Indira Gandhi National Open University

MCSP – 232

A Project Report On

SHOPPING CART SYSTEM

SUBMITTED BY:

XXXXXXXXXX

ENROLLMENT NO: **XXXXXXXXXX**

UNDER GUIDENCE OF:

XXXXXXXXXXXXXX

Submitted to the School of Computer and Information Sciences, IGNOU in partial fulfilment of the requirements for the award of the degree

MASTER IN COMPUTER APPLICATION (MCA)

Year of Submission: 2025

ignou
THE PEOPLE'S
UNIVERSITY

Indira Gandhi National Open University
Maidan Garhi
New Delhi – 110068

SHOPPING CART – MERN STACK WEB APPLICATION

INTRODUCTION

SHOPPING CART - Ecommerce Web Application, where shopping becomes a seamless and enjoyableexperience. Our platform combines convenience, variety, and security to provide you with an exceptional online shopping destination. Whether you're looking for the latest fashion trends, electronics, home decor, or any other products, we've got you covered.

At our Ecommerce Web Application, we understand the importance of user-friendly interfaces and intuitive navigation. We have designed our website to be easy to use, ensuring that you can find what you're looking for quickly and effortlessly. Our search functionality allows you to filter and sort through thousands of products, making it simple to locate the items that meet your needs and preferences.

Variety is key, and we take pride in offering an extensive selection of products from various categories. From popular brands to emerging designers, our Ecommerce Web Application showcases a diverse range of high-quality merchandise. Whether you're a fashion enthusiast,a tech-savvy individual, or someone looking for unique gifts, our platform caters to your interests.

Security is our utmost priority, and we have implemented robust measures to protect your personal and financial information. Our Ecommerce Web Application utilizes secure payment gateways and encryption protocols, ensuring that your transactions are safe and confidential. You can shop with peace of mind, knowing that your data is protected.

In addition to an exceptional shopping experience, our Ecommerce Web Application offers convenient features such as wishlists, order tracking, and customer support. You can easily create a wishlist of your favorite items, track the progress of your orders, and reach out to our dedicated customer support team for any inquiries or assistance.

We believe in delivering exceptional customer service, and our team is committed to providing prompt and reliable support. Whether you have questions about products, shipping, or returns, our knowledgeable representatives are here to help you every step of the way. Your satisfaction is our priority.

MODEL

The **MERN stack** itself is well-suited for **agile development** due to its modular and component-based nature. **Agile methodologies**, such as Scrum or Kanban, can be employedto manage the development process effectively. Here are some reasons why Agile is suitablefor MERN stack web application development:

1. **Iterative development:** Agile allows for incremental development and continuous delivery, which aligns well with the iterative nature of building applications in the MERN stack. It enables developers to break down complex features into smaller user stories or tasks that can be completed in short iterations, often referred to as sprints.

2. **Flexibility and adaptability:** Agile methodologies prioritize adaptability to change. In MERN stack development, requirements can evolve rapidly, and new features maybe added or modified throughout the development process. Agile allows the development team to respond to changing needs and adjust the project's direction accordingly.

3. **Collaboration and communication:** Agile methodologies emphasize close collaboration and communication among team members, stakeholders, and clients. MERN stack development often involves multiple technologies and components, making effective collaboration crucial. Agile practices like daily stand-up meetings, sprint reviews, and retrospectives facilitate better communication and teamwork.

4. **Quick feedback and continuous improvement:** Agile methodologies focus on frequent feedback loops, allowing developers to receive continuous feedback from stakeholders and end-users. MERN stack applications can benefit from this feedback-driven approach, enabling the development team to identify and address issues early in the development cycle, resulting in a more refined and user-friendly application.

In MERN (MongoDB, Express.js, React, and Node.js) web app development, there are several popular models or architectural patterns that are commonly used. These models helpin organizing the code and managing data in a structured manner. Here are some of the key models used in MERN development:

1. **MVC (Model-View-Controller):**

MVC is a widely used architectural pattern in web development. In this pattern, the application is divided into three components: the Model, which represents the data andbusiness logic, the View, which is responsible for rendering the user interface, and theController, which handles the user input and manages the flow of data between the Model and the View.

2. Mongoose Models:

When working with MongoDB in a MERN stack, Mongoose is a popular Object DataModeling (ODM) library used for schema creation and validation. Mongoose providesa way to define data models using JavaScript objects, allowing you to define the structure, types, and relationships between different data entities.

3. Data Access Objects (DAO):

DAO is a design pattern that provides an abstraction layer for accessing the data from the database. In a MERN stack, you can create DAOs to encapsulate the logic for interacting with MongoDB. These DAOs handle tasks such as querying the database, performing CRUD operations (Create, Read, Update, Delete), and managing data relationships.

4. RESTful APIs:

Representational State Transfer (REST) is an architectural style that is commonly used for designing web services. In a MERN stack, you can build RESTful APIs using Express.js on the server-side to define routes and handle HTTP requests (GET, POST,PUT, DELETE). These APIs serve as the interface between the frontend (built with React) and the backend (built with Node.js and Express.js).

5. GraphQL Schemas and Resolvers:

GraphQL is an alternative to RESTful APIs and provides a flexible query language for APIs. With GraphQL, you define a schema that describes the available data and operations, and resolvers that handle the queries and mutations. In a MERN stack, youcan use libraries like Apollo Server to implement GraphQL APIs on the server-side.

LIFE CYCLE

The MERN (MongoDB, Express.js, React.js, Node.js) stack is a popular combination of technologies used for building web applications. The MERN life cycle model refers to the typical workflow and stages involved in developing a MERN stack application. Here's an overview of the MERN life cycle model:

1. **Requirement Gathering:** In this initial stage, you gather the requirements for your application, including features, functionality, and design preferences. It involves discussions with stakeholders to understand their needs and expectations.

2. **Design and Planning:** Once the requirements are gathered, you start designing the architecture and planning the overall structure of your MERN application. Thisincludes creating wireframes, defining the database schema, and deciding on the components and user interface design.

3. **Backend Development with Node.js and Express.js:** In this stage, you begin developing the backend of your application using Node.js and Express.js. This involves creating server-side logic, setting up APIs, and integrating MongoDB for data storage. You handle tasks such as authentication, authorization, and database interactions.

4. **Frontend Development with React.js:** After the backend is in place, you move on to the frontend development using React.js. You create user interfaces, implement client-side routing, and connect to the backend APIs. React components are used to build interactive and dynamic user interfaces.

5. **Integration and Testing:** Once both the backend and frontend are developed, you integrate the components together to ensure proper communication between the serverand client. You also conduct various tests, including unit tests, integration tests, and end-to-end tests, to identify and fix any bugs or issues.

6. **Deployment:** After successful testing, you prepare your application for deployment. This typically involves configuring server environments, setting up production databases, optimizing performance, and addressing security considerations. You maydeploy your MERN application on cloud platforms like AWS, Azure, or Heroku.

7. **Maintenance and Updates:** Once the application is deployed, you enter the maintenance phase. Here, you monitor the application's performance, handle any user-reported issues, and apply updates and bug fixes as required. You may also add new features or enhancements based on user feedback or changing requirements.

It's important to note that the MERN life cycle model is iterative and agile in nature, allowing for continuous improvement and adaptation throughout the development process. Developers often follow agile methodologies like Scrum or Kanban to manage the development workflow effectively.

MODULES

1. **Admin Module:** The role of an admin is crucial in managing and maintaining the platform. The admin performs various tasks to ensure the smooth functioning of the eCommerce website. Here are some of the key responsibilities of an admin in an eCommerce web application:

   - Product Management: The admin module allows administrators to add, edit,and delete products on the website. It includes features such as adding product descriptions, uploading images, setting prices, managing inventory,and categorizing products.

   - Order Management: Administrators can view and manage customer ordersthrough the admin module. They can track order statuses, update order details, process refunds or cancellations, generate invoices, and manage shipping and delivery information.

   - Customer Management: The admin module provides functionality to managecustomer information, including viewing customer profiles, managing user accounts, updating contact details, and handling customer inquiries or support requests.

   - Content Management: Administrators can manage website content such asadding or editing pages, creating blog posts, managing FAQs, updating banners or promotional content, and managing other static or dynamic content displayed on the website.

   - User Management and Roles: The admin module allows administrators to manage user accounts, including creating new admin accounts, assigning roles and permissions, and controlling access to different sections or featuresof the admin panel.

2. **User Module:** The user module in an e-commerce web application typically encompasses the functionality and features related to user registration, authentication,profile management, and account settings. It allows users to create accounts, log in securely, update their personal information, manage their orders, and perform other actions specific to their individual accounts. Here are some key components and features commonly found in the user module of an e-commerce web application:

   - User Registration: The module should include a registration form that allows new users to create an account by providing their necessary details such as name, email address, password, and possibly additional information likeshipping address, contact number, etc.

   - User Authentication: Users should be able to log in to their accounts securely using their registered email address or username and password. This ensures that only authorized users can access their account information and perform actions specific to their accounts.

   - User Profile: Once logged in, users should have a dedicated profile page wherethey can view and update their personal information, including name, email address, contact details, shipping address, and other relevant details. Users may also have the option to upload a profile picture or customize their profilesettings.

   - Account Settings: The user module should provide a section for users to manage their account settings. This may include options to change passwords,update notification preferences, manage payment methods, and handle other account-related configurations.

   - Order History: Users should be able to view their order history, including details such as order date, items purchased, payment status, shipping status, and tracking information. This feature enables users to keep track of their pastpurchases and access relevant information whenever required.

Objective of the Project

The primary objective of this project, "Shopping Cart System," is to design, develop, and implement a robust, scalable, and user-friendly e-commerce web application utilizing the MERN (MongoDB, Express.js, React.js, Node.js) stack. This system aims to provide a comprehensive online shopping experience for customers and efficient management tools for administrators.

The modern digital landscape necessitates intuitive and efficient online platforms for commerce. Traditional retail is increasingly being supplemented, and often supplanted, by e-commerce, offering unparalleled convenience, wider product selection, and competitive pricing. The "Shopping Cart System" seeks to address the growing demand for a streamlined digital marketplace, enabling businesses to effectively present their products and manage sales, while providing consumers with an accessible and secure purchasing environment.

Specific Objectives Include:

To develop a secure and intuitive user interface (UI) and user experience (UX) for customers:

User-Centric Design: Implement a responsive design that adapts seamlessly across various devices (desktops, tablets, mobile phones) to ensure consistent accessibility and usability. This includes optimizing layouts, font sizes, and interactive elements for different screen real estates, leveraging modern CSS frameworks and responsive React components. The aim is to minimize cognitive load and maximize user satisfaction.

Intuitive Navigation: Create a clear and logical navigation structure that allows users to easily browse categories, search for products, view product details, add items to a cart, and proceed to checkout with minimal effort. This involves well-defined menu hierarchies, breadcrumbs, and search functionalities with filtering and sorting options.

Engaging Product Presentation: Design visually appealing product pages that effectively showcase product images (with zoom and multiple views), detailed descriptions, pricing, availability, and customer reviews. High-quality imagery and concise, informative text are paramount to drive purchasing decisions.

Seamless Shopping Cart Management: Provide a clear and editable shopping cart interface where users can view selected items, modify quantities, remove items, and see real-time updates on total costs. The cart should persist across sessions (e.g., using local storage or user accounts) for enhanced convenience.

Streamlined Checkout Process: Implement a multi-step, guided checkout flow that is simple, transparent, and minimizes friction. This includes clear steps for shipping address entry, payment method selection, order review, and confirmation. Error handling and real-time validation should guide users through the process efficiently.

To implement a robust and efficient backend system for product, order, and user management:

Scalable API Development: Build a RESTful API using Node.js and Express.js that supports all necessary CRUD (Create, Read, Update, Delete) operations for products, categories, users, orders, and other relevant entities. The API design will prioritize efficiency, maintainability, and scalability to handle increasing traffic and data volumes.

Database Management with MongoDB: Utilize MongoDB as the NoSQL database to store product information, user profiles, order details, and other application data. The schema design will be flexible to accommodate evolving data requirements and optimized for performance. Indexing strategies will be employed to accelerate data retrieval.

Secure Authentication and Authorization: Develop a secure user authentication system (e.g., JWT-based) to manage user registration, login, and session management. Implement role-based access control (RBAC) to differentiate between customer and administrator privileges, ensuring that only authorized users can perform specific actions (e.g., only admins can add/edit products).

Order Processing Logic: Develop the backend logic for handling order creation, status updates (e.g., pending, processing, shipped, delivered), payment integration, and inventory management. This includes deducting items from stock upon purchase and managing returns or cancellations.

Admin Panel Functionality: Provide a comprehensive administration dashboard to allow store owners to manage products (add, edit, delete, categorize), view and process orders, manage user accounts, and potentially generate sales reports. This centralized control ensures efficient operation of the e-commerce platform.

To ensure the security and integrity of user data and transactions:

Data Encryption: Implement robust encryption mechanisms for sensitive user data, particularly passwords (e.g., bcrypt hashing), both in transit (HTTPS/SSL/TLS) and at rest within the database.

Secure Payment Gateway Integration: Integrate with a reputable and secure third-party payment gateway (e.g., Stripe, PayPal) to handle all financial transactions, ensuring compliance with industry security standards (e.g., PCI DSS). The system will not store sensitive payment card information directly.

Input Validation and Sanitization: Implement rigorous input validation on both the client and server sides to prevent common web vulnerabilities such as SQL injection (though less relevant for NoSQL, similar principles apply to NoSQL injection), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). Data will be sanitized before storage and display.

Access Control: Enforce strict access control policies based on user roles (customer vs. admin) to prevent unauthorized access to sensitive data or functionalities.

Error Handling and Logging: Implement comprehensive error handling and logging mechanisms to identify and respond to security incidents, system failures, and unexpected behavior in a timely manner.

To incorporate features for scalability, maintainability, and future enhancements:

Modular Architecture: Design the application with a modular and component-based architecture (especially with React.js for the frontend and well-structured API routes for the backend) to facilitate easier maintenance, debugging, and future feature additions.

Performance Optimization: Optimize the application for performance, including efficient database queries, code splitting for frontend assets, image optimization, and caching strategies where appropriate, to ensure fast loading times and a smooth user experience.

Extensible Design: Develop the system with extensibility in mind, allowing for easy integration of new features, third-party services (e.g., analytics, marketing automation), and potential microservices architecture in the future.

Comprehensive Documentation: Maintain thorough documentation for the codebase, API endpoints, database schema, and deployment procedures to ensure ease of understanding for future development teams.

By achieving these objectives, the "Shopping Cart System" will not only provide a functional e-commerce platform but also serve as a scalable, secure, and maintainable foundation for various online retail ventures.

II. Project Planning and Scheduling

Effective project planning and scheduling are paramount to the successful delivery of any software development endeavor. This section outlines the proposed phases, tasks, and a high-level timeline for the "Shopping Cart System" project. Given the dynamic nature of software development, an Agile methodology, as mentioned in the synopsis, will be adopted, allowing for iterative development, continuous feedback, and adaptability to evolving requirements.

The project will be structured into several key phases, each with specific deliverables and milestones.

A. Project Phases and Activities
Phase 1: Inception and Planning (Weeks 1-3)

Activities:

Detailed Requirement Gathering and Analysis: Refine and expand upon the initial requirements. Conduct interviews with stakeholders (simulated for this proposal), research existing e-commerce platforms, and define functional and non-functional requirements in detail. This includes user stories, system features, performance expectations, and security considerations.

Technology Stack Finalization: Confirm the MERN stack and identify specific tools and libraries (e.g., specific React libraries, Express middleware, MongoDB ODM like Mongoose).

Architectural Design: Develop the high-level architecture, including data flow, module interactions, and API design. This will involve defining routes, request/response formats, and database schema in greater detail.

Database Schema Design: Create detailed Entity-Relationship (ER) diagrams (already partially present in the synopsis, but will be refined) and define collection structures, fields, data types, and relationships for MongoDB.

Wireframing and Mockups: Design low-fidelity wireframes and high-fidelity mockups for key user interfaces (e.g., homepage, product listing, product detail, shopping cart, checkout, admin dashboard).

Project Plan and Schedule Refinement: Develop a detailed project plan, including tasks, sub-tasks, dependencies, resource allocation, and a refined timeline with milestones.

Risk Assessment: Identify potential risks (technical, operational, security, schedule) and develop mitigation strategies.

Phase 2: Core Backend Development (Weeks 4-8)

Activities:

Environment Setup: Set up development, testing, and potentially staging environments.

