.

COMPANY

CONNECT WITH US

Lorem Ipsum is simply dummy text of the printing and typesetting industry.

Home

+91-8287710862

Collection Page

FOREVER.

HOME COLLECTION ABOUT CONTACT

Q P

FILTERS

ALL COLLECTIONS —

Sort by: Low to High

CATEGORIES

☐ Men

☐ Women

☐ kids

TYPE

☐ Topwear

☐ Bottomwear

☐ Winterwear

Crop Top

T150

Straight fit jeans

T150

Kid's Tshirt

T250

kids blue tee

T500

FOREVER.

COMPANY

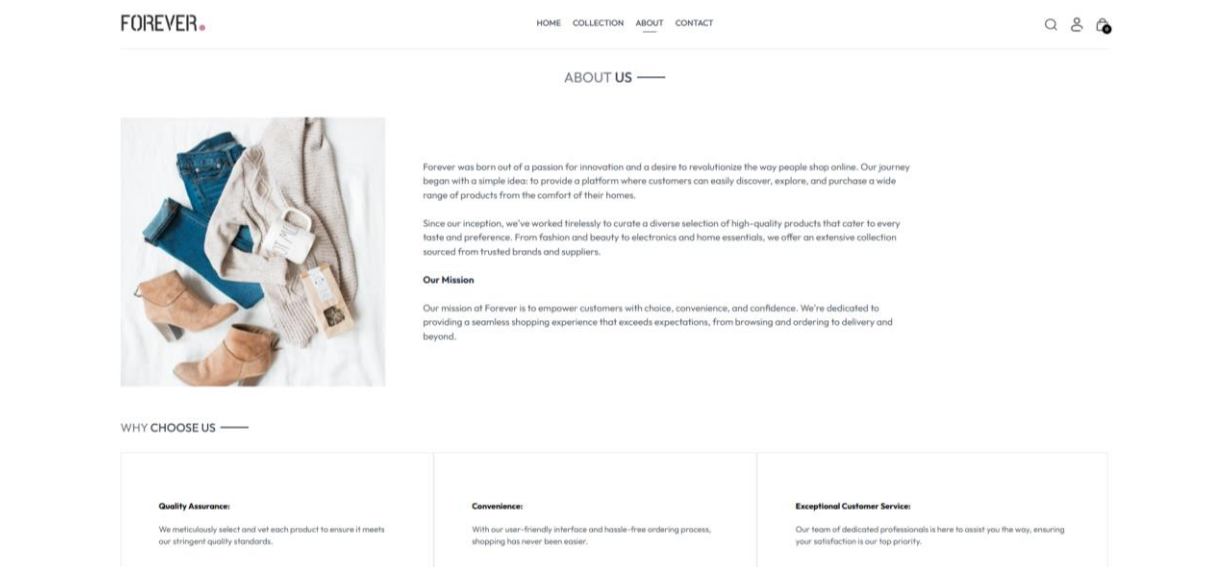
CONNECT WITH US

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

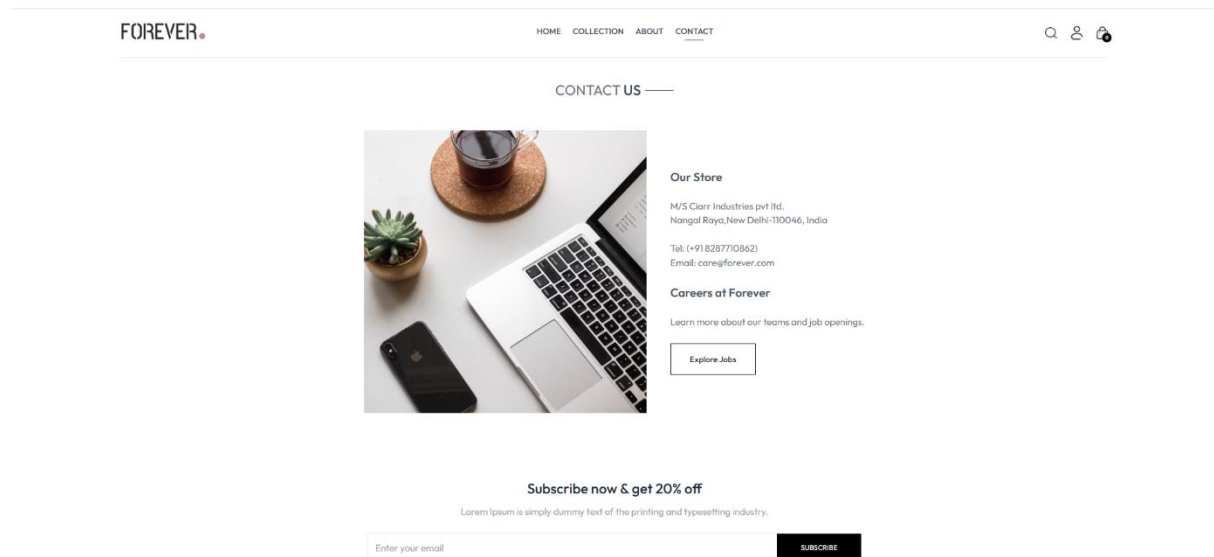
Home
About us
Delivery
Privacy policy

+91-8287710862
contact@forever.com

About Page



Contact Page





Order Page

FOREVER.

HOME COLLECTION ABOUT CONTACT

Q U C





Crop Top

★★★★★ (122)

₹150

100% cotton knitted lightweight crop top for women's

Select Size

S M **L** XL XXL

ADD TO CART

100% Original product.
Cash on delivery is available on this product.
Easy return and exchange policy within 7 days.


Cart Page

FOREVER.

HOME COLLECTION ABOUT CONTACT

Q U C

YOUR CART




Crop Top

₹150

L

1



CART TOTALS

Subtotal

₹ 150.00

Shipping Fee

₹ 40.00

Total

₹ 190.00

PROCEED TO CHECKOUT

FOREVER.

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a

COMPANY

Home
About us

CONNECT WITH US

+91-8287710862
contact@forever.com

Checkout Page

FOREVER.