API Development (Authentication & Authorization): Implement user registration, login, logout, and session management using JWT (JSON Web Tokens). Establish middleware for role-based access control (admin vs. customer).

User Management API: Develop endpoints for creating, reading, updating, and deleting user profiles.

Product Management API: Create APIs for adding, retrieving (with filtering/sorting), updating, and deleting products. Implement category management.

Order Management API: Develop APIs for creating orders, retrieving order history, updating order statuses, and basic inventory deduction.

Payment Integration API: Set up the backend for integrating with a chosen payment gateway (e.g., mock integration for testing, or actual integration with a sandbox environment).

Unit Testing (Backend): Write comprehensive unit tests for all API endpoints and business logic to ensure correctness and prevent regressions.

Phase 3: Core Frontend Development (Weeks 9-14)

Activities:

Component Development: Build reusable React components for UI elements (e.g., navigation bar, product cards, buttons, forms).

Page Layouts: Create static layouts for key pages (homepage, product listing, product detail, cart, checkout, user profile, admin dashboard).

State Management: Implement a state management solution (e.g., React Context API) to manage global application state.

API Integration (Frontend): Connect frontend components to the developed backend APIs, fetching and displaying data dynamically.

User Flows: Implement complete user flows, including product browsing, adding to cart, checkout process, user registration/login, and viewing order history.

Admin Dashboard UI: Develop the frontend for the admin panel, enabling administrators to manage products, view orders, and manage users.

Form Handling and Validation: Implement client-side form validation for user inputs.

Responsiveness Implementation: Ensure all frontend components are fully responsive and adapt to different screen sizes.

Phase 4: Integration, Testing, and Security Implementation (Weeks 15-18)

Activities:

Integration Testing: Test the seamless communication and interaction between frontend and backend components.

Security Implementation & Testing:

Implement HTTPS/SSL/TLS.

Refine password hashing (Bcrypt).

Implement input validation, sanitization, and output encoding to prevent XSS, CSRF, and injection attacks.

Configure CORS policies.

Implement rate limiting for critical endpoints (e.g., login).

Conduct penetration testing (simulated, or using automated tools).

Performance Testing: Identify and resolve performance bottlenecks (e.g., slow queries, large image loads, inefficient rendering).

User Acceptance Testing (UAT): Conduct UAT with simulated end-users and administrators to gather feedback and validate that the system meets user expectations.

Bug Fixing and Refinement: Address all identified bugs, UI/UX issues, and performance bottlenecks.

Phase 5: Deployment and Maintenance (Weeks 19-20 and Ongoing)

Activities:

Deployment Strategy: Plan and execute the deployment to a chosen cloud platform (e.g., Heroku, AWS, DigitalOcean).

Production Environment Setup: Configure production database, server environment, and necessary security measures.

Monitoring and Logging: Set up monitoring tools and logging systems to track application performance, errors, and security events.

Documentation: Finalize all technical and user documentation.

Post-Deployment Support & Maintenance: Provide ongoing support, apply bug fixes, and release minor updates.

Future Enhancements Planning: Based on initial feedback and market trends, begin planning for future features and improvements.

Detailed Project Scheduling (Agile Iterations)

Given the Agile methodology, the project will be broken down into sprints, typically 2-3 weeks in duration. This allows for frequent delivery of working software and early feedback.

Overall Timeline (20 Weeks / 10 Sprints of 2 weeks each)

Sprint 1-2 (Weeks 1-4): Inception & Planning Deep Dive, Basic Backend Setup

Focus: Core API authentication, user models, basic product API.

Deliverables: Detailed requirement document, API spec, initial database schema, functional login/registration endpoints, basic product listing endpoint.

Sprint 3-4 (Weeks 5-8): Advanced Backend, Frontend Setup

Focus: Product management APIs, order creation API, frontend environment setup, reusable React components.

Deliverables: CRUD for products/categories, basic order creation, initial frontend pages (home, product list), robust component library.

Sprint 5-6 (Weeks 9-12): Core User Experience & Payment Integration

Focus: Shopping cart functionality, full checkout flow, payment gateway integration.

Deliverables: Add/remove from cart, quantity updates, multi-step checkout, successful payment processing (sandbox).

Sprint 7-8 (Weeks 13-16): Admin Functionality & User Enhancements

Focus: Admin dashboard for product/order/user management, user profile management, order history viewing.

Deliverables: Fully functional admin panel, user profile updates, comprehensive order history for customers.

Sprint 9 (Weeks 17-18): Testing, Bug Fixing, Performance Optimization

Focus: Comprehensive testing (integration, performance, security), bug resolution.

Deliverables: Test reports, performance improvements, stable application ready for UAT.

Sprint 10 (Weeks 19-20): UAT, Documentation & Deployment Preparation

Focus: Final user acceptance testing, comprehensive documentation, deployment pipeline setup.

Deliverables: UAT sign-off, complete project documentation, deployed application.

Post-Deployment (Ongoing):

Monitoring, bug fixes, minor enhancements, and preparing for future scope.

TECHNOLOGIES

1 MERN Stack

1.1. MongoDB

MongoDB is an open-source database; it is also the leading NoSQL (*) database currently used by millions of people. It is written in one of the most popular programming languages today. In addition, MongoDB is cross-platform data that operates on the concepts of Collections and Documents, providing high performance with high availability and ease of expansion.

(*) NoSQL is a source database format that does not use Transact-SQL to access information, this database was developed on JavaScript Framework on JSON data type. With its introduction, it has overcome the disadvantages of RDBMS relational data model to improve operating speed, functionality, model scalability, cache etc.

Furthermore, MongoDB is a cross-platform database, performing on Collection and Document approach, it produces sharp production, huge availability, and effortless scalability.

Commonly used terms in MongoDB:

♦ **_id:** Almost every document required this field. The _id field illustrates a exceptional value in the MongoDB document. The _id field can also beinterpreted as the primary key in the document. If you add a new document, MongoDB will automatically generate a _id representing that document and beunique in the MongoDB database.

♦ **Collection:** A group of many documents in MongoDB. Collection can be interpreted as a corresponding table in the RDBMS (Relational DatabaseManagement System) database. Collection resides in a single database. Collections do not have to define columns, rows or data types first.

♦ **Cursor:** This is a pointer to the outcome set of a query. The client can emphasize over a cursor to get the result.

♦ **Database:** The location of the collections, similar to the RDMS database that contains the tables. Each Database has a separate file stored on physical memory. Some MongoDB owners may contain various databases.

♦ **Document:** A transcript belonging to a Collection. Documents, in turn, includename and value fields.

- **Field:** A name-value pair in a document. A document may not need all the fields. The fields are like columns in a relational database.

- **JSON:** Short for JavaScript Object Notation. Human readability is in the plaintext format representing structured data. JSON currently supports a lot of programming languages.

- **Index:** Exclusive data structures used to save a small allocation of data sets forsimple scanning. The index puts the value of a individual field or sets of fields, sorted by the value of these fields. Index effectively supports the analysis of queries. Without an index, MongoDB will have to scan all the documents of the set to choose the documents that pair the query. This scan is ineffective andrequires MongoDB to progress a vast amount of data.

1.2. Express Js

Express.js is a framework built on top of Nodejs. It provides powerful features for web or mobile development. Express.js supports HTTP and middleware methods, making the API extremely powerful and easy to use.

Express implements extra features to developer which help them get a better programming environment, not scaling down the speed of NodeJS.

Importantly, the well-known frameworks of NodeJS apply Express.js as a substance function, for instance: Sails.js,

1.3. ReactJS

- **Virtual-DOM:** It is a JavaScript object, each object contains all the information needed to create a DOM, when the data changes it will calculate the change between the object and the real tree, which will help optimize re- render DOM tree. It can be assumed that is a virtual model can handle client data.

- **Component**: React is built around components, not templates like other frameworks. A component can be created by the create Class function of the React object, the starting point when accessing this library.

  ReactJS creates HTML tags unlike we normally write but uses Component to wrap HTML tags into stratified objects to render.

  Among React Components, render function is the most important. It is a function that handles the generation of HTML tags as well as a demonstration of the ability to process via Virtual-DOM. Any changes of data at any time will be processed and updated immediately by Virtual-DOM.

- Props and State

Props (short for properties) are used to pass data from a parent component to its child components. They are read-only and should not be modified directly within the child component. Props are defined by the parent component and passed as attributes when rendering the child component. The child component can access the prop values using **this.props** (in class components) or **props** (infunctional components).

State is used to manage data that can change over time within a component. State is defined within a component's constructor and can be updated using the**setState()** method provided by React. Unlike props, state is private to the component and can only be accessed and modified by the component itself.

1.4. NodeJS

Node.js is an open source, a system application and furthermore is an environment for servers. Nodejs is an independent development platform built on Chrome's JavaScriptRuntime that we can build network applications quickly and easily. Google V8 JavaScript engine is used by Node.js to execute code. Moreover, a huge proportion ofessential modules are written in JavaScript

Nodejs accommodate a built-in library which allows applications to serve as a Web server left out demanding software like Apache HTTP Server, Nginx or IIS.

An event-driven, non-blocking I / O mechanisms (Input / Output) are implemented by Node.js. It optimizes application throughout and is extremely high extensible. Node.js use asynchronous in it functions. Therefore, Node.js processes and executes all tasks in the background (background processing).

products that have a lot of traffic are applying Node.js. Nonetheless, Node.js handle the application that need to spread expeditiously, develop innovation, or build Startup projects as rapidly as possible.

- **Benefits of Using Node JS**

Node.js is the exclusive application that with only a single thread, it can obtain and handle numerous connections. Building new threads for each query is not needed, therefore the structure expends the least amount of RAM and run rapidly. Secondly, Node.js produces the most of server property without generate latency with the JavaScript’s non-blocking I/O.

✓ **JSON APIs:** JSON Web services can take advantages of that because of theevent-driven, non-blocking I/O structures and JavaScript-enabled model.

- ✓ **Single page application:** NodeJS is very suitable with an application on a single page. Node.js has the capability to handle different requests concurrent and quick return. Node JS should be used in an application that does not have to reload the page, including users who makes a vast number of requests and need a quick procedure to show professionalism.

- ✓ **Shelling tools Unix:** Node.js usually uses Unix to work. They can handle multiple processes and return them for best performance. Programmers often use Node.js to build real Web applications like chat, feeds, etc.

- ✓ **Streaming Data:** Typical websites send HTTP requests and also receiveresponses. Node.js can handle many questions and feedback, so they aresuitable if the developer wants to create an application on the page. In addition,Node.js also builds proxies to stream the data, this is to ensure maximum operation for other data streams.

- ✓ **Real-time Web Application:** Node.js is sufficient to develop real-time innovations like chat apps, social networking services like Facebook, Twitter because of the opening of mobile application

2 System Specification required

Hardware Requirements:

1. **Computer:** Any modern computer with a minimum of 4GB RAM (8GB or morerecommended) and a multi-core processor.

2. **Storage:** Sufficient storage space to install the required software and store your project files.

Software Requirements:

1. **Operating System:** MERN stack development is platform-independent, so you can use Windows, macOS, or Linux based on your preference.

2. **Text Editor or Integrated Development Environment (IDE):** You'll need a texteditor or IDE to write your code. Popular choices include Visual Studio Code, Sublime Text, Atom, or WebStorm.

3. **Web Browser:** A modern web browser like Google Chrome, Mozilla Firefox, or Safari for testing your web application.

4. **Node.js:** Install the latest stable version of Node.js. Node.js comes with npm

(NodePackage Manager), which you'll use to install project dependencies.

5. **MongoDB:** Install MongoDB, a NoSQL database. 6. Git: Install Git, a version control system, to track changes in your code.
6. **Package Manager:** npm (included with Node.js) or Yarn (an alternative packagemanager) to manage your project's dependencies.
7. Postman is a popular tool used for making API calls and testing API endpoints. It provides a user-friendly interface that allows developers to send HTTP requests, view responses, and analyze API interactions.

Data Dictionary

A data dictionary is a structured document that provides detailed information about the data elements used in a system or application. In the context of the MERN stack, which consists of MongoDB, Express.js, React, and Node.js, the data dictionary would primarily focus on the data structures used in MongoDB, as well as any additional data models or schemas implemented in the application.

Here's an example of a data dictionary for a MERN stack application:

1. User Collection

- Fields:

- `id` (ObjectId): Unique identifier for the user.

- `name` (string): User's full name.

- `email` (string): User's email address.

- `password` (string): User's hashed password.

- `createdAt` (Date): Date and time when the user was created.

- `updatedAt` (Date): Date and time when the user was last updated.

2. Post Collection

- Fields:

- `id` (ObjectId): Unique identifier for the post.

- `title` (string): Title of the post.

- `content` (string): Content of the post.

- `author` (ObjectId): Reference to the user who created the post.

- `createdAt` (Date): Date and time when the post was created.

- `updatedAt` (Date): Date and time when the post was last updated.

3. Comment Collection

- Fields:

- `id` (ObjectId): Unique identifier for the comment.

- `content` (string): Content of the comment.

- \`author\` (ObjectId): Reference to the user who created the comment.
- \`post\` (ObjectId): Reference to the post the comment belongs to.
- \`createdAt\` (Date): Date and time when the comment was created.
- \`updatedAt\` (Date): Date and time when the comment was last updated.

This is a simplified example of a data dictionary for a basic MERN stack application. In practice, you may have additional collections, fields, or complex data relationships depending on your specific application requirements.

Note that the data dictionary typically focuses on the structure and fields of the data rather than the behavior or functionality of the application. It serves as a reference for developers and stakeholders to understand the data model and relationships within the MERN stack application.

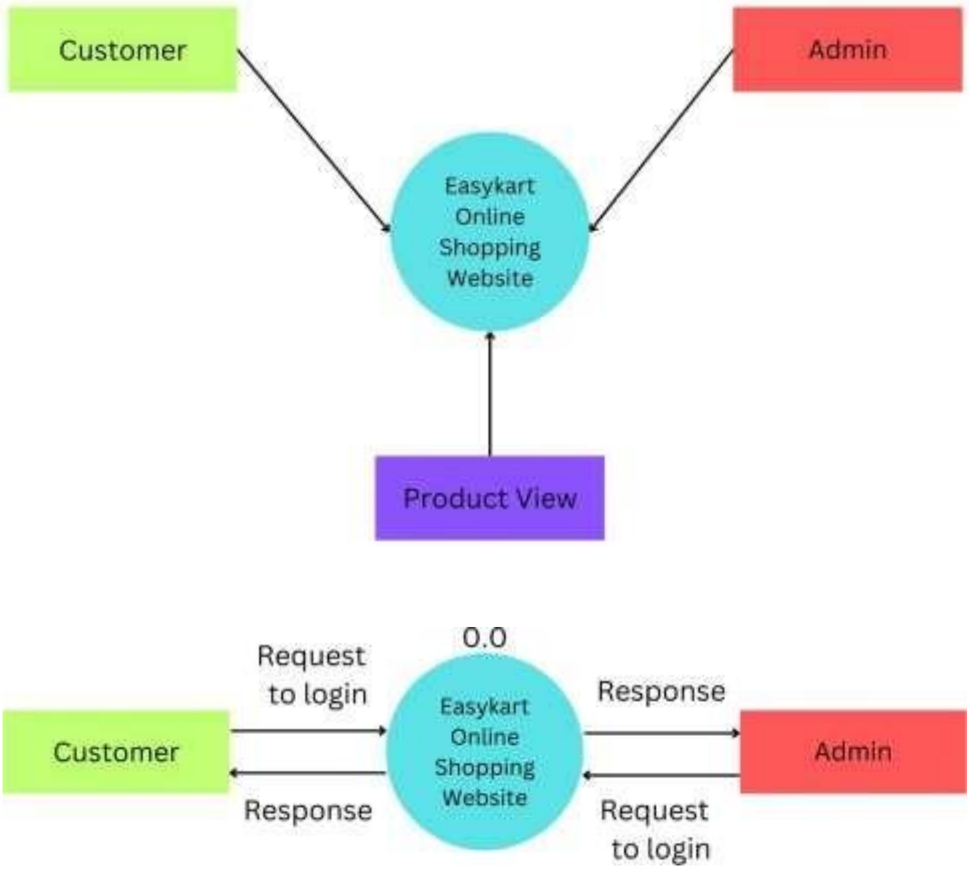
ANALYSIS

Software analysis and design include all activities that help transform requirement specification into implementation. It is the process of collecting and interpreting facts, identifying problems, and decomposing the system into its components. System analysis is conducted for the purpose of studying the system or its parts in order to determine its objectives. It is a problem-solving technique that improves the system and ensures that all components of the system are operating efficiently to achieve their purpose.

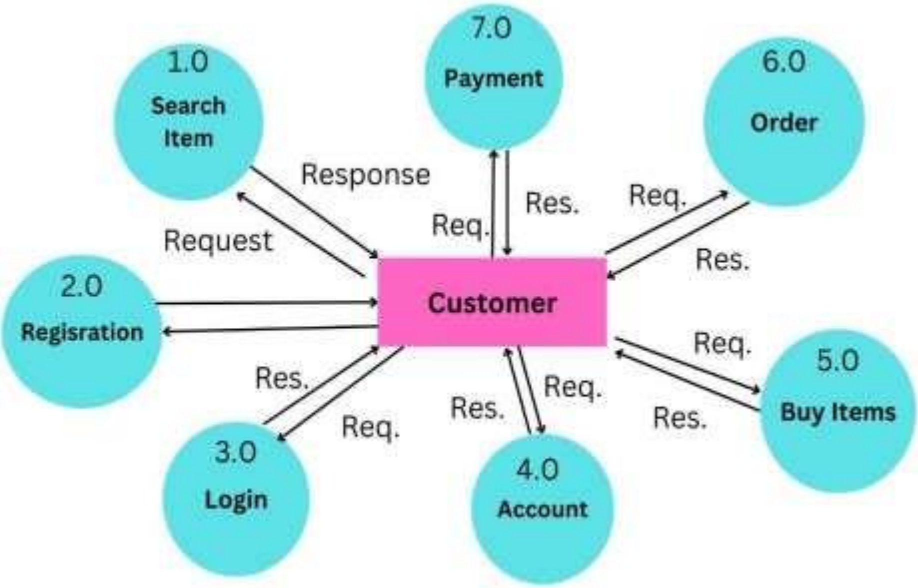
DATA FLOW DIAGRAM:

A data flow diagram (DFD) is a graphical or visual representation using a standardized set ofsymbols and notations to describe a business's 9 operations through data movement. They are often elements of a formal methodology such as Structured Systems Analysis and Design Method (SSADM). Superficially, DFDs can resemble flow charts or Unified Modeling Language (UML), but they are not meant to represent details of software logic.

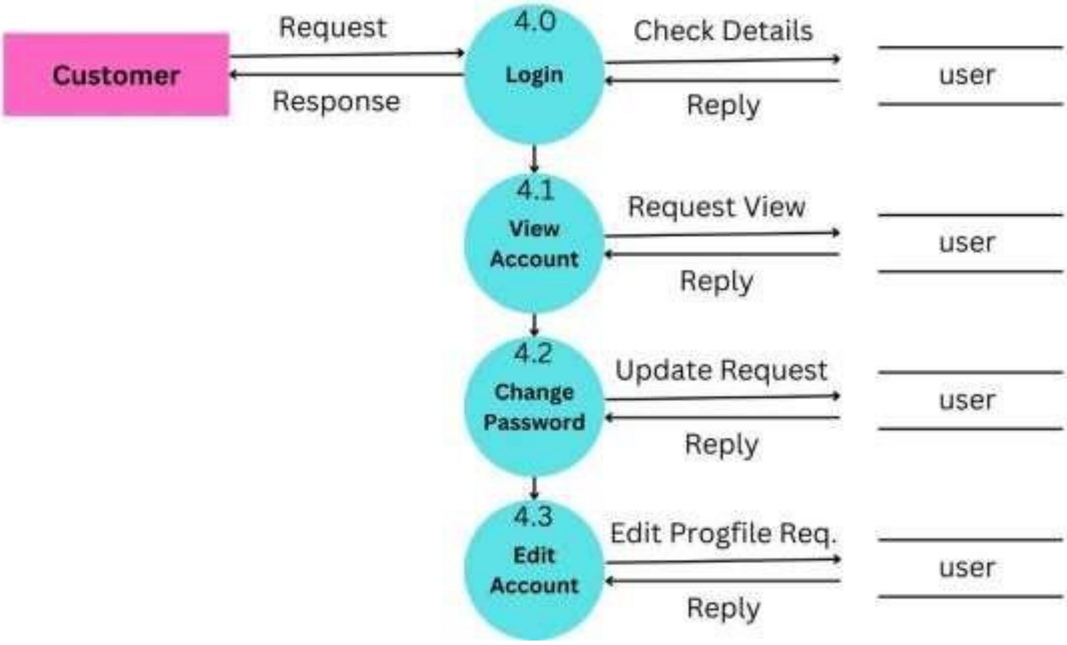
1. Zero level DFD (0.0)



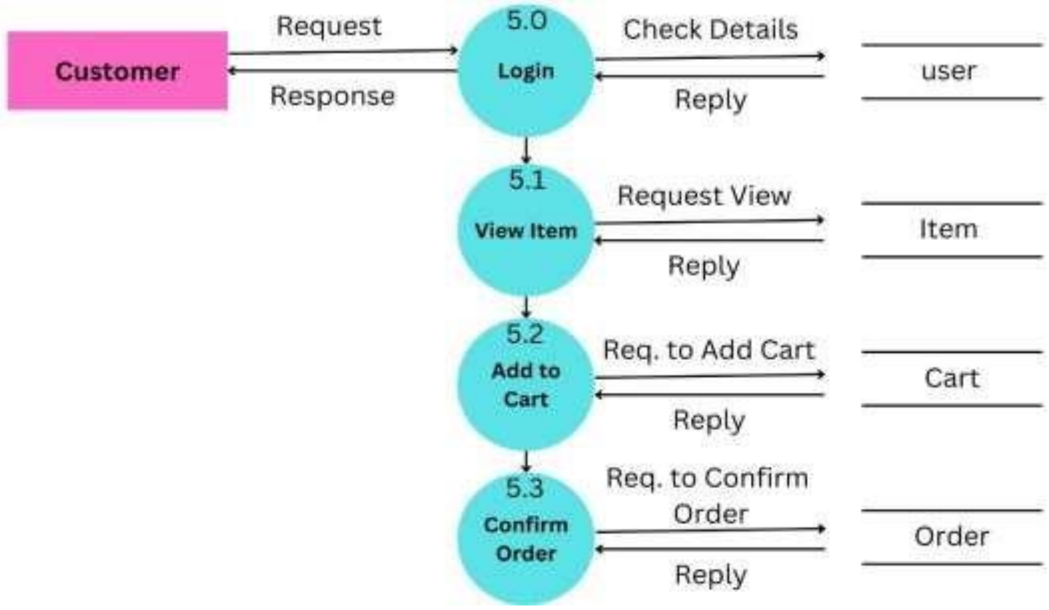
2. 1st level User DFD



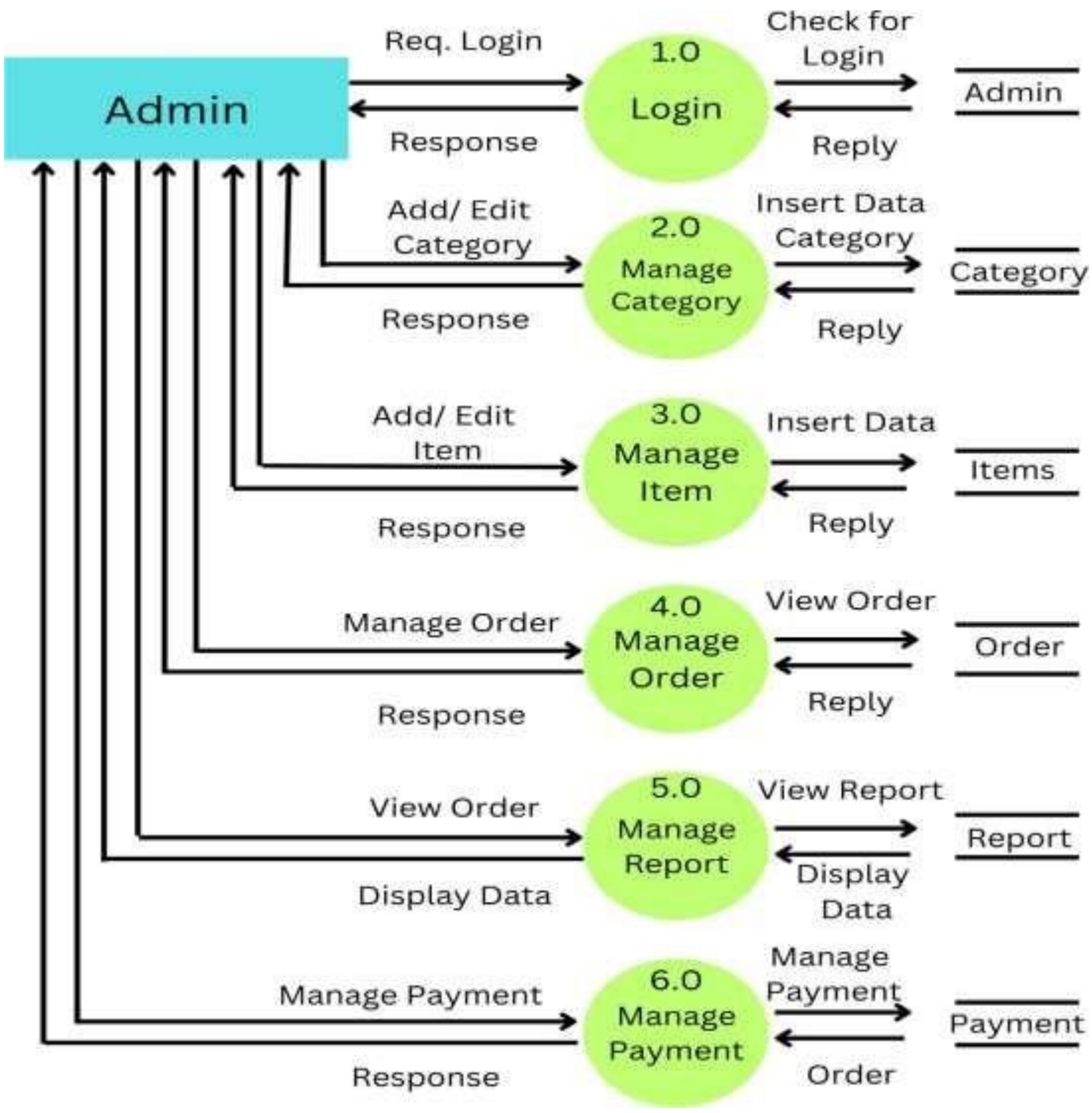
3. 2nd Level User DFD (4.0)



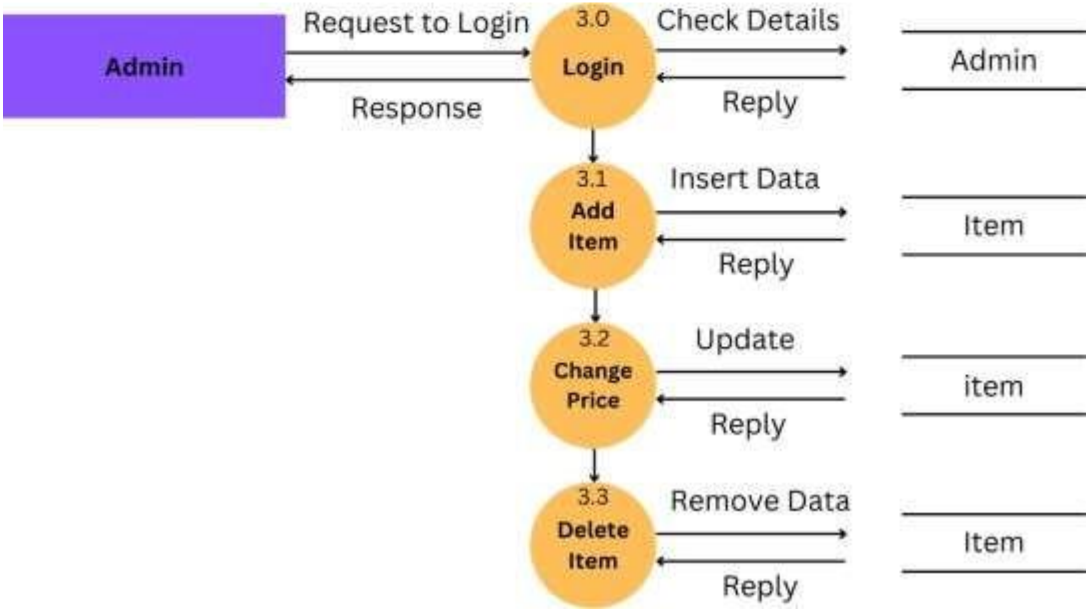
4. 2nd Level User DFD (5.0)



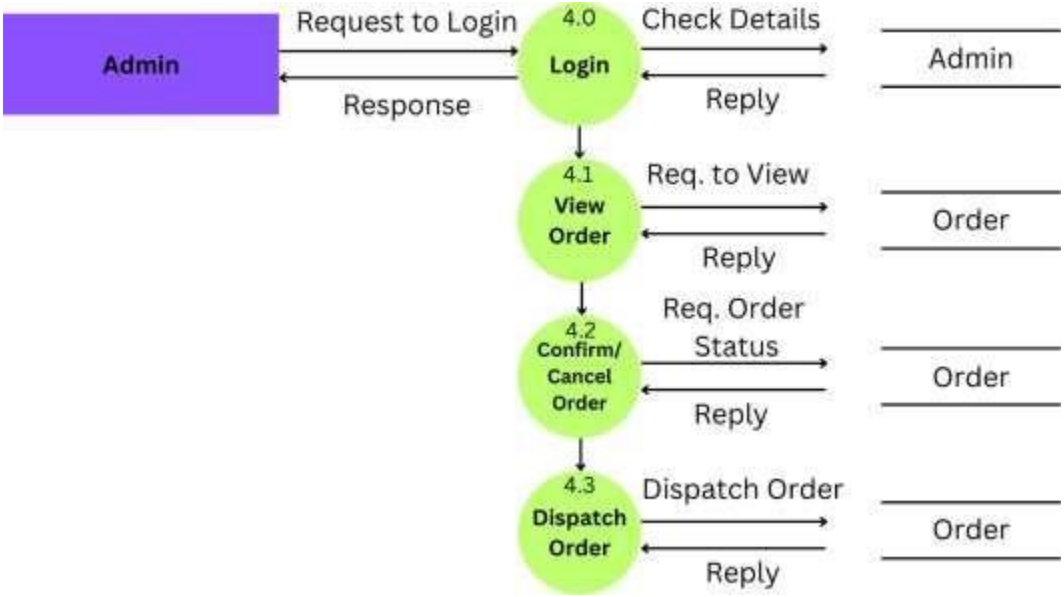
5. 1st Level Admin DFD



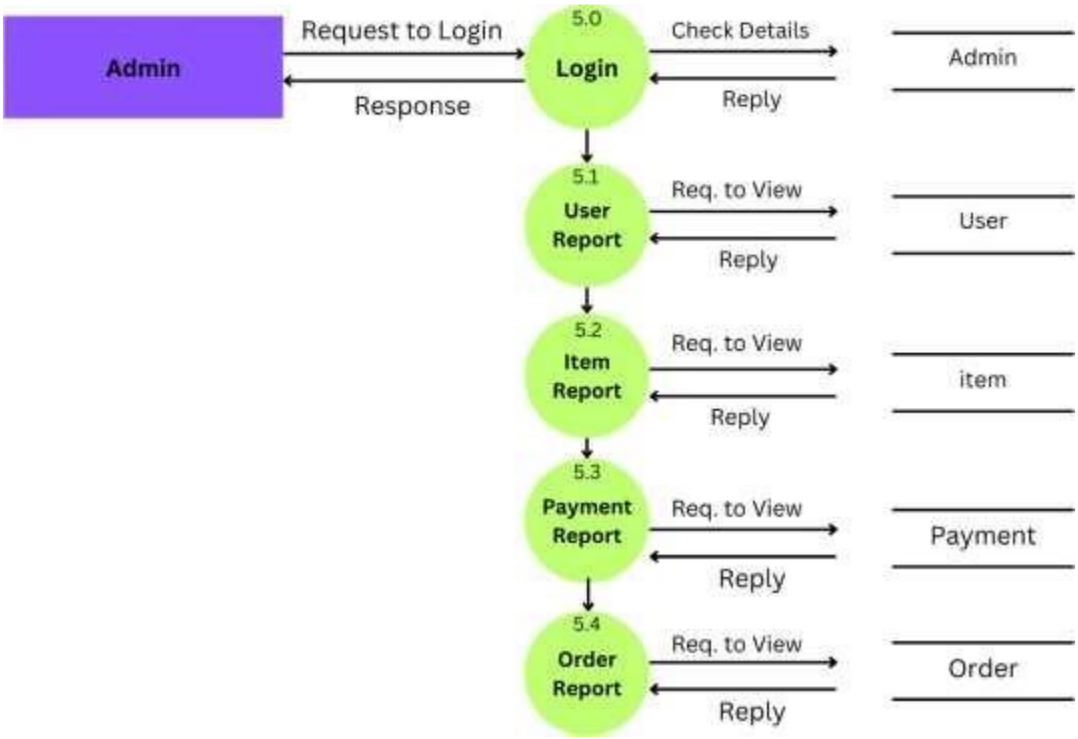
6. 2nd Level Admin DFD (3.0)



7. 2nd Level Admin DFD (4.0)



8. 2nd Level Admin DFD (5.0)



9. Use Case

A use case is a methodology used in system analysis to identify, clarify and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particularenvironment and related to a particular goal. The method creates a documentthat describes all the steps taken by a user to complete an activity.