HOME COLLECTION ABOUT CONTACT

Q U C

DELIVERY INFORMATION

Sahil

Manav

sahilmanav2003@gmail.com

Nangal raya

new delhi

delhi

110046

India

8287710862

CART TOTALS

Subtotal

₹ 150.00


Shipping Fee

₹ 40.00

Total

₹ 190.00

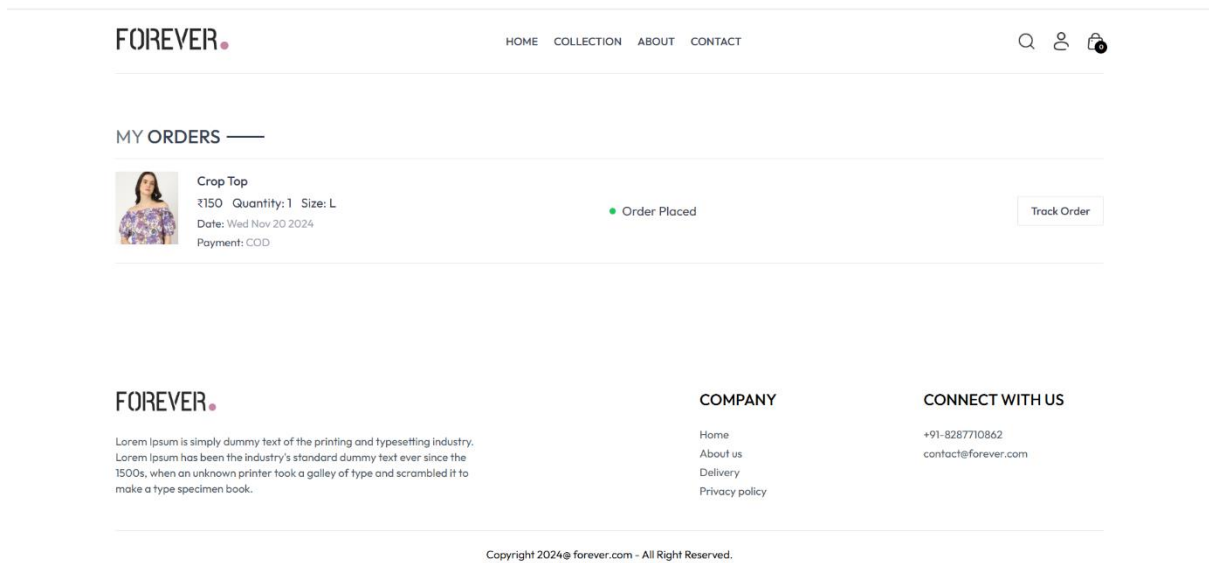
PAYMENT METHOD

☐ 

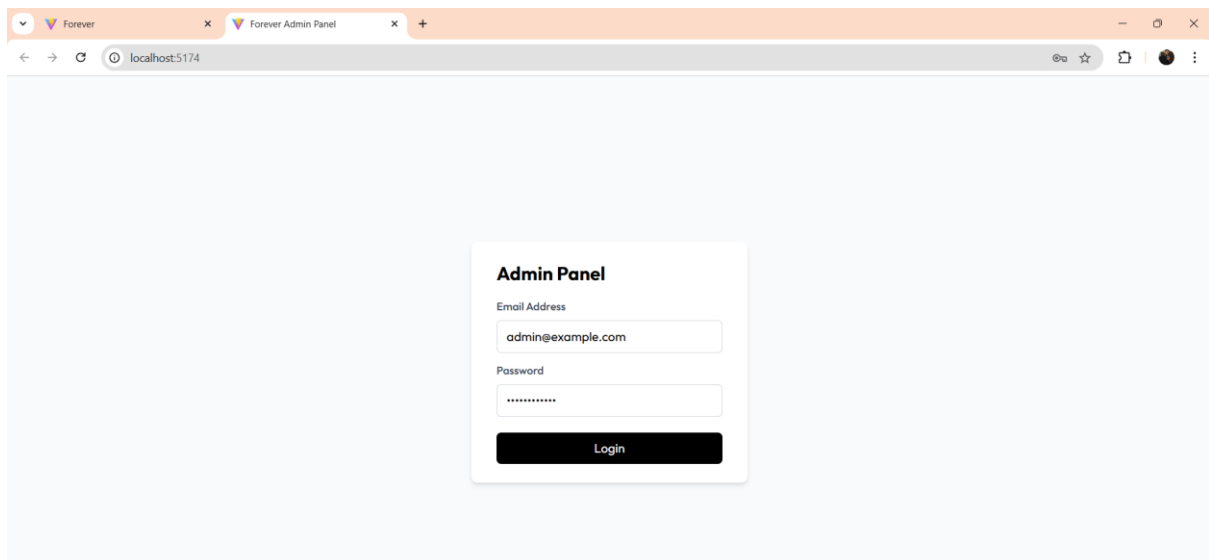
☒ CASH ON DELIVERY

PLACE ORDER

Placed Order Page



Admin Login Page



Adding Items Page

FOREVER ADMIN PANEL

Logout

+ Add Items

List Items

Orders

Upload Image

Upload Upload Upload

Product name

Men's Casual Full Sleeve -Tshirt

Product description

100% Cotton

Product category Sub category Product Price

Men Topwear 525

Product Sizes

S M L XL XXL

☒ Add to bestseller

ADD

All Products Page

FOREVER ADMIN PANEL

Logout

+ Add Items

List Items

Orders

All Products List

Image	Name	Category	Price	Action
	Men's Casual Full Sleeve -Tshirt	Men	₹525	X
	kids bluee tee	Kids	₹500	X
	Kid's Tshirt	Kids	₹250	X
	Crop Top	Women	₹150	X
	Straight fit jeans	Men	₹150	X

All Orders Page

The screenshot shows the Forever Admin Panel interface. On the left is a sidebar with navigation options: Add Items, List Items, and Orders (highlighted). The main content area is titled 'Order Page' and lists two orders:

Item	Customer	Items	Price	Status
Crop Top x 1 L	Sahil Manav Nangal raya, new delhi, delhi, India, 110046 8287710862	1	₹190	Order Placed
kids bluee tee x 1 M	Jitin Nailwal c/130, NEW DELHI, Delhi, India, 110046 08287710862	1	₹540	Packing

MongoDB

The screenshot shows the MongoDB Atlas interface for a project named 'Sahil's Org'. The left sidebar contains navigation options: Overview, DATABASE, Clusters, SERVICES, and SECURITY. The main content area displays the 'e-commerce' database structure:

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
orders	20	10.01KB	923B	36KB	1	36KB	36KB
products	5	1.66KB	338B	36KB	1	36KB	36KB
users	9	4.08KB	464B	36KB	2	72KB	36KB

Atlas Sahil's Org - ... Access Manager Billing All Clusters Get Help Sahil

Project 0 Data Services Charts

Overview

DATABASE

Clusters

SERVICES

Atlas Search

Stream Processing

Triggers

Migration

Data Federation

SECURITY

Quickstart

Backup

Database Access

Network Access

Advanced

Goto

DATABASES: 2 COLLECTIONS: 9

+ Create Database

Search Namespaces

e-commerce

orders

products

users

sample_mflix

e-commerce.orders

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 18.01KB TOTAL DOCUMENTS: 20 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter Type a query: { field: 'value' } Reset Apply Options

QUERY RESULTS: 1-20 OF 20

```

{
  "_id": ObjectId("66f4769282a2298d60fde87e"),
  "userId": "66f475f282a2298d60fde873",
  "items": Array (1)
    amount: 145
  "address": Object
    status: "Out for delivery"
    paymentMethod: "COD"
    payment: false
    date: 1727297178996
    __v: 0

```

Atlas Sahil's Org - ... Access Manager Billing All Clusters Get Help Sahil

Project 0 Data Services Charts

Overview

DATABASE

Clusters

SERVICES

Atlas Search

Stream Processing

Triggers

Migration

Data Federation

SECURITY

Quickstart

Backup

Database Access

Network Access

Advanced

Goto

DATABASES: 2 COLLECTIONS: 9

+ Create Database

Search Namespaces

e-commerce

orders

products

users

sample_mflix

e-commerce.products

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.68KB TOTAL DOCUMENTS: 6 INDEXES TOTAL SIZE: 36KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter Type a query: { field: 'value' } Reset Apply Options

QUERY RESULTS: 1-5 OF 5

```

{
  "_id": ObjectId("67345e3bfd92631b17d41570"),
  "name": "Straight fit jeans",
  "description": "Men's bottomwear",
  "price": 150,
  "image": Array (1)
    category: "Men"
    subCategory: "Bottomwear"
  "sizes": Array (5)
    bestseller: false
    date: 1731485243379
    __v: 0

```

Atlas Sahil's Org - ... Access Manager Billing All Clusters Get Help Sahil

Project 0 Data Services Charts

Overview

DATABASE

Clusters

SERVICES

Atlas Search

Stream Processing

Triggers

Migration

Data Federation

SECURITY

Quickstart

Backup

Database Access

Network Access

Advanced

Goto

DATABASES: 2 COLLECTIONS: 9

+ Create Database

Search Namespaces

e-commerce

orders

products

users

sample_mflix

e-commerce.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 4.08KB TOTAL DOCUMENTS: 9 INDEXES TOTAL SIZE: 72KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter Type a query: { field: 'value' } Reset Apply Options

QUERY RESULTS: 1-9 OF 9

```

{
  "_id": ObjectId("66f3d958e191a8149fb9cd63"),
  "name": "Sahil",
  "email": "sahilmanav8@gmail.com",
  "password": "123456789",
  "cartData": Object
    date: 2024-09-25T09:16:01.236+00:00
    __v: 0
}

{
  "_id": ObjectId("66f475f282a2298d60fde873"),
  "name": "Sahil Manav",
  "email": "sahilmanav23@gmail.com",
  "password": "52b5105f4ogEERyzUAR99sa73shae0JVxIQEEd91JuYmI/Mn6V39Yk96JS",
  "cartData": Object

```

CHAPTER 8

FUTURE SCOPE

The future scope of an e-commerce shopping website is vast and ever-evolving as technology, consumer preferences, and business models continue to grow and adapt. With advancements in artificial intelligence (AI), augmented reality (AR), blockchain, and machine learning, the e-commerce industry is poised for significant transformations. Below are some of the key areas where e-commerce shopping websites can expand and innovate in the future:

1. AI-Driven Personalization

- **Future Scope:** E-commerce websites are increasingly leveraging AI to provide highly personalized shopping experiences. This includes personalized product recommendations, dynamic pricing, and tailored content based on a user's browsing history, preferences, and purchasing behavior. AI can help in predicting what customers might be interested in and provide customized promotions or discounts.
- **Impact:** This will lead to better customer engagement, higher conversion rates, and an overall enhanced user experience.