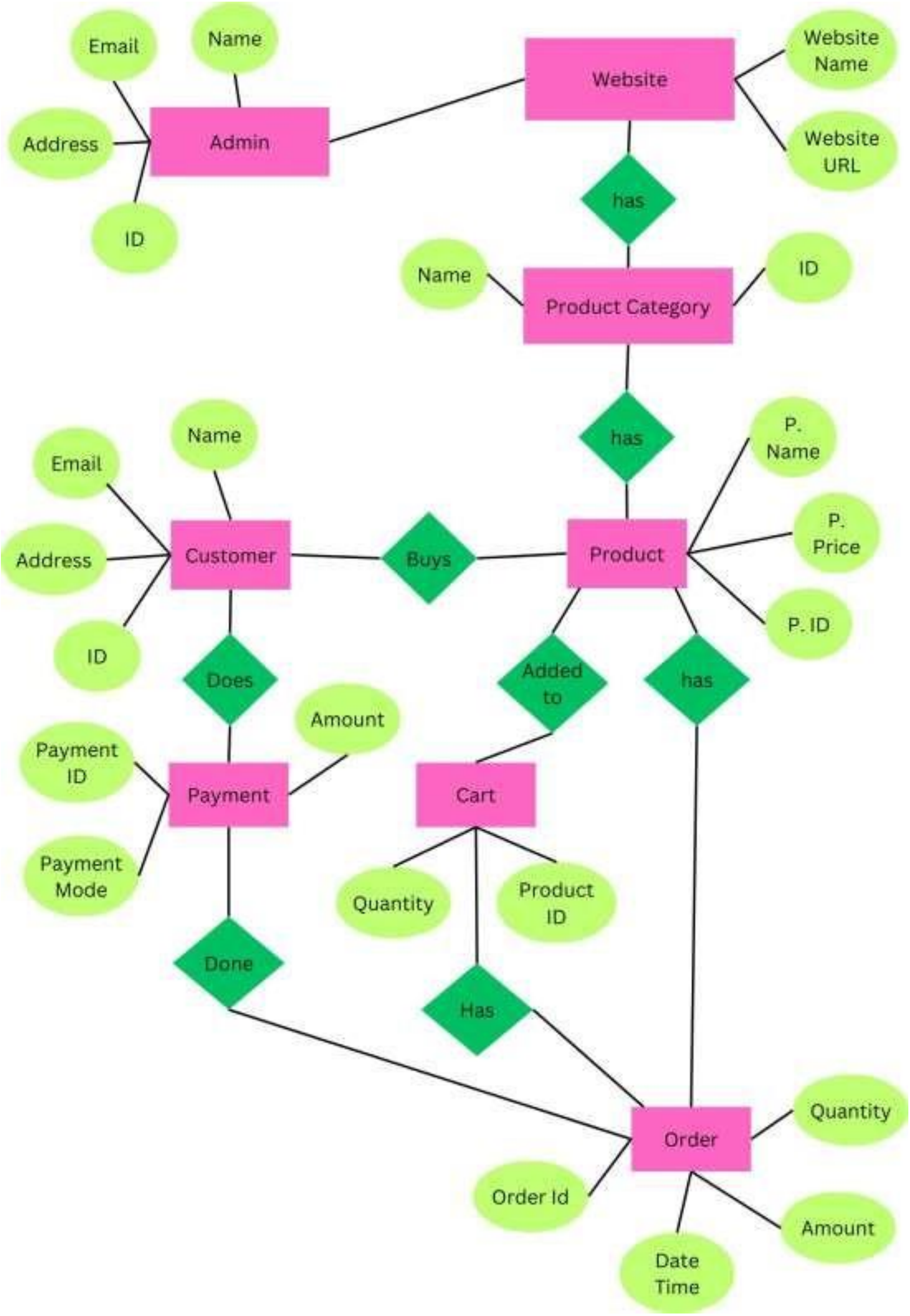
10. Sequence Diagram

A sequence diagram shows object interactions arranged time sequence. It delineates the items and classes engaged with the situation and the succession of message traded between the expected to do the usefulness of the situation. Sequence diagram are sometimes called event scenarios.

11. Activity Diagram

Activities diagram describes the behavior of the work flow of the system, where the activity is a state that does something, the main reason for using this diagram is to model the work flow beyond the system that is being designed it is also useful in analyzing use case by describing what actions need to take place and when should takeplace.

12. ER Diagram



Implementation of Security Mechanisms at Various Levels

Security is a paramount concern for any e-commerce system, as it deals with sensitive user data (personal information, payment details) and critical business operations (transactions, inventory). A multi-layered approach to security will be implemented, covering the frontend, backend, database, and network communication.

A. Frontend Security

While the frontend primarily deals with presenting data and collecting user input, it's the first line of defense against client-side attacks and plays a crucial role in user trust.

1. **Input Validation and Sanitization:**
   - **Purpose: Prevent malicious data from reaching the backend and causing vulnerabilities like XSS.**
   - **Implementation: All user inputs (e.g., search queries, form fields like name, email, address, product descriptions) will be validated on the client-side using JavaScript. This includes checking data types, length constraints, format (e.g., email regex), and preventing special characters that could lead to injection.**
   - **Techniques: Utilize React's state management and event handlers to perform real-time validation feedback to the user. Libraries like `formik` or `react-hook-form` with validation schemas (e.g., `yup`) can streamline this process. Although client-side validation enhances UX, it must always be complemented by server-side validation, as client-side validation can be bypassed.**
2. **Cross-Site Scripting (XSS) Prevention:**
   - **Purpose: Prevent attackers from injecting malicious scripts into web pages viewed by other users.**
   - **Implementation: When displaying user-generated content (e.g., product reviews, comments), ensure that it is properly escaped or sanitized. React inherently provides some protection against XSS by escaping content by default when rendering JSX, but for dynamically inserted HTML (e.g., using `dangerouslySetInnerHTML`), explicit sanitization libraries (e.g., `DOMPurify`) must be used.**
3. **Content Security Policy (CSP):**
   - **Purpose: Mitigate XSS attacks by specifying trusted sources of content (scripts, styles, images, etc.).**
   - **Implementation: Configure CSP headers on the server-side to instruct the browser about legitimate sources for various resources. This helps prevent the browser from loading malicious scripts injected from untrusted domains.**
4. **Secure Local Storage/Session Storage Usage:**
   - **Purpose: Avoid storing sensitive information client-side.**
   - **Implementation: Avoid storing sensitive user data (e.g., raw tokens, passwords, private keys) directly in `localStorage` or `sessionStorage`. JWTs (JSON Web Tokens) should be stored securely, ideally in HTTP-only cookies, which are less susceptible to XSS attacks than `localStorage` when managed correctly, or if stored in `localStorage`, additional measures (e.g., token refresh mechanisms, short expiry times) must be considered.**
5. **Protection against Clickjacking:**

    - **Purpose: Prevent malicious websites from tricking users into clicking on hidden elements of the e-commerce site.**
    - **Implementation: Implement X-Frame-Options or Content-Security-Policy with `frame-ancestors` directive to prevent the site from being embedded in iframes on other domains.**

B. Backend Security

The backend is the core of the application where most business logic and data interactions occur. Securing the backend is critical to protecting data integrity and confidentiality.

1. **Authentication and Authorization:**
    - **Purpose: Verify user identity and control access to resources based on roles.**
    - **Implementation:**
        - **User Registration: Implement strong password policies (minimum length, complexity requirements). Store passwords using strong, one-way hashing algorithms like Bcrypt with sufficient salt rounds. Never store plain text passwords.**
        - **Login: Implement rate limiting on login attempts to prevent brute-force attacks. Provide clear but generic error messages for failed login attempts to avoid leaking information about valid usernames.**
        - **JSON Web Tokens (JWT): Use JWTs for session management. Upon successful login, a short-lived access token and a longer-lived refresh token will be issued. The access token will be sent with each request to authorize access.**
        - **Role-Based Access Control (RBAC): Implement middleware in Express.js to verify user roles (e.g., `customer`, `admin`) before allowing access to specific routes or performing certain actions. For example, only an admin user can access `/api/admin/products` to add or edit products.**
        - **Token Management: Access tokens should have short expiry times to minimize the window of opportunity for token compromise. Refresh tokens should be used to obtain new access tokens, and they should be stored securely (e.g., in an HTTP-only cookie). Blacklisting compromised tokens will also be considered.**
2. **API Security:**
    - **Purpose: Protect API endpoints from unauthorized access and malicious use.**
    - **Implementation:**
        - **HTTPS/SSL/TLS: All communication between the frontend and backend will be encrypted using HTTPS to prevent eavesdropping and Man-in-the-Middle (MitM) attacks.**
        - **CORS (Cross-Origin Resource Sharing): Properly configure CORS headers in Express.js to specify which origins are allowed to access the API. This prevents unauthorized domains from making requests to the backend.**
        - **Rate Limiting: Implement rate limiting on sensitive or resource-intensive API endpoints (e.g., login, registration, password reset, search) to prevent denial-of-service (DoS) attacks and abuse. Libraries like `express-rate-limit` can be used.**
        - **Input Validation and Sanitization (Server-side): This is the most critical validation layer. All data received from the client must be validated and sanitized on the server before processing or storing it in the database. This prevents various injection attacks (e.g., NoSQL injection for**

        MongoDB, if not careful with query construction) and ensures data integrity.
        - **Parameter Tampering Prevention: Ensure that critical parameters (e.g., product price, order total) are never sent from the client or, if sent, are re-validated and re-calculated on the server-side to prevent users from manipulating them.**

3. **Error Handling and Logging:**
    - **Purpose: Identify and respond to security incidents and system anomalies.**
    - **Implementation: Implement centralized error handling in Express.js to catch unhandled exceptions and prevent sensitive information from being exposed in error responses. Use robust logging mechanisms (e.g., Winston, Morgan) to record significant events, including authentication attempts, failed requests, and system errors. Logs should be regularly reviewed and securely stored.**
4. **Dependency Management:**
    - **Purpose: Mitigate vulnerabilities introduced by third-party libraries.**
    - **Implementation: Regularly update Node.js, npm packages, and other dependencies to their latest stable versions to benefit from security patches. Use tools like `npm audit` or `Snyk` to scan for known vulnerabilities in dependencies.**
5. **Environment Variables:**
    - **Purpose: Securely manage sensitive configuration data.**
    - **Implementation: Store all sensitive configuration (e.g., database connection strings, API keys, JWT secrets) in environment variables rather than hardcoding them in the codebase. Use a `.env` file for local development and proper environment variable management in production.**
6. **HTTP Headers Configuration:**
    - **Purpose: Enhance overall application security.**
    - **Implementation: Configure various HTTP security headers (e.g., `Strict-Transport-Security`, `X-Content-Type-Options`, `X-Frame-Options`, `X-XSS-Protection`) using Express.js middleware like `helmet` to protect against common web vulnerabilities.**

C. Database Security (MongoDB)

Securing the database is critical as it holds all application data.

1. **Authentication and Authorization:**
    - **Purpose: Control who can access the database and what operations they can perform.**
    - **Implementation: Enable authentication in MongoDB. Create dedicated database users with specific roles and privileges (e.g., read-only access for some operations, read/write for others). Avoid using the default root user for application interactions.**
2. **Network Security:**
    - **Purpose: Restrict network access to the database.**
    - **Implementation: Configure network rules (firewalls, security groups) to only allow connections from the application server's IP address. Do not expose the MongoDB port to the public internet. Use a Virtual Private Cloud (VPC) for secure database deployment.**
3. **Encryption:**
    - **Purpose: Protect data at rest and in transit.**
    - **Implementation:**

        - **TLS/SSL: Enable TLS/SSL for all connections to the MongoDB database to encrypt data in transit.**
        - **Encryption at Rest: For highly sensitive data, consider encryption at rest (e.g., MongoDB's WiredTiger storage engine offers native encryption, or file-system level encryption). For specific sensitive fields within documents, application-level encryption can be implemented before storing data in MongoDB.**

4. **Input Validation (Mongoose Schema):**
    - **Purpose: Ensure data integrity and prevent malformed data.**
    - **Implementation: Utilize Mongoose schemas to define the structure, data types, and validation rules for documents stored in MongoDB. This adds another layer of validation before data is persisted.**
5. **Regular Backups:**
    - **Purpose: Data recovery in case of data loss or corruption.**
    - **Implementation: Implement a regular backup strategy for the MongoDB database, storing backups securely and off-site.**
6. **Least Privilege Principle:**
    - **Purpose: Minimize potential damage from compromised accounts.**
    - **Implementation: Configure database users with the minimum necessary privileges required for their operations. For example, the application user only needs read/write access to specific collections, not administrative privileges.**

D. Server/Infrastructure Security

Securing the underlying infrastructure hosting the MERN stack.

1. **Operating System Security:**
    - **Purpose: Protect the server from direct attacks.**
    - **Implementation: Keep the operating system (Linux, Windows Server) updated with the latest security patches. Disable unnecessary services, use strong SSH key authentication instead of passwords, and configure firewalls to restrict access to necessary ports only.**
2. **Process Isolation:**
    - **Purpose: Isolate different services to prevent a compromise in one from affecting others.**
    - **Implementation: Run Node.js applications as non-root users. If using microservices (future scope), consider containerization (Docker) for process isolation.**
3. **Monitoring and Alerting:**
    - **Purpose: Detect and respond to suspicious activities promptly.**
    - **Implementation: Implement system-level monitoring for CPU usage, memory, disk I/O, network traffic, and logs. Set up alerts for unusual patterns or potential security breaches.**
4. **Regular Security Audits and Penetration Testing:**
    - **Purpose: Proactively identify vulnerabilities.**
    - **Implementation: Conduct regular security audits and penetration testing by independent security experts to uncover vulnerabilities that might be missed during development.**

By integrating these security mechanisms at every level of the "Shopping Cart System," the project aims to provide a robust, secure, and trustworthy e-commerce platform for both users and administrators.

Future Scope and Further Enhancements

The "Shopping Cart System" project, developed using the MERN stack, lays a strong foundation for a functional e-commerce platform. However, the rapidly evolving nature of online commerce demands continuous innovation and enhancement. This section outlines the potential future scope and further enhancements that can be incorporated to evolve the system into a more comprehensive, intelligent, and user-centric solution. These enhancements are categorized to provide a structured roadmap for future development.