2. Augmented Reality (AR) and Virtual Reality (VR) Shopping

- **Future Scope:** As AR and VR technologies advance, e-commerce websites can incorporate virtual try-ons for products like clothes, accessories, or makeup. Customers could use their smartphones or AR glasses to visualize how a product would look in their home or on themselves before making a purchase.
- **Impact:** This will create immersive shopping experiences and reduce uncertainty, improving customer satisfaction and decreasing return rates.

3. Voice Commerce

- **Future Scope:** Voice-assisted shopping using AI-powered devices like Amazon Alexa, Google Assistant, or Apple's Siri is on the rise. E-commerce websites can integrate voice commands into their platforms, allowing users to search for products, add items to their cart, and place orders using only their voice.
- **Impact:** Voice commerce will improve convenience and accessibility, making online shopping easier, especially for users with disabilities or those looking for hands-free experiences.

4. Chatbots and Virtual Assistants

- **Future Scope:** AI-driven chatbots and virtual assistants are already helping customers with product inquiries, order status, and personalized shopping experiences. In the future, these bots will become more sophisticated, using natural language processing (NLP) to understand and respond to customer queries in a more human-like manner.
- **Impact:** This will result in improved customer support, faster response times, and a more efficient shopping experience, allowing users to get assistance 24/7.

5. Blockchain for Transparency and Security

- **Future Scope:** Blockchain technology can revolutionize e-commerce by enhancing security, transparency, and reducing fraud. It can be used for secure payments, tracking product authenticity, and ensuring transparent supply chains.
- **Impact:** Blockchain can foster trust with customers by verifying product origins, preventing counterfeit products, and ensuring the safety of transactions.

6. Sustainability and Ethical Shopping

- **Future Scope:** As consumers become more conscious about sustainability, e-commerce platforms can integrate features that help users make eco-friendly purchases. This could include highlighting sustainable brands, offering carbon-neutral delivery options, or providing information about a product's environmental impact.
- **Impact:** E-commerce websites that promote sustainability can appeal to the growing eco-conscious consumer base and differentiate themselves in a competitive market.

7. Subscription-Based Models

- **Future Scope:** Subscription services are growing in popularity, and e-commerce businesses can expand their offerings to include subscription-based models for products like beauty items, groceries, or fashion. These models can provide customers with personalized, recurring deliveries, making shopping more convenient.
- **Impact:** Subscription models encourage customer retention and help businesses predict revenue streams while fostering long-term customer loyalty.

8. Social Commerce

- **Future Scope:** Social media platforms like Instagram, Facebook, and TikTok are increasingly becoming shopping destinations. E-commerce websites can integrate directly with these platforms to allow users to make purchases without leaving the app.
- **Impact:** This trend will bridge the gap between social media and e-commerce, enabling impulse buying and creating new opportunities for marketing, customer engagement, and sales generation.

9. AI-Powered Logistics and Supply Chain Optimization

- **Future Scope:** AI and machine learning can help optimize supply chains, from inventory management to warehousing and delivery. Predictive analytics can improve inventory forecasting, automate restocking, and help businesses manage product shortages and surpluses more effectively.
- **Impact:** Improved logistics lead to faster delivery times, reduced operational costs, and better management of product availability, which can significantly enhance the customer experience.

10. Integration of Multiple Payment Methods

- **Future Scope:** The future of e-commerce will likely see the integration of even more diverse payment options, including cryptocurrencies, digital wallets, and biometric authentication (like fingerprint or facial recognition payments).
- **Impact:** This can provide more flexibility and security to users, enhancing user trust and making payments more seamless across different platforms and currencies.