A. Core E-commerce Feature Enhancements

1. **Advanced Search and Filtering:**
    - **Current: Basic keyword search and category filtering.**
    - **Future: Implement faceted search (filtering by attributes like brand, price range, color, size, ratings), autocomplete suggestions, and "did you mean" functionality. Integration with search engines like Elasticsearch or Algolia could provide highly performant and relevant search results, handling large product catalogs efficiently.**
2. **Personalization and Recommendation Engine:**
    - **Current: No personalization.**
    - **Future: Implement algorithms to recommend products based on user browsing history, purchase history, items in the cart, popular products, or collaborative filtering (e.g., "customers who bought this also bought..."). This significantly enhances user engagement and sales. Data collected from user interactions would feed into this system, leveraging MongoDB's flexible schema for storing user behavior data.**
3. **Customer Reviews and Ratings System:**
    - **Current: Basic placeholder for reviews.**
    - **Future: Develop a robust system for customers to submit star ratings and written reviews for products. Include features like review moderation by administrators, sorting reviews (e.g., most helpful, newest), and filtering by star rating. This builds trust and aids purchasing decisions.**
4. **Wishlist Functionality:**
    - **Current: Not explicitly defined beyond "wishlists" in introduction.**
    - **Future: Allow users to save products to a personal wishlist for later purchase. Include options to share wishlists, move items from wishlist to cart, and receive notifications when wishlist items go on sale or are back in stock.**
5. **Multi-Vendor Support / Marketplace:**
    - **Current: Single vendor system.**
    - **Future: Transform the platform into a marketplace where multiple vendors can register, list their products, and manage their own inventories and orders. This would require dedicated vendor dashboards, commission management, and more complex order routing logic.**
6. **Inventory Management Enhancements:**
    - **Current: Basic deduction upon purchase.**
    - **Future: Implement real-time inventory tracking, low-stock alerts for administrators, stock reservation for items in cart, and bulk import/export functionalities for product data. Integration with barcode scanners could further streamline operations for physical inventory.**

7. **Order Tracking and Notifications:**
   - **Current: Basic order status viewing.**
   - **Future: Provide detailed order tracking (e.g., "Order placed," "Processing," "Shipped," "Out for delivery," "Delivered"). Implement automated email/SMS notifications for status changes, shipping updates, and delivery confirmations.**
8. **Promotions and Discount Management:**
   - **Current: No explicit mention.**
   - **Future: Develop a system for administrators to create and manage various types of promotions: percentage discounts, fixed amount discounts, "buy one get one free" offers, free shipping, coupon codes, and loyalty programs. This requires sophisticated rule-based pricing logic.**
9. **Product Variants and Options:**
   - **Current: Assumed simple products.**
   - **Future: Support for products with multiple variants (e.g., T-shirt in different sizes and colors) and configurable options, each with its own SKU, price, and inventory.**
10. **Abandoned Cart Recovery:**
    - **Current: Not addressed.**
    - **Future: Implement a system to track abandoned carts and send automated follow-up emails to remind users about their unpurchased items, potentially offering incentives.**

B. User Experience (UX) and Interface (UI) Improvements

1. **Enhanced User Dashboard:**
   - **Current: Basic profile and order history.**
   - **Future: A more comprehensive user dashboard displaying recent activity, saved payment methods, loyalty points, personalized recommendations, and easy access to support.**
2. **Improved Checkout Experience:**
   - **Current: Standard multi-step checkout.**
   - **Future: Implement guest checkout, one-page checkout, express checkout options (e.g., using saved addresses and payment methods), and integration with digital wallets (Apple Pay, Google Pay).**
3. **Advanced Image and Media Handling:**
   - **Current: Basic image display.**
   - **Future: Implement image optimization (lazy loading, responsive images), image galleries, product videos, and 360-degree product views to enhance visual merchandising.**
4. **Accessibility (A11Y) Compliance:**
   - **Current: Basic web standards.**
   - **Future: Ensure the entire application adheres to WCAG (Web Content Accessibility Guidelines) standards, making it usable for individuals with disabilities (e.g., screen reader compatibility, keyboard navigation).**

Scalability and Performance Enhancements

1. **Caching Mechanisms:**
    - **Current: Not explicitly mentioned beyond general performance optimization.**
    - **Future: Implement various caching strategies:**
        - **Client-side caching: Browser caching for static assets.**
        - **Server-side caching: Redis or Memcached for frequently accessed data (e.g., product listings, categories) to reduce database load.**
        - **CDN (Content Delivery Network): For serving static assets (images, CSS, JS) to users from geographically closer servers, improving load times.**
2. **Load Balancing and Horizontal Scaling:**
    - **Current: Single server deployment assumption.**
    - **Future: Deploy the Node.js backend behind a load balancer (e.g., Nginx) to distribute incoming traffic across multiple server instances. This allows for horizontal scaling to handle increased user loads and provides high availability.**
3. **Database Optimization and Sharding:**
    - **Current: Basic MongoDB usage.**
    - **Future: As data grows, implement advanced MongoDB indexing, optimize complex queries, and consider sharding strategies to distribute data across multiple servers, improving performance and scalability.**
4. **Microservices Architecture:**
    - **Current: Monolithic MERN application.**
    - **Future: Decompose the monolithic application into smaller, independent microservices (e.g., separate services for user management, product catalog, order processing, payment) communicating via APIs. This enhances scalability, fault isolation, and independent deployment.**
5. **Serverless Functions:**
    - **Current: Traditional Node.js server.**
    - **Future: Utilize serverless functions (e.g., AWS Lambda, Google Cloud Functions) for specific, event-driven tasks (e.g., image resizing upon upload, sending order confirmation emails) to reduce server overhead and costs.**

Integration with External Services

1. **CRM (Customer Relationship Management) Integration:**

- Current: No CRM.
- Future: Integrate with CRM systems (e.g., Salesforce, HubSpot) to manage customer interactions, support, and sales pipelines more effectively.

2. Email Marketing and Automation:
   - Current: No email capabilities.
   - Future: Integrate with email marketing platforms (e.g., Mailchimp, SendGrid) for transactional emails (order confirmations, shipping updates) and marketing campaigns (newsletters, promotions).
3. Analytics Tools:
   - Current: No explicit analytics.
   - Future: Integrate with Google Analytics, Mixpanel, or other analytics platforms to gain insights into user behavior, traffic sources, conversion rates, and sales performance.
4. Live Chat/Customer Support Integration:
   - Current: Manual customer support.
   - Future: Embed live chat widgets (e.g., Intercom, Zendesk Chat) or integrate with customer support ticketing systems for real-time assistance.
5. SMS Integration:
   - Current: No SMS.
   - Future: For critical notifications like order updates, delivery confirmations, or password resets.
6. Social Media Integration:
   - Current: No social login/sharing.
   - Future: Implement social login (e.g., Google, Facebook) for easier registration. Enable social sharing of products.

Advanced Business Intelligence and Reporting

1. Comprehensive Admin Dashboard and Reporting:
   - Current: Basic product/order/user management.
   - Future: Develop a more sophisticated admin dashboard with advanced analytics and reporting features, including:
     - Sales Reports: Daily, weekly, monthly, yearly sales, sales by product, sales by category, top-selling products.

        - **Customer Reports: New customers, active customers, customer lifetime value.**
        - **Inventory Reports: Stock levels, low-stock items, fast/slow-moving items.**
        - **Marketing Reports: Campaign performance, coupon usage.**
        - **Dashboards with Visualizations: Use charting libraries (e.g., Recharts for React, D3.js) to provide interactive data visualizations for key performance indicators (KPIs).**

2. **A/B Testing Framework:**
    - **Current: No A/B testing.**
    - **Future: Integrate a framework that allows for A/B testing of different UI elements, product presentations, or pricing strategies to optimize conversion rates.**

DevOps and Deployment Enhancements

1. **CI/CD Pipeline:**
    - **Current: Manual deployment.**
    - **Future: Implement a Continuous Integration/Continuous Deployment (CI/CD) pipeline using tools like Jenkins, GitLab CI/CD, or GitHub Actions. This automates testing and deployment processes, ensuring faster and more reliable releases.**
2. **Containerization (Docker) and Orchestration (Kubernetes):**
    - **Current: Direct server deployment.**
    - **Future: Containerize the MERN application components using Docker. Deploy and manage these containers using Kubernetes for robust orchestration, scaling, and self-healing capabilities.**

By planning for these future enhancements, the "Shopping Cart System" can evolve into a dynamic, feature-rich, and highly competitive e-commerce solution capable of adapting to market demands and providing a superior user experience. This phased approach allows for continuous delivery of value and ensures the long-term viability and success of the project.

1. Coding

~~~~~~~~~~~~~~~~~~~~~Frontend Codes~~~~~~~~~~~~~~~~~~~~~~~~

=============== frontend/src/App.js=====================

```
import { Routes, Route } from "react-router-dom"; import
HomePage from "./pages/HomePage"; import About from
"./pages/About";
import Contact from "./pages/Contact";import Policy
from "./pages/Policy";
import Pagenotfound from "./pages/Pagenotfound"; import
Register from "./pages/Auth/Register"; import Login from
"./pages/Auth/Login";
import Dashboard from "./pages/user/Dashboard";
import PrivateRoute from "./components/Routes/Private"; import
ForgotPasssword from "./pages/Auth/ForgotPasssword"; import
AdminRoute from "./components/Routes/AdminRoute";
import AdminDashboard from "./pages/Admin/AdminDashboard"; import
CreateCategory from "./pages/Admin/CreateCategory"; import CreateProduct
from "./pages/Admin/CreateProduct"; import Users from
"./pages/Admin/Users";
import Orders from "./pages/user/Orders"; import Profile from
"./pages/user/Profile"; import Products from
"./pages/Admin/Products";
import UpdateProduct from "./pages/Admin/UpdateProduct"; import
Search from "./pages/Search";
import ProductDetails from "./pages/ProductDetails"; import
Categories from "./pages/Categories";
import CategoryProduct from "./pages/CategoryProduct"; import
CartPage from "./pages/CartPage";
import AdminOrders from "./pages/Admin/AdminOrders";function App()
{
          return (
           <>
            <Routes>
             <Route path="/" element={<HomePage />} />
             <Route path="/product/:slug" element={<ProductDetails />} />
             <Route path="/categories" element={<Categories />} />
             <Route path="/cart" element={<CartPage />} />
             <Route path="/category/:slug" element={<CategoryProduct />} />
             <Route path="/search" element={<Search />} />
             <Route path="/dashboard" element={<PrivateRoute />}>
              <Route path="user" element={<Dashboard />} />
              <Route path="user/orders" element={<Orders />} />
```

```
          <Route path="user/profile" element={<Profile />} />
        </Route>
        <Route path="/dashboard" element={<AdminRoute />}>
          <Route path="admin" element={<AdminDashboard />} />
          <Route path="admin/create-category" element={<CreateCategory />} />
          <Route path="admin/create-product" element={<CreateProduct />} />
          <Route path="admin/product/:slug" element={<UpdateProduct />} />
          <Route path="admin/products" element={<Products />} />
          <Route path="admin/users" element={<Users />} />
          <Route path="admin/orders" element={<AdminOrders />} />
        </Route>
        <Route path="/register" element={<Register />} />
        <Route path="/forgot-password" element={<ForgotPasssword />} />
        <Route path="/login" element={<Login />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="/policy" element={<Policy />} />
        <Route path="*" element={<Pagenotfound />} />
      </Routes>
    </>
  );
}

export default App;
```

============ frontend/src/page/About.js==================

```
import React from "react";
import Layout from "../Components/Layout/Layout"; import
aboutUsImage from "../images/About-Us-page.png";

const About = () => {
  return (
    <Layout title={"About - SHOPPING CART"}>
      <div className="row aboutus">
        <div className="col-mid-6 image">
          <img src={aboutUsImage} alt="About Us" style={{ width: "100%" }} />
        </div>
        <div className="col-md-4">
          <h1 className="bg-dark p-1 text-white text-center">About Us</h1>
          <p className="text-justify mt-2">
            Welcome to our ecommerce website! We are an online store
            dedicatedto providing you with a seamless and enjoyable shopping
            experience.
```

```
              Our platform offers a wide range of products from various
              categories, ensuring that you can find everything you need in one
              convenient place.
            </p>
            <p>
              At our ecommerce website, we prioritize customer satisfaction above
              all else. We strive to offer high-quality products at competitive prices,
              so you can shop with confidence knowing that you are gettinggreat
              value for your money. Our team carefully selects each item in our
              inventory, ensuring that they meet our strict standards of
              quality and reliability.
            </p>
          </div>
        </div>
      </Layout>
    );
};

export default About;
```

============ frontend/src/page/Contact.js===================

```
import React from "react";
import Layout from "../Components/Layout/Layout";
import contactImage from "../images/HomeTown-Fan-Support-Web.png"; import {
TfiEmail, TfiHeadphoneAlt } from "react-icons/tfi";
import { MdOutlineAddIcCall } from "react-icons/md";

  const Contact = () => {
    return (
      <Layout title={"Contact - SHOPPING CART"}>
        <div className="row contactus">
          <div className="col-mid-6 image">
            <img src={contactImage} alt="Contact Us" style={{ width: "100%" }} />
          </div>
          <div className="col-md-4">
            <h1 className="bg-dark p-2 text-white text-center">Contact Us</h1>
            <p className="text-justify mt-2">
              Thank you for visiting our website! We value your interest and are
              here to assist you in any way we can. If you have any questions,
              concerns, or feedback, please don't hesitate to get in touch with
              us. We are committed to providing exceptional support and ensuring
```

```
                    your experience with our ecommerce platform is seamless.
                  </p>
                  <p className="mt-3">
                    <TfiEmail /> www.help@SHOPPING CART.com
                  </p>
                  <p className="mt-3">
                    <TfiHeadphoneAlt /> +91 8969989233
                  </p>
                  <p className="mt-3">
                    <MdOutlineAddIcCall /> 1800-0000-0000
                  </p>
                </div>
              </div>
            </Layout>
          );
};

export default Contact;
```

============ frontend/src/page/PageNotFound.js============

```
import React from "react";
import Layout from "../Components/Layout/Layout";import { Link }
from "react-router-dom";

          const PageNotFound = () => {
            return (
              <Layout title={"Page Not Found - SHOPPING CART"}>
                <div className="pnf">
                  <h1 className="pnf-title">404</h1>
                  <h2 className="pnf-heading">Oops! Page Not Found</h2>
                  <Link  to="/"  className="pnf-btn">
                    Go Back
                  </Link>
                </div>
              </Layout>
            );
};

export default PageNotFound;
```

============ frontend/src/page/Policy.js===================

```
import React from "react";
import    Layout    from    "../Components/Layout/Layout";   import
policyImage from "../images/privacy.png";

          const Policy = () => {
            return (
              <Layout title={"Policy - SHOPPING CART"}>
                <div className="row policy">
                  <div className="col-mid-6 image">
                    <img src={policyImage} alt="policy" />
                  </div>
                  <div className="col-md-4">
                    <h1 className="bg-dark p-2 text-white text-center">Privacy Policy</h1>
                    <p
                      className="text-justify mt-2"
                      style={{ padding: "5px", lineHeight: "30px" }}
                    >
                      <ol type="1">
                        <li className="policyList">Information collection</li>
                        <li className="policyList">Data usage</li>
                        <li className="policyList">Data sharing</li>
                        <li className="policyList">Cookies and tracking technologies</li>
                        <li className="policyList">Security measures</li>
                        <li className="policyList">User choices and rights</li>
                        <li className="policyList">Data retention</li>
                        <li className="policyList">Legal basis</li>
                        <li className="policyList">Updates to the privacy policy</li>
                      </ol>
                    </p>
                  </div>
                </div>
              </Layout>
            );
};

export default Policy;
```

============ frontend/src/page/CartPage.js=============

```
import React, { useState, useEffect } from "react"; import Layout
from "./../components/Layout/Layout"; import { useCart } from
"../context/cart";
import { useAuth } from "../context/auth"; import {
useNavigate } from "react-router-dom";
import DropIn from "braintree-web-drop-in-react"; import {
AiFillWarning } from "react-icons/ai"; import axios from "axios";
import toast from "react-hot-toast"; import
"../styles/CartStyles.css";

const CartPage = () => {
          const [auth, setAuth] = useAuth();
          const [cart, setCart] = useCart();
          const [clientToken, setClientToken] = useState("");
          const [instance, setInstance] = useState("");
          const [loading, setLoading] = useState(false);
          const navigate = useNavigate();

          //total price
          const totalPrice = () => {
           try {
            let total = 0;
            cart?.map((item) => {
            total = total + item.price;
            });
            return total.toLocaleString("en-US", {
             style: "currency",
             currency: "USD",
            });
           } catch (error) {
            console.log(error);
           }
          };
          //detele item
          const removeCartItem = (pid) => {
           try {
            let myCart = [...cart];
            let index = myCart.findIndex((item) => item._id === pid);
            myCart.splice(index, 1);
            setCart(myCart);
```

```
    localStorage.setItem("cart", JSON.stringify(myCart));
  }  catch  (error)  {
    console.log(error);
  }
};

//get payment gateway token
const getToken = async () => {
try {
    const { data } = await axios.get("/api/v1/product/braintree/token");
    setClientToken(data?.clientToken);
  }  catch  (error)  {
    console.log(error);
  }
};
useEffect(() => {
  getToken();
}, [auth?.token]);