CHAPTER 9

FEASIBILITY STUDY

1. Technical Feasibility:

- **Technology Stack:** The use of the ****MERN stack**** (MongoDB, Express.js, React.js, Node.js) ensures flexibility, scalability, and high performance, suitable for handling high traffic volumes. MongoDB offers fast data retrieval and easy scaling, while React.js enables responsive user interfaces. Node.js supports fast, asynchronous server-side operations.
- **Cloud Hosting:** Cloud platforms like AWS, Google Cloud, or Microsoft Azure offer scalable solutions that can grow with the business. They support high availability, security, and fast data processing, which is crucial for e-commerce websites handling customer orders and payment transactions.
- **Payment Gateway Integration:** Implementing payment systems like Stripe, PayPal, or Razorpay ensures secure, hassle-free transactions and broadens the customer base by supporting multiple payment methods.
- **Security:** The website will require SSL/TLS encryption for secure data transmission and PCI-DSS compliance to securely handle credit card and transaction data.

2. Economic Feasibility:

- **Initial Investment:** The costs of setting up the website involve web development, hosting, domain registration, third-party service integrations (payment gateways, shipping APIs), and marketing expenses. Initial costs can range from a few thousand dollars to tens of thousands, depending on the features and scale of the website.
- **Revenue Streams:** Revenue can be generated through:
 - **Direct Sales:** Selling products (physical goods or digital products) through the platform.
 - **Advertising:** Display ads or allow third-party companies to advertise on the platform once it gains significant traffic.
 - **Affiliate Marketing:** Collaborating with other retailers or brands and earning commissions for promoting their products.

- **Subscription Models:** Offering membership benefits like discounts, free shipping, or exclusive deals.

- **Profitability:** The platform can become profitable within 12–18 months if initial marketing campaigns and product offerings are strategically planned to attract users and encourage frequent purchases.

3. Legal Feasibility:

- **Data Protection:** Adherence to data privacy laws such as GDPR (General Data Protection Regulation) in Europe, CCPA(California Consumer Privacy Act), or local privacy regulations to protect customer data. Ensuring proper encryption, consent forms, and user rights to manage their data is essential.

- **Payment Security:** Ensuring compliance with PCI-DSS for secure payment transactions, safeguarding customers' financial details and mitigating fraud risks.

- **Intellectual Property:** Ensure that all product images, descriptions, logos, and content on the website are either original or licensed to avoid copyright infringement.

- **Terms and Conditions:** Clear Terms of Service, Privacy Policy, and Return/Refund Policies should be outlined to protect the business from legal claims and ensure transparency with customers.

4. Operational Feasibility:

- **Inventory Management:** Efficient inventory tracking systems should be implemented to manage product stock levels, orders, and returns. Solutions like Shopify, WooCommerce, or custom-built inventory systems can assist with this.

- **Shipping and Logistics:** Reliable logistics and shipping partners are essential to deliver products in a timely manner. Consider using services like FedEx, UPS, or local delivery services for regional fulfillment.

- **Customer Service:** A well-structured customer support system, including live chat, help desks, or AI-driven chatbots, can ensure timely responses to customer inquiries, boosting satisfaction.

- **Marketing:** Ongoing digital marketing efforts such as SEO (Search Engine Optimization), SEM (Search Engine Marketing), email campaigns, and social media advertising will be crucial for attracting customers and driving traffic.

5. Scheduling Feasibility:

- Development Timeline: The project can be completed in phases, with the initial development and beta launch taking around 3 to 6 months. A typical development schedule might look like:

- Phase 1 (1-2 months): Requirement gathering, wireframing, and designing the website layout.

- Phase 2 (2-3 months): Front-end and back-end development, integrating payment systems, user registration, and inventory management.

- Phase 3 (1 month): Testing, debugging, and final deployment.

- Post-launch Activities: After launch, there will be continuous work needed for maintenance, updates, feature enhancements, and addressing customer feedback.

- Scalability: The website should be developed with scalability in mind, allowing new features and functionalities to be added as the business grows.

6. Market Feasibility:

- Target Audience: Identifying the target market is crucial for success. Whether focusing on niche products or offering a wide range of consumer goods, understanding the audience's preferences, purchasing behavior, and demographics will help tailor marketing efforts and product offerings.

- Competitive Analysis: Researching competitors and analyzing their offerings, pricing strategies, and customer experiences will help position the website effectively in the market.

- Market Trends: Leveraging trends such as mobile commerce, social commerce, and voice search can give the platform a competitive edge and meet changing customer expectations.