//handle payments
const handlePayment = async () => {
  try {
    setLoading(true);
    const { nonce } = await instance.requestPaymentMethod();
    const { data } = await axios.post("/api/v1/product/braintree/payment", {
      nonce,
      cart,
    });
    setLoading(false);
    localStorage.removeItem("cart");
    setCart([]);
    navigate("/dashboard/user/orders")
    ;
    toast.success("Payment Completed Successfully ");
  }  catch  (error)  {
    console.log(error);
    setLoading(false);
  }
};
return (
  <Layout>
    <div className=" cart-page">
      <div className="row">
        <div className="col-md-12">
    <h1 className="text-center bg-light p-2 mb-1">
      {!auth?.user
        ? "Hello Guest"
        : `Hello  ${auth?.token && auth?.user?.name}`}
      <p className="text-center">
        {cart?.length
          ? `You Have ${cart.length} items in your cart ${
              auth?.token ? "" : "please login to checkout !"
            }`
          : " Your Cart Is Empty"}
      </p>
    </h1>
  </div>
</div>
<div className="container ">
  <div className="row ">
    <div className="col-md-7  p-0 m-0">
      {cart?.map((p) => (
        <div className="row card flex-row" key={p._id}>
          <div className="col-md-4">
            <img
              src={`/api/v1/product/product-photo/${p._id}`}
              className="card-img-top"
              alt={p.name}
              width="100%"
              height={"130px"}
            />
          </div>
          <div className="col-md-4">
            <p>{p.name}</p>
            <p>{p.description.substring(0, 30)}</p>
            <p>Price : {p.price}</p>
          </div>
          <div className="col-md-4 cart-remove-btn">
            <button
              className="btn                  btn-danger"
              onClick={() => removeCartItem(p._id)}
            >
              Remove
            </button>
          </div>
        </div>
      ))}
```

```
</div>
<div className="col-md-5 cart-summary ">
  <h2>Cart Summary</h2>
  <p>Total | Checkout | Payment</p>
  <hr />
  <h4>Total : {totalPrice()} </h4>
  {auth?.user?.address ? (
    <>
      <div className="mb-3">
        <h4>Current Address</h4>
        <h5>{auth?.user?.address}</h5>
        <button
          className="btn btn-outline-warning"
          onClick={() => navigate("/dashboard/user/profile")}
        >
          Update Address
        </button>
      </div>
    </>
  ) : (
    <div className="mb-3">
      {auth?.token ? (
        <button
          className="btn btn-outline-warning"
          onClick={() => navigate("/dashboard/user/profile")}
        >
          Update Address
        </button>
      ) : (
        <button
          className="btn  btn-outline-warning"
          onClick={() =>
            navigate("/login",  {
              state: "/cart",
            })
          }
        >
          Plase Login to checkout
        </button>
      )}
    </div>
  )}
  <div className="mt-2">
```

```
                    {!clientToken || !auth?.token || !cart?.length ? (
                      ""
                    ) : (
                      <>
                        <DropIn
                          options={{
                            authorization: clientToken,
                            paypal: {
                              flow: "vault",
                            },
                          }}
                          onInstance={(instance) => setInstance(instance)}
                        />

                        <button
                          className="btn btn-primary"
                          onClick={handlePayment}
                          disabled={loading || !instance || !auth?.user?.address}
                        >
                          {loading ? "Processing ...." : "Make Payment"}
                        </button>
                      </>
                    )}
                  </div>
                </div>
              </div>
            </div>
          </div>
        </Layout>
      );
};

export default CartPage;
```

============ frontend/src/page/Categories.js=============

```
import React, { useState, useEffect } from "react";import { Link
} from "react-router-dom";
import useCategory from "../hooks/useCategory"; import Layout
from "../components/Layout/Layout";const Categories = () => {
          const          categories          =
          useCategory();return (
         <Layout title={"All Categories"}>
          <div className="container" style={{ marginTop: "100px" }}>
           <div className="row container">
            {categories.map((c) => (
             <div className="col-md-4 mt-5 mb-3 gx-3 gy-3" key={c._id}>
              <div className="card">
               <Link to={`/category/${c.slug}`} className="btn cat-btn">
                {c.name}
               </Link>
              </div>
             </div>
            ))}
           </div>
          </div>
         </Layout>
        );
};

export default Categories;
```

===========
frontend/src/page/CategoryProduct.js===========

```
import React, { useState, useEffect } from "react"; import Layout
from "../components/Layout/Layout";
import  {  useParams,  useNavigate  }  from  "react-router-dom"; import
"../styles/CategoryProductStyles.css";
import     axios     from     "axios";     const
CategoryProduct  =  ()  =>  {const  params  =
useParams();
        const navigate = useNavigate();
        const [products, setProducts] = useState([]);
        const      [category,      setCategory]      =
        useState([]);

        useEffect(() => {
         if (params?.slug) getPrductsByCat();
        }, [params?.slug]);
        const getPrductsByCat = async () => {
         try {
          const { data } = await axios.get(
           `/api/v1/product/product-category/${params.slug}`
          );
          setProducts(data?.products)
          ;
```

```
      setCategory(data?.category
      );
    } catch (error) {
      console.log(error);
    }
  };

  return (
    <Layout>
      <div className="container mt-3 category">
        <h4 className="text-center">Category - {category?.name}</h4>
        <h6 className="text-center">{products?.length} result found </h6>
        <div className="row">
          <div className="col-md-9 offset-1">
            <div className="d-flex flex-wrap">
              {products?.map((p) => (
                <div className="card m-2" key={p._id}>
                  <img
                    src={`/api/v1/product/product-photo/${p._id}`}
                    className="card-img-top"
                    alt={p.name}
                  />
                  <div className="card-body">
                    <div className="card-name-price">
                      <h5 className="card-title">{p.name}</h5>
                      <h5 className="card-title card-price">
                        {p.price.toLocaleString("en-US", {
                          style: "currency",
                          currency: "USD",
                        })}
                      </h5>
                    </div>
                    <p className="card-text ">
                      {p.description.substring(0, 60)}...
                    </p>
                    <div className="card-name-price">
                      <button
                        className="btn btn-info ms-1"
                        onClick={() => navigate(`/product/${p.slug}`)}
                      >
                        More Details
                      </button>
                      {/* <button
                    className="btn btn-dark ms-1"
                    onClick={() => {
                        setCart([...cart,           p]);
                        localStorage.setItem(
                        "cart",
                        JSON.stringify([...cart, p])
                        );
                        toast.success("Item Added to cart");
                      }}
                    >
                      ADD TO CART
                    </button> */}
                      </div>
                    </div>
                  </div>
                ))}
              </div>
              {/* <div className="m-2 p-3">
              {products && products.length < total && (
                <button
                  className="btn btn-warning"
                  onClick={(e)         =>          {
                  e.preventDefault();
                  setPage(page + 1);
                  }}
                >
                  {loading ? "Loading ..." : "Loadmore"}
                </button>
              )}
            </div> */}
            </div>
          </div>
        </div>
      </Layout>
    );
};

export default CategoryProduct;
```

==========
frontend/src/page/ProductDetails.js==========

```
import React, { useState, useEffect } from "react"; import Layout
from "./../components/Layout/Layout";import axios from "axios";
import  {  useParams,  useNavigate  }  from  "react-router-dom"; import
"../styles/ProductDetailsStyles.css";
```

```
const ProductDetails = () => {
  const params = useParams();
  const navigate = useNavigate();
  const [product, setProduct] = useState({});
  const [relatedProducts, setRelatedProducts] = useState([]);

  //initalp details
  useEffect(() => {
    if (params?.slug) getProduct();
  }, [params?.slug]);
  //getProduct
  const getProduct = async () => {
    try {
      const { data } = await axios.get(
        `/api/v1/product/get-product/${params.slug}`
      );
      setProduct(data?.product);
      getSimilarProduct(data?.product._id, data?.product.category._id);
    } catch (error) {
      console.log(error);
    }
  };
  //get similar product
  const getSimilarProduct = async (pid, cid) => {
    try {
      const { data } = await axios.get(
        `/api/v1/product/related-product/${pid}/${cid}`
      );
      setRelatedProducts(data?.products);
    } catch (error) {
      console.log(error);
    }
  };
  return (
    <Layout>
      <div className="row container product-details">
        <div className="col-md-6">
          <img
            src={`/api/v1/product/product-photo/${product._id}`}
            className="card-img-top"
            alt={product.name}
            height="300"
            width={"350px"}
          />
        </div>
        <div className="col-md-6 product-details-info">
          <h1 className="text-center">Product Details</h1>
          <hr />
          <h6>Name : {product.name}</h6>
          <h6>Description : {product.description}</h6>
          <h6>
            Price :
            {product?.price?.toLocaleString("en-US",  {
              style: "currency",
              currency: "USD",
            })}
          </h6>
          <h6>Category : {product?.category?.name}</h6>
          <button class="btn btn-secondary ms-1">ADD TO CART</button>
        </div>
      </div>
      <hr />
      <div className="row container similar-products">
        <h4>Similar Products 〰</h4>
        {relatedProducts.length < 1 && (
          <p className="text-center">No Similar Products found</p>
        )}
        <div className="d-flex flex-wrap">
          {relatedProducts?.map((p) => (
            <div className="card m-2" key={p._id}>
              <img
                src={`/api/v1/product/product-photo/${p._id}`}
                className="card-img-top"
                alt={p.name}
              />
              <div className="card-body">
                <div className="card-name-price">
                  <h5 className="card-title">{p.name}</h5>
                  <h5 className="card-title card-price">
                    {p.price.toLocaleString("en-US",  {
                      style: "currency",
                      currency: "USD",
                    })}
                  </h5>
                </div>
```

```
                  <p className="card-text ">
                    {p.description.substring(0, 60)}...
                  </p>
                  <div className="card-name-price">
                    <button
                      className="btn btn-info ms-1"
                      onClick={() => navigate(`/product/${p.slug}`)}
                    >
                      More Details
                    </button>
                    {/* <button
                    className="btn btn-dark ms-1"
                    onClick={() => { setCart([...cart,
                    p]); localStorage.setItem(
                        "cart",
                        JSON.stringify([...cart, p])
                      );
                      toast.success("Item Added to cart");
                    }}
                  >
                    ADD TO CART
                  </button> */}
                  </div>
                </div>
              </div>
            ))}
          </div>
        </div>
      </Layout>
    );
};

export default ProductDetails;
```

============ frontend/src/page/Search.js=============

```
import React from "react";
import   Layout   from   "./../components/Layout/Layout";  import   {
useSearch } from "../context/search";
const Search = () => {
    const     [values,     setValues]     =
    useSearch();return (
      <Layout title={"Search results"}>
```

```
              <div className="container">
                <div className="text-center">
                  <h1>Search Resuts</h1>
                  <h6>
                    {values?.results.length < 1
                      ? "No Products Found"
                      : `Found ${values?.results.length}`}
                  </h6>
                  <div className="d-flex flex-wrap mt-4">
                    {values?.results.map((p) => (
                      <div className="card m-2" style={{ width: "18rem" }}>
                        <img
                          src={`/api/v1/product/product-photo/${p._id}`}
                          className="card-img-top"
                          alt={p.name}
                        />
                        <div className="card-body">
                          <h5 className="card-title">{p.name}</h5>
                          <p className="card-text">
                            {p.description.substring(0, 30)}...
                          </p>
                          <p className="card-text"> $ {p.price}</p>
                          <button class="btn btn-primary ms-1">More Details</button>
                          <button class="btn btn-secondary ms-1">ADD TO CART</button>
                        </div>
                      </div>
                    ))}
                  </div>
                </div>
              </div>
            </Layout>
          );
};

export default Search;
```

========frontend/src/page/Admin/AdminDashboard.js========

```
import React from "react";
import AdminMenu from "../../components/Layout/AdminMenu";import Layout
from "./../../components/Layout/Layout";
import   {   useAuth   }   from   "../../context/auth";  const
AdminDashboard = () => {
    const [auth] = useAuth();
    return (
      <Layout>
        <div className="container-fluid m-3 p-3 dashboard">
          <div className="row">
            <div className="col-md-3">
              <AdminMenu />
            </div>
            <div className="col-md-9">
              <div className="card w-75 p-3">
                <h3> Admin Name : {auth?.user?.name}</h3>
                <h3> Admin Email : {auth?.user?.email}</h3>
                <h3> Admin Contact : {auth?.user?.phone}</h3>
              </div>
            </div>
          </div>
        </div>
      </Layout>
    );
};

export default AdminDashboard;
```

========frontend/src/page/Admin/AdminOrders.js========

```
import React, { useState, useEffect } from "react";import axios
from "axios";
import toast from "react-hot-toast";
import AdminMenu from "../../components/Layout/AdminMenu";import Layout
from "../../components/Layout/Layout";
import { useAuth } from "../../context/auth"; import
moment from "moment";
import { Select } from "antd";const { Option
} = Select;

const AdminOrders = () => {
    const [status, setStatus] =
      useState(["Not Process",
      "Processing"
      , "Shipped",
      "deliverd",
      "cancel",
    ]);
    const [changeStatus, setCHangeStatus] = useState("");
    const      [orders,      setOrders]      =
    useState([]); const  [auth,  setAuth]  =
    useAuth(); const getOrders = async ()
    => {
      try {
        const { data } = await axios.get("/api/v1/auth/all-orders");
        setOrders(data);
      }   catch   (error)   {
        console.log(error);
      }
    };

    useEffect(() => {
      if (auth?.token) getOrders();
    }, [auth?.token]);

    const handleChange = async (orderId, value) =>
      {try {
        const { data } = await axios.put(`/api/v1/auth/order-status/${orderId}`, {
          status: value,
        });
        getOrders();
      }   catch   (error)   {
        console.log(error);
      }
    };
    return (
      <Layout title={"All Orders Data"}>
        <div className="row dashboard">
          <div className="col-md-3">
            <AdminMenu />
          </div>
          <div className="col-md-9">
            <h1 className="text-center">All Orders</h1>
            {orders?.map((o, i) => {
              return (
                <div className="border shadow">
                  <table className="table">
                    <thead>
                      <tr>
                        <th scope="col">#</th>
                        <th scope="col">Status</th>
                        <th scope="col">Buyer</th>
                        <th scope="col"> date</th>
                        <th scope="col">Payment</th>
                        <th scope="col">Quantity</th>
                      </tr>
                    </thead>
                    <tbody>
                      <tr>
                        <td>{i + 1}</td>
                        <td>
                          <Select
                            bordered={false}
                            onChange={(value) => handleChange(o._id, value)}
                            defaultValue={o?.status}
                          >
                            {status.map((s, i) => (
                              <Option key={i} value={s}>
                                {s}
                              </Option>
                            ))}
                          </Select>
                        </td>
                        <td>{o?.buyer?.name}</td>
                        <td>{moment(o?.createAt).fromNow()}</td>
                        <td>{o?.payment.success ? "Success" : "Failed"}</td>
                        <td>{o?.products?.length}</td>
                      </tr>
                    </tbody>
                  </table>
                  <div className="container">
                    {o?.products?.map((p, i) => (
                      <div className="row mb-2 p-3 card flex-row" key={p._id}>
                        <div className="col-md-4">
                          <img
                            src={`/api/v1/product/product-photo/${p._id}`}
                            className="card-img-top"
                            alt={p.name}
                            width="100px"
                            height={"100px"}
                          />
                        </div>
                        <div className="col-md-8">
                          <p>{p.name}</p>
                          <p>{p.description.substring(0, 30)}</p>
                          <p>Price : {p.price}</p>
```

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

export default AdminOrders;
```

========frontend/src/page/Admin/CreateCategory.js========

```
import React, { useEffect, useState } from "react"; import Layout
from "./../../components/Layout/Layout";
import AdminMenu from "./../../components/Layout/AdminMenu"; import toast
from "react-hot-toast";
import axios from "axios";
import CategoryForm from "../../components/Form/CategoryForm"; import {
Modal } from "antd";
const CreateCategory = () => {
                    const [categories, setCategories] =
                    useState([]); const [name, setName] =
                    useState("");
                    const [visible, setVisible] = useState(false);
                    const [selected, setSelected] =
                    useState(null);
                    const [updatedName, setUpdatedName] = useState("");
                    //handle Form
                    const handleSubmit = async (e) => {
                      e.preventDefault();
                      try {
                        const { data } = await axios.post("/api/v1/category/create-category", {
                          name,
                        });
                        if (data?.success) {
                          toast.success(`${name} is
                          created`);getAllCategory();
```

```
    } else {
      toast.error(data.message
      );
    }
  } catch (error) {
    console.log(error);
    // toast.error("somthing went wrong in input form");
  }
};

//get all cat
const getAllCategory = async () => {
  try {
    const { data } = await axios.get("/api/v1/category/get-category");
    if (data?.success) {
      setCategories(data?.category);
    }
  } catch (error) {
    console.log(error);
    toast.error("Something wwent wrong in getting catgeory");
  }
};

useEffect(() => {
  getAllCategory();
}, []);

//update category
const handleUpdate = async (e) => {
  e.preventDefault();
  try {
    const { data } = await axios.put(
      `/api/v1/category/update-category/${selected._id}`,
      { name: updatedName }
    );
    if (data?.success) {
      toast.success(`${updatedName} is
      updated`);setSelected(null);
      setUpdatedName("");
      setVisible(false);
      getAllCategory();
    } else {
      toast.error(data.message
      );
    }
  }   catch   (error)   {
    console.log(error);
  }
};
//delete category
const handleDelete = async (pId) => {
  try {
    const { data } = await axios.delete(
      `/api/v1/category/delete-category/${pId}`
    );
    if               (data.success)               {
      toast.success(`category is deleted`);

      getAllCategory();
    }               else               {
      toast.error(data.message
      );
    }
  } catch (error) { toast.error("Somtihing
    went wrong");
  }
};
return (
  <Layout title={"Dashboard - Create Category"}>
    <div className="container-fluid m-3 p-3 dashboard">
      <div className="row">
        <div className="col-md-3">
          <AdminMenu />
        </div>
        <div className="col-md-9">
          <h1>Manage Category</h1>
          <div className="p-3 w-50">
            <CategoryForm
              handleSubmit={handleSubmit}
              value={name}
              setValue={setName}
            />
          </div>
          <div className="w-75">
            <table className="table">
              <thead>
                <tr>
                  <th scope="col">Name</th>
                  <th scope="col">Actions</th>
                </tr>
        </thead>
        <tbody>
          {categories?.map((c) => (
            <>
              <tr>
                <td key={c._id}>{c.name}</td>
                <td>
                  <button
                    className="btn btn-primary ms-2"
                    onClick={() => {
                      setVisible(true);
                      setUpdatedName(c.name
                      );setSelected(c);
                    }}
                  >
                    Edit
                  </button>
                  <button
                    className="btn  btn-danger  ms-
                    2"       onClick={()        =>        {
                    handleDelete(c._id);
                    }}
                  >
                    Delete
                  </button>
                </td>
              </tr>
            </>
          ))}
        </tbody>
      </table>
    </div>
    <Modal
      onCancel={()  =>  setVisible(false)}
      footer={null}
      visible={visible}
    >
      <CategoryForm
        value={updatedName}
        setValue={setUpdatedName}
        handleSubmit={handleUpdate}
      />
    </Modal>
  </div>
</div>
```

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

export default CreateCategory;
```

========frontend/src/page/Admin/CreateProduct.js========

```
import React, { useState, useEffect } from "react"; import Layout
from "./../../components/Layout/Layout";
import AdminMenu from "./../../components/Layout/AdminMenu"; import toast
from "react-hot-toast";
import axios from "axios"; import { Select }
from "antd";
import { useNavigate } from "react-router-dom";const { Option
} = Select;

          const CreateProduct = () => {
            const navigate = useNavigate();
            const [categories, setCategories] =
            useState([]); const [name, setName] =
            useState("");
            const [description, setDescription] = useState("");
            const [price, setPrice] = useState("");
            const [category, setCategory] = useState("");
            const [quantity, setQuantity] = useState("");
            const [shipping, setShipping] = useState("");
            const [photo, setPhoto] = useState("");

            //get all category
            const getAllCategory = async () => {
              try {
                const { data } = await axios.get("/api/v1/category/get-category");
                if (data?.success) {
                  setCategories(data?.category);
                }
              } catch (error) {
                console.log(error);
                toast.error("Something wwent wrong in getting catgeory");
              }
            };
```

```
useEffect(() => {
  getAllCategory();
}, []);
//create product function
const handleCreate = async (e) =>
  {e.preventDefault();
  try {
    const productData = new FormData();
    productData.append("name", name);
    productData.append("description",
    description); productData.append("price",
    price); productData.append("quantity",
    quantity); productData.append("photo", photo);
    productData.append("category", category);
    const { data } = axios.post(
    "/api/v1/product/create-product",
      productData
    );
    if (data?.success) {
      toast.error(data?.message
      );
    } else {
      toast.success("Product Created
      Successfully");
      navigate("/dashboard/admin/products");
    }
  } catch (error) {
    console.log(error);
    toast.error("something went wrong");
  }
};

return (
  <Layout title={"Dashboard - Create Product"}>
    <div className="container-fluid m-3 p-3 dashboard">
      <div className="row">
        <div className="col-md-3">
          <AdminMenu />
        </div>
        <div className="col-md-9">
          <h1>Create Product</h1>
          <div className="m-1 w-75">
            <Select
              bordered={false}
              placeholder="Select a
              category"size="large"
  showSearch
  className="form-select mb-3"
  onChange={(value)      =>      {
  setCategory(value);
  }}
>
  {categories?.map((c) => (
    <Option key={c._id} value={c._id}>
      {c.name}
    </Option>
  ))}
</Select>
<div className="mb-3">
  <label className="btn btn-outline-secondary col-md-12">
    {photo ? photo.name : "Upload Photo"}
    <input
      type="file"
      name="photo"
      accept="image/*"
      onChange={(e) => setPhoto(e.target.files[0])}
      hidden
    />
  </label>
</div>
<div className="mb-3">
  {photo && (
    <div className="text-center">
      <img
        src={URL.createObjectURL(photo)}
        alt="product_photo"
        height={"200px"}
        className="img img-responsive"
      />
    </div>
  )}
</div>
<div className="mb-3">
  <input
    type="text"
    value={name}
    placeholder="write a name"
    className="form-control"
    onChange={(e) => setName(e.target.value)}
  />
```

```
</div>
<div className="mb-3">
  <textarea
    type="text"
    value={description}
    placeholder="write a description"
    className="form-control"
    onChange={(e) => setDescription(e.target.value)}
  />
</div>

<div className="mb-3">
  <input
    type="number"
    value={price}
    placeholder="write a Price"
    className="form-control"
    onChange={(e) => setPrice(e.target.value)}
  />
</div>
<div className="mb-3">
  <input
    type="number"
    value={quantity}
    placeholder="write a quantity"
    className="form-control"
    onChange={(e) => setQuantity(e.target.value)}
  />
</div>
<div className="mb-3">
  <Select
    bordered={false}
    placeholder="Select Shipping "
    size="large"
    showSearch
    className="form-select mb-3"
    onChange={(value) => {
    setShipping(value);
    }}
  >
    <Option value="0">No</Option>
    <Option value="1">Yes</Option>
  </Select>
```

```
                  </div>
                  <div className="mb-3">
                    <button className="btn btn-primary" onClick={handleCreate}>
                      CREATE PRODUCT
                    </button>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </Layout>
      );
};

export default CreateProduct;
```

========frontend/src/page/Admin/Products.js========

```
import React, { useState, useEffect } from "react";
import AdminMenu from "../../components/Layout/AdminMenu";import Layout
from "./../../components/Layout/Layout";
import axios from "axios";
import toast from "react-hot-toast"; import { Link } from
"react-router-dom";const Products = () => {
      const [products, setProducts] = useState([]);

      //getall products
      const getAllProducts = async () => {
        try {
          const { data } = await axios.get("/api/v1/product/get-product");
          setProducts(data.products);
        }   catch   (error)   {
          console.log(error);
          toast.error("Someething Went Wrong");
        }
      };

      //lifecycle     method
      useEffect(()    =>    {
      getAllProducts();
      }, []);
      return (
        <Layout>
```

```
             <div className="row dashboard">
              <div className="col-md-3">
               <AdminMenu />
              </div>
              <div className="col-md-9 ">
               <h1 className="text-center">All Products List</h1>
               <div className="d-flex flex-wrap">
                {products?.map((p) => (
                 <Link
                  key={p._id}
                  to={`/dashboard/admin/product/${p.slug}`}
                  className="product-link"
                 >
                  <div className="card m-2" style={{ width: "18rem" }}>
                   <img
                    src={`/api/v1/product/product-photo/${p._id}`}
                    className="card-img-top"
                    alt={p.name}
                   />
                   <div className="card-body">
                    <h5 className="card-title">{p.name}</h5>
                    <p className="card-text">{p.description}</p>
                   </div>
                  </div>
                 </Link>
                ))}
               </div>
              </div>
             </div>
            </Layout>
           );
};

export default Products;
```

========frontend/src/page/Admin/UpdateProduct.js========

```
import React, { useState, useEffect } from "react"; import Layout
from "./../../components/Layout/Layout";
import AdminMenu from "./../../components/Layout/AdminMenu"; import toast
from "react-hot-toast";
import axios from "axios"; import { Select }
from "antd";
```

```
import { useNavigate, useParams } from "react-router-dom"; const {
Option } = Select;

          const UpdateProduct = () => {
           const navigate =
           useNavigate(); const params =
           useParams();
           const [categories, setCategories] =
           useState([]); const [name, setName] =
           useState("");
           const [description, setDescription] = useState("");
           const [price, setPrice] = useState("");
           const [category, setCategory] = useState("");
           const [quantity, setQuantity] = useState("");
           const [shipping, setShipping] = useState("");
           const [photo, setPhoto] = useState("");
           const [id, setId] = useState("");

           //get single product
           const getSingleProduct = async () => {
            try {
             const { data } = await axios.get(
              `/api/v1/product/get-product/${params.slug}`
             );
             setName(data.product.name);
             setId(data.product._id);
             setDescription(data.product.description)
             ;         setPrice(data.product.price);
             setPrice(data.product.price);
             setQuantity(data.product.quantity);
             setShipping(data.product.shipping);
             setCategory(data.product.category._id);
            } catch (error) {
             console.log(error);
            }
           };
           useEffect(() => {
            getSingleProduct();
            //eslint-disable-next-line
           }, []);
           //get all category
           const getAllCategory = async () => {
            try {
             const { data } = await axios.get("/api/v1/category/get-category");
             if (data?.success) {
```

```
      setCategories(data?.category);
    }
  } catch (error) {
    console.log(error);
    toast.error("Something wwent wrong in getting catgeory");
  }
};

useEffect(() => {
  getAllCategory();
}, []);

//create product function
const handleUpdate = async (e) => {
  e.preventDefault();
  try {
    const productData = new FormData();
    productData.append("name", name);
    productData.append("description", description);
    productData.append("price", price);
    productData.append("quantity", quantity);
    photo && productData.append("photo", photo);
    productData.append("category", category);
    const { data } = axios.put(
      `/api/v1/product/update-product/${id}`,
      productData
    );
    if (data?.success) {
      toast.error(data?.message
      );
    } else {
      toast.success("Product Updated
      Successfully");
      navigate("/dashboard/admin/products");
    }
  } catch (error) {
    console.log(error);
    toast.error("something went wrong");
  }
};

//delete a product
const handleDelete = async () => {
  try {
    let answer = window.prompt("Are You Sure want to delete this product ? ");
    if (!answer) return;
    const { data } = await axios.delete(
      `/api/v1/product/delete-product/${id}`
    );
    toast.success("Product DEleted Succfully");
    navigate("/dashboard/admin/products");
  }   catch   (error)   {
    console.log(error);
    toast.error("Something went wrong");
  }
};
return (
  <Layout title={"Dashboard - Create Product"}>
    <div className="container-fluid m-3 p-3">
      <div className="row">
        <div className="col-md-3">
          <AdminMenu />
        </div>
        <div className="col-md-9">
          <h1>Update Product</h1>
          <div className="m-1 w-75">
            <Select
              bordered={false}
              placeholder="Select                a
              category"size="large"
              showSearch
              className="form-select mb-3"
              onChange={(value)      =>      {
              setCategory(value);
              }}
              value={category}
            >
              {categories?.map((c) => (
                <Option key={c._id} value={c._id}>
                  {c.name}
                </Option>
              ))}
            </Select>
            <div className="mb-3">
              <label className="btn btn-outline-secondary col-md-12">
                {photo ? photo.name : "Upload Photo"}
                <input
                  type="file"
      name="photo"
      accept="image/*"
      onChange={(e) => setPhoto(e.target.files[0])}
      hidden
    />
  </label>
</div>
<div className="mb-3">
  {photo ? (
    <div className="text-center">
      <img
        src={URL.createObjectURL(photo)}
        alt="product_photo"
        height={"200px"}
        className="img img-responsive"
      />
    </div>
  ) : (
    <div className="text-center">
      <img
        src={`/api/v1/product/product-photo/${id}`}
        alt="product_photo"
        height={"200px"} className="img
        img-responsive"
      />
    </div>
  )}
</div>
<div className="mb-3">
  <input
    type="text"
    value={name}
    placeholder="write a name"
    className="form-control"
    onChange={(e) => setName(e.target.value)}
  />
</div>
<div className="mb-3">
  <textarea
    type="text"
    value={description}
    placeholder="write a description"
    className="form-control"
    onChange={(e) => setDescription(e.target.value)}
  />
</div>

<div className="mb-3">
  <input
    type="number"
    value={price}
    placeholder="write a Price"
    className="form-control"
    onChange={(e) => setPrice(e.target.value)}
  />
</div>
<div className="mb-3">
  <input
    type="number"
    value={quantity}
    placeholder="write a quantity"
    className="form-control"
    onChange={(e) => setQuantity(e.target.value)}
  />
</div>
<div className="mb-3">
  <Select
    bordered={false}
    placeholder="Select Shipping "
    size="large"
    showSearch
    className="form-select mb-3"
    onChange={(value)      =>      {
    setShipping(value);
    }}
    value={shipping ? "yes" : "No"}
  >
    <Option value="0">No</Option>
    <Option value="1">Yes</Option>
  </Select>
</div>
<div className="mb-3">
  <button className="btn btn-primary" onClick={handleUpdate}>
    UPDATE PRODUCT
  </button>
</div>
```

```
                 <div className="mb-3">
                  <button className="btn btn-danger" onClick={handleDelete}>
                   DELETE PRODUCT
                  </button>
                 </div>
                </div>
               </div>
              </div>
             </div>
            </Layout>
           );
};

export default UpdateProduct;
```

=============frontend/src/page/Admin/Users.js=============

```
import React from "react";
import AdminMenu from "../../components/Layout/AdminMenu";import Layout
from "./../../components/Layout/Layout";

          const Users = () => {
           return (
            <Layout title={"Dashboard - All Users"}>
             <div className="container-fluid m-3 p-3">
              <div className="row">
               <div className="col-md-3">
                <AdminMenu />
               </div>
               <div className="col-md-9">
                <h1>All Users</h1>
               </div>
              </div>
             </div>
            </Layout>
           );
};

export default Users;
```

========frontend/src/page/Auth/ForgotPassword.js========

```
import React, { useState } from "react";
import Layout from "./../../components/Layout/Layout";
import axios from "axios";
import { useNavigate } from "react-router-dom";import toast
from "react-hot-toast";
import "../../styles/AuthStyles.css";

const ForgotPasssword = () => {
           const [email, setEmail] = useState("");
           const     [newPassword,     setNewPassword]     =
           useState("");const [answer, setAnswer] = useState("");

           const navigate = useNavigate();

           // form function
           const handleSubmit = async (e) => {
             e.preventDefault();
             try {
               const res = await axios.post("/api/v1/auth/forgot-password", {
                 email,
                 newPassword
                 ,answer,
               });
               if     (res     &&     res.data.success)     {
                 toast.success(res.data                  &&
                 res.data.message);

                 navigate("/login");
               }             else             {
                 toast.error(res.data.messag
                 e);
               }
             }   catch   (error)   {
               console.log(error);
               toast.error("Something went wrong");
             }
           };
           return (
             <Layout title={"Forgot Password - Ecommerce APP"}>
               <div className="form-container ">
                 <form onSubmit={handleSubmit}>
                   <h4 className="title">RESET PASSWORD</h4>

                   <div className="mb-3">
```

```
              <input
                type="email"
                value={email}
                onChange={(e) => setEmail(e.target.value)}
                className="form-control"
                id="exampleInputEmail1"
                placeholder="Enter Your Email "
                required
              />
            </div>
            <div className="mb-3">
              <input
                type="text"
                value={answer}
                onChange={(e) => setAnswer(e.target.value)}
                className="form-control"
                id="exampleInputEmail1"
                placeholder="Enter Your favorite Sport Name "
                required
              />
            </div>
            <div className="mb-3">
              <input type="password"
                value={newPassword}
                onChange={(e) =>
                setNewPassword(e.target.value)}
                className="form-control"
                id="exampleInputPassword1"
                placeholder="Enter Your Password"
                required
              />
            </div>

            <button type="submit" className="btn btn-primary">
              RESET
            </button>
          </form>
        </div>
      </Layout>
    );
};

export default ForgotPasssword;
```

============frontend/src/page/Auth/Login.js===============