Conclusion:

The feasibility study for an e-commerce shopping website indicates that the project is viable from a technical, economic, legal, operational, and scheduling standpoint. With the right planning, a focus on security and compliance, effective marketing, and a strong customer experience, the website has the potential to thrive in the competitive e-commerce space.

CHAPTER 10

LIMITATIONS

While the **MERN stack** (MongoDB, Express.js, React.js, Node.js) is a powerful and popular choice for developing modern web applications, including e-commerce websites, there are some limitations and challenges associated with using this stack. Below are key limitations of building an e-commerce shopping website using the MERN stack:

1. Scalability Challenges with MongoDB

- **Limited Transaction Support:** Although MongoDB is a flexible NoSQL database, it has limitations in handling complex transactions across multiple collections. This can be a drawback for e-commerce websites that require strong consistency, especially when handling inventory management, payment processing, and multi-step transactions.
- **Data Redundancy:** As a NoSQL database, MongoDB does not enforce strict relational structures, which can lead to data redundancy. For instance, customer data or product information may be repeated across collections, potentially increasing storage and complicating data management.

2. Performance Overhead

- **React.js Rendering:** While React is known for its fast rendering, e-commerce websites with large catalogs and complex UIs might experience performance bottlenecks. Handling large amounts of dynamic data or real-time updates (e.g., inventory status or pricing) in React can cause performance issues without proper optimization techniques (such as lazy loading, code splitting, etc.).
- **Server-Side Rendering (SSR):** React often requires additional configuration and tools (like Next.js) for optimal SEO performance and faster initial loading through server-side rendering. Without SSR, search engines may have difficulty indexing product pages properly, which could impact visibility.

3. Security Concerns

- **Authentication and Authorization:** Although the MERN stack can implement user authentication and authorization, it may require additional development time to securely manage roles, permissions, and sessions. Implementing strong security measures like **JWT tokens**, **OAuth**, and **password hashing** is necessary to protect user data, but improper implementation can leave vulnerabilities.
- **Vulnerabilities in Node.js:** Node.js, while powerful, can be prone to certain security vulnerabilities, such as **Denial of Service (DoS)** attacks if not properly handled. For e-commerce platforms, security is paramount, and developers must continuously stay updated with the latest security patches and practices to protect customer data and payment information.

4. Limited Built-In E-Commerce Features

- **E-Commerce Features Are Not Built-In:** MERN stack itself does not provide any out-of-the-box e-commerce functionalities such as product catalogs, shopping carts, or order management. These features must be built manually, which can be time-consuming. As a result, developers must rely on third-party libraries or custom code, which may introduce bugs or inefficiencies.
- **No Ready-Made Solutions:** Unlike platforms like **Shopify** or **WooCommerce**, which come with a built-in CMS and e-commerce features, MERN requires developers to build everything from scratch, which could increase development time and cost.

5. Real-Time Features Limitations

- **Real-Time Communication:** While Node.js is great for handling real-time interactions, integrating real-time features like live customer support chat, notifications, or live inventory updates requires additional tools and configurations (e.g., **Socket.io**). Ensuring these features are reliable at scale can be difficult, especially during traffic spikes.

6. SEO Challenges

- **Single-Page Application (SPA) Nature:** Since React is often used to build **Single-Page Applications (SPAs)**, it can lead to SEO challenges. By default, search engines may not index JavaScript-heavy pages correctly. For an e-commerce website, where SEO is crucial for organic search traffic, this can be a significant drawback unless server-side rendering (SSR) or static site generation (SSG) is implemented, which adds extra complexity.

7. Complexity of State Management

- **State Management in React:** Managing complex states in large-scale e-commerce applications with React can become cumbersome without proper state management solutions (like Redux or Context API). E-commerce websites tend to have many dynamic features (shopping cart, user authentication, order tracking, etc.), and handling them in a clean, scalable way can be complex.

8. Mobile Responsiveness and Optimization

- **Mobile Optimization:** React-based applications can be slow to load or become inefficient on mobile devices if not optimized properly. E-commerce websites rely on a seamless mobile experience, and failures in optimization could lead to poor user experience and high bounce rates.
- **Cross-Device Consistency:** Ensuring consistent functionality across multiple devices (desktop, tablet, mobile) is challenging. While React offers great flexibility, achieving uniform design and behavior across all platforms can be time-consuming and require additional testing and adjustments.

9. Integration with Legacy Systems

- **Compatibility with Existing Systems:** If the e-commerce business has existing infrastructure or legacy systems (e.g., inventory management or ERP systems), integrating them with the MERN stack may be complex. The lack of a direct interface between the front-end (React) and these legacy systems might require additional middleware or APIs.

10. Hosting and Server Costs

- **Server Resource Utilization:** Running an e-commerce website on Node.js can lead to high server resource consumption, especially if the website experiences high traffic. As demand increases, scaling the application across multiple servers or regions might be required, which can increase hosting costs.
- **Cost of Third-Party Services:** For certain features like payment gateways, email services, or shipping integrations, developers may have to rely on third-party services. These can add additional costs and dependencies to the project, impacting the overall budget.

CHAPTER 11

CONCLUSION

In conclusion, the **MERN stack** (MongoDB, Express.js, React.js, Node.js) offers a compelling and flexible technology stack for developing an **e-commerce shopping website**, enabling developers to create highly dynamic, responsive, and scalable web applications. The **MERN stack** allows for efficient full-stack development using JavaScript across both the server-side and client-side, leading to smoother development cycles, easier code maintenance, and better integration between various layers of the application.

Key Strengths of the MERN Stack for E-Commerce Development:

1. **Unified Technology Stack:** The biggest advantage of using the MERN stack is that it is entirely based on JavaScript, which means that developers can work with the same language from front-end to back-end. This unification reduces the need for learning multiple programming languages and frameworks, streamlining the development process. Additionally, **React.js** simplifies front-end development by allowing developers to build reusable components, which makes creating dynamic, interactive user interfaces more efficient. For the back-end, **Node.js** offers fast, scalable solutions for handling numerous simultaneous requests, which is ideal for handling the high traffic demands often associated with e-commerce platforms.
2. **Real-Time Capabilities:** E-commerce websites often require real-time updates (such as live product inventory changes, order tracking, or customer support chats). With **Node.js**, the MERN stack enables real-time capabilities with features like **web sockets** (using tools such as **Socket.io**) to deliver updates instantly to the users without having to refresh the page. This creates a more interactive and user-friendly experience.
3. **Scalability:** The MERN stack is highly scalable, which is vital for any e-commerce platform expected to grow over time. MongoDB, as a NoSQL database, allows for easy horizontal scaling, meaning it can handle large amounts of data, such as product catalogs, customer details, and order histories, without significant performance degradation. With cloud-based infrastructure services like **AWS** or **Google Cloud**, businesses can scale their applications efficiently as traffic spikes or as they expand into new markets.

4. **Flexibility in Development:** The MERN stack allows developers to have full control over the architecture and functionality of the application. As e-commerce websites often have unique business requirements (such as custom shopping carts, product recommendations, or third-party integrations), the flexibility offered by MERN ensures that the platform can be easily customized to meet these specific needs. Additionally, React.js allows for a component-driven architecture, making it easier to manage large-scale websites and maintain their codebases.
5. **Cost Efficiency:** Since all the technologies in the MERN stack are open-source and free to use, businesses can reduce their initial development costs. Furthermore, leveraging cloud platforms for hosting and scaling can provide a cost-effective way to manage resources based on demand. For startups or small businesses, MERN provides an affordable entry point for building an e-commerce platform without significant upfront costs.

CHAPTER 12

BIBLIOGRAPHY

1. Vite.js Website:

- ☐ *Next.js Documentation.*
 - ☐ Next. URL: <https://nextjs.org/>

2. Node.js Website:

- ☐ *Node.js Documentation.*
 - ☐ Node.js. URL: <https://nodejs.org/>

☐ **3. React.js Book:**

- ☐ React: Up and Running" by Stoyan Stefanov

4. Tailwind CSS Website:

- ☐ *Tailwind CSS Documentation.*
 - ☐ Tailwind CSS. URL: <https://tailwindcss.com/>

5. MongoDB Book:

- ☐ "MongoDB Basics"
by Peter Membrey and Tim Hawkins

6. Geeks for Geeks:

- ☐ <https://www.geeksforgeeks.org/>