```
import React, { useState } from "react";
import Layout from "./../../components/Layout/Layout";import axios
from "axios";
import { useNavigate, useLocation } from "react-router-dom";import toast
from "react-hot-toast";
import "../../styles/AuthStyles.css";
import { useAuth } from "../../context/auth";const Login =
() => {
           const [email, setEmail] = useState("");
           const      [password,      setPassword]      =
           useState("");   const    [auth,    setAuth]    =
           useAuth();

           const          navigate          =
           useNavigate();const location =
           useLocation();

           // form function
           const handleSubmit = async (e) => {
             e.preventDefault();
             try {
               const res = await axios.post("/api/v1/auth/login", {
                 email,
                 password,
               });
               if      (res      &&      res.data.success)      {
                 toast.success(res.data                         &&
                 res.data.message);setAuth({
                   ...auth,
                   user:       res.data.user,
                   token: res.data.token,
                 });
                 localStorage.setItem("auth",
                 JSON.stringify(res.data));navigate(location.state || "/");
               }                else                {
                 toast.error(res.data.messag
                 e);
               }
             }   catch   (error)   {
               console.log(error);
               toast.error("Something went wrong");
```

```
  }
};
return (
  <Layout title="Register - Ecommer App">
    <div className="form-container " style={{ minHeight: "90vh" }}>
      <form onSubmit={handleSubmit}>
        <h4 className="title">LOGIN FORM</h4>

        <div className="mb-3">
          <input
            type="email"
            autoFocus
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="form-control"
            id="exampleInputEmail1"
            placeholder="Enter Your Email "
            required
          />
        </div>
        <div className="mb-3">
          <input
            type="password"
            value={password}
            onChange={(e)                                    =>
            setPassword(e.target.value)}
            className="form-control"
            id="exampleInputPassword1"
            placeholder="Enter Your Password"
            required
          />
        </div>
        <div className="mb-3">
          <button
            type="button"
            className="btn      forgot-btn"
            onClick={()            =>            {
            navigate("/forgot-password");
            }}
          >
            Forgot Password
          </button>
        </div>
```

```
                      <button type="submit" className="btn btn-primary">
                        LOGIN
                      </button>
                    </form>
                  </div>
                </Layout>
              );
};

export default Login;
```

===========frontend/src/page/Auth/Register.js==============

```
import React, { useState } from "react";
import Layout from "./../../components/Layout/Layout";import axios
from "axios";
import { useNavigate } from "react-router-dom";import toast
from "react-hot-toast";
import      "../../styles/AuthStyles.css";     const
Register = () => {
              const [name, setName] = useState("");
              const [email, setEmail] = useState("");
              const     [password,     setPassword]     =
              useState(""); const  [phone,  setPhone]  =
              useState("");
              const     [address,     setAddress]     =
              useState("");const [answer, setAnswer] =
              useState("");     const     navigate     =
              useNavigate();

              // form function
              const handleSubmit = async (e) => {
                e.preventDefault();
                try {
                  const res = await axios.post("/api/v1/auth/register", {
                    name,
                    email,
                    password
                    ,   phone,
                    address,
                    answer,
                  });
```

```
    if (res && res.data.success) {
      toast.success(res.data &&
      res.data.message);navigate("/login");
    } else {
      toast.error(res.data.messag
      e);
    }
  } catch (error) {
    console.log(error);
    toast.error("Something went wrong");
  }
};

return (
  <Layout title="Register - Ecommer App">
    <div className="form-container" style={{ minHeight: "90vh" }}>
      <form onSubmit={handleSubmit}>
        <h4 className="title">REGISTER FORM</h4>
        <div className="mb-3">
          <input
            type="text"
            value={name}
            onChange={(e) =>
            setName(e.target.value)}className="form-
            control" id="exampleInputEmail1"
            placeholder="Enter Your Name"
            required
            autoFocu
            s
          />
        </div>
        <div className="mb-3">
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="form-control"
            id="exampleInputEmail1"
            placeholder="Enter Your Email "
            required
          />
        </div>
        <div className="mb-3">
          <input
            type="password"
            value={password}
            onChange={(e)                                  =>
            setPassword(e.target.value)}
            className="form-control"
            id="exampleInputPassword1"
            placeholder="Enter Your Password"
            required
          />
        </div>
        <div className="mb-3">
          <input
            type="text"
            value={phone}
            onChange={(e)                             =>
            setPhone(e.target.value)}
            className="form-control"
            id="exampleInputEmail1"
            placeholder="Enter Your Phone"
            required
          />
        </div>
        <div className="mb-3">
          <input type="text"
            value={address}
            onChange={(e)                                =>
            setAddress(e.target.value)}className="form-
            control" id="exampleInputEmail1"
            placeholder="Enter Your Address"
            required
          />
        </div>
        <div className="mb-3">
          <input
            type="text"
            value={answer}
            onChange={(e) => setAnswer(e.target.value)}
            className="form-control"
            id="exampleInputEmail1"  placeholder="What
            is Your Favorite sports" required
          />
        </div>
        <button type="submit" className="btn btn-primary">
          REGISTER
```

```
                    </button>
                  </form>
                </div>
              </Layout>
            );
};

export default Register;
```

===========frontend/src/page/user/Dashboard.js==============

```
import React from "react";
import Layout from "../../components/Layout/Layout"; import UserMenu
from "../../components/Layout/UserMenu"; import { useAuth } from
"../../context/auth";
          const Dashboard = () => {
            const [auth] = useAuth();
            return (
              <Layout title={"Dashboard - Ecommerce App"}>
                <div className="container-flui m-3 p-3 dashboard">
                  <div className="row">
                    <div className="col-md-3">
                      <UserMenu />
                    </div>
                    <div className="col-md-9">
                      <div className="card w-75 p-3">
                        <h3>{auth?.user?.name}</h3>
                        <h3>{auth?.user?.email}</h3>
                        <h3>{auth?.user?.address}</h3>
                      </div>
                    </div>
                  </div>
                </div>
              </Layout>
            );
};

export default Dashboard;
```

============frontend/src/page/user/Orders.js==============

```
import React, { useState, useEffect } from "react";
import   UserMenu   from   "../../components/Layout/UserMenu";  import
Layout from "./../../components/Layout/Layout"; import axios from "axios";
import  {  useAuth  }  from  "../../context/auth";  import
moment from "moment";

const Orders = () => {
          const      [orders,      setOrders]      =
          useState([]); const  [auth,  setAuth]  =
          useAuth(); const getOrders = async ()
          => {
           try {
            const { data } = await axios.get("/api/v1/auth/orders");
            setOrders(data);
           }  catch  (error)  {
            console.log(error);
           }
          };

          useEffect(() => {
           if (auth?.token) getOrders();
          },  [auth?.token]);
          return (
           <Layout title={"Your Orders"}>
            <div className="container-flui p-3 m-3 dashboard">
             <div className="row">
              <div className="col-md-3">
               <UserMenu />
              </div>
              <div className="col-md-9">
               <h1 className="text-center">All Orders</h1>
               {orders?.map((o, i) => {
                return (
                 <div className="border shadow">
                  <table className="table">
                   <thead>
                    <tr>
                     <th scope="col">#</th>
                     <th scope="col">Status</th>
                     <th scope="col">Buyer</th>
                     <th scope="col"> date</th>
                     <th scope="col">Payment</th>
                     <th scope="col">Quantity</th>
                    </tr>
                    </thead>
                    <tbody>
                      <tr>
                        <td>{i + 1}</td>
                        <td>{o?.status}</td>
                        <td>{o?.buyer?.name}</td>
                        <td>{moment(o?.createAt).fromNow()}</td>
                        <td>{o?.payment.success ? "Success" : "Failed"}</td>
                        <td>{o?.products?.length}</td>
                      </tr>
                    </tbody>
                  </table>
                  <div className="container">
                    {o?.products?.map((p, i) => (
                      <div className="row mb-2 p-3 card flex-row" key={p._id}>
                        <div className="col-md-4">
                          <img
                            src={`/api/v1/product/product-photo/${p._id}`}
                            className="card-img-top"
                            alt={p.name}
                            width="100px"
                            height={"100px"}
                          />
                        </div>
                        <div className="col-md-8">
                          <p>{p.name}</p>
                          <p>{p.description.substring(0, 30)}</p>
                          <p>Price : {p.price}</p>
                        </div>
                      </div>
                    ))}
                  </div>
                </div>
              );
            })}
          </div>
        </div>
      </div>
    </Layout>
  );
};

export default Orders;
```

============frontend/src/page/user/Profile.js=========== ====

```
import React, { useState, useEffect } from "react";
import UserMenu from "../../components/Layout/UserMenu"; import
Layout from "./../../components/Layout/Layout"; import { useAuth } from
"../../context/auth";
import toast from "react-hot-toast";import axios
from "axios";
const Profile = () => {
          //context
          const [auth, setAuth] = useAuth();
          //state
          const [name, setName] = useState("");
          const [email, setEmail] = useState("");
          const [password, setPassword] =
          useState(""); const [phone, setPhone] =
          useState("");
          const [address, setAddress] = useState("");

          //get user data
          useEffect(() => {
           const { email, name, phone, address } =
           auth?.user;setName(name);
           setPhone(phone);
           setEmail(email);
           setAddress(address
           );
          }, [auth?.user]);

          // form function
          const handleSubmit = async (e) => {
           e.preventDefault();
           try {
            const { data } = await axios.put("/api/v1/auth/profile", {
             name,
             email,
             password
             , phone,
             address,
            });
            if (data?.errro) {
             toast.error(data?.error)
             ;
            } else {
```

```
      setAuth({ ...auth, user: data?.updatedUser });
      let ls = localStorage.getItem("auth");
      ls = JSON.parse(ls);
      ls.user =
      data.updatedUser;
      localStorage.setItem("auth", JSON.stringify(ls));
      toast.success("Profile Updated Successfully");
    }
  } catch (error) {
    console.log(error);
    toast.error("Something went wrong");
  }
};
return (
  <Layout title={"Your Profile"}>
    <div className="container-fluid m-3 p-3 dashboard">
      <div className="row">
        <div className="col-md-3">
          <UserMenu />
        </div>
        <div className="col-md-8">
          <div className="form-container" style={{ marginTop: "-40px" }}>
            <form onSubmit={handleSubmit}>
              <h4 className="title">USER PROFILE</h4>
              <div className="mb-3">
                <input
                  type="text"
                  value={name}
                  onChange={(e) =>
                  setName(e.target.value)} className="form-
                  control" id="exampleInputEmail1"
                  placeholder="Enter Your Name"
                  autoFocus
                />
              </div>
              <div className="mb-3">
                <input
                  type="email"
                  value={email}
                  onChange={(e) => setEmail(e.target.value)}
                  className="form-control"
                  id="exampleInputEmail1"
                  placeholder="Enter Your Email "
                  disabled
                />
```

```
                  </div>
};                <div className="mb-3">
                   <input
                    type="password"
                    value={password}
                    onChange={(e) =>
                    setPassword(e.target.value)}
                    className="form-control"
                    id="exampleInputPassword1"
                    placeholder="Enter Your Password"
                   />
                  </div>
                  <div className="mb-3">
                   <input
                    type="text"
                    value={phone}
                    onChange={(e) =>
                    setPhone(e.target.value)}
                    className="form-control"
                    id="exampleInputEmail1"
                    placeholder="Enter Your Phone"
                   />
                  </div>
                  <div className="mb-3">
                   <input type="text"
                    value={address}
                    onChange={(e) =>
                    setAddress(e.target.value)}className="form-
                    control" id="exampleInputEmail1"
                    placeholder="Enter Your Address"
                   />
                  </div>

                  <button type="submit" className="btn btn-primary">
                   UPDATE
                  </button>
                 </form>
                </div>
               </div>
              </div>
             </div>
            </Layout>
           );
};
```

```
export default Profile;
```

========frontend/src/hooks/UseCategory.js============

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

export default function useCategory() {
                const [categories, setCategories] = useState([]);

                //get cat
                const getCategories = async () => {
                  try {
                    const { data } = await axios.get("/api/v1/category/get-category");
                    setCategories(data?.category);
                  }   catch   (error)   {
                    console.log(error);
                  }
                };

                useEffect(()  =>  {
                  getCategories()
                  ;
                }, []);

                return categories;
}
```

===============frontend/src/context/auth.js================

```
import { useState, useEffect, useContext, createContext } from "react";import axios
from "axios";

const     AuthContext     =     createContext();     const
AuthProvider = ({ children }) => {const [auth, setAuth]
= useState({
                  user:  null,
                  token: "",
                });

                //default axios
                axios.defaults.headers.common["Authorization"] = auth?.token;

                useEffect(() => {
```

```
          const data = localStorage.getItem("auth");
          if (data) {
            const parseData = JSON.parse(data);
            setAuth({
              ...auth,
              user:      parseData.user,
              token: parseData.token,
            });
          }
          //eslint-disable-next-line
        }, []);
        return (
          <AuthContext.Provider value={[auth, setAuth]}>
            {children}
          </AuthContext.Provider>
        );
};

// custom hook
const  useAuth  =  ()  =>  useContext(AuthContext); export  {

useAuth, AuthProvider };
```

===============frontend/src/context/cart.js================

```
import { useState, useContext, createContext, useEffect } from "react";

const    CartContext    =    createContext();    const
CartProvider = ({ children }) => {const [cart, setCart]
= useState([]);

        useEffect(() => {
          let existingCartItem = localStorage.getItem("cart");
          if (existingCartItem) setCart(JSON.parse(existingCartItem));
        }, []);

        return (
          <CartContext.Provider value={[cart, setCart]}>
            {children}
          </CartContext.Provider>
        );
};

// custom hook
const useCart = () => useContext(CartContext);
```

```
export { useCart, CartProvider };
```

=============frontend/src/context/search.js================

```
import { useState, useContext, createContext } from "react";

const    SearchContext    =    createContext();    const
SearchProvider  =  ({  children  })  =>  { const  [auth,
setAuth] = useState({
                  keyword: "",
                  results: [],
                });

                return (
                  <SearchContext.Provider value={[auth, setAuth]}>
                    {children}
                  </SearchContext.Provider>
                );
};

// custom hook
const  useSearch  =  ()  =>  useContext(SearchContext); export  {

useSearch, SearchProvider };
```

=======frontend/src/components/Form/CategoryForm.js========

```
import React from "react";

              const CategoryForm = ({ handleSubmit, value, setValue }) => {
                return (
                  <>
                    <form onSubmit={handleSubmit}>
                      <div className="mb-3">
                        <input
                          type="text"
                          className="form-control"
                          placeholder="Enter new category"
                          value={value}
                          onChange={(e) => setValue(e.target.value)}
                        />
                      </div>
```

```
          <button type="submit" className="btn btn-primary">
            Submit
          </button>
        </form>
      </>
    );
};

export default CategoryForm;
```

=======frontend/src/components/Form/SearchInput.js========

```
import React from "react";
import { useSearch } from "../../context/search";import axios
from "axios";
import   {   useNavigate   }   from   "react-router-dom";  const
SearchInput = () => {
    const      [values,      setValues]      =
    useSearch();    const     navigate     =
    useNavigate();

    const handleSubmit = async (e) => {
      e.preventDefault();
      try {
        const { data } = await axios.get(
          `/api/v1/product/search/${values.keyword}`
        );
        setValues({ ...values, results: data });
        navigate("/search");
      }   catch   (error)   {
        console.log(error);
      }
    };
    return (
      <div>
        <form
          className="d-flex search-form"
          role="search"
          onSubmit={handleSubmit}
        >
          <input
            className="form-control me-2"
            type="search"
            placeholder="Search"
            aria-label="Search"
            value={values.keyword}
            onChange={(e) => setValues({ ...values, keyword: e.target.value })}
          />
          <button className="btn btn-outline-success" type="submit">
            Search
          </button>
        </form>
      </div>
    );
};

export default SearchInput;
```

=======frontend/src/components/Layout/AdminMenu.js========

```
import React from "react";
import { NavLink } from "react-router-dom"; const
AdminMenu = () => {
  return (
    <>
      <div className="text-center">
        <div className="list-group dashboard-menu">
          <h4>Admin Panel</h4>
          <NavLink
            to="/dashboard/admin/create-category"
            className="list-group-item list-group-item-action"
          >
            Create Category
          </NavLink>
          <NavLink
            to="/dashboard/admin/create-
            product"
            className="list-group-item list-group-item-action"
          >
            Create Product
          </NavLink>
          <NavLink
            to="/dashboard/admin/products
            "
            className="list-group-item list-group-item-action"
          >
            Products
          </NavLink>
```

```
                    <NavLink
                      to="/dashboard/admin/orders
                      "
                      className="list-group-item list-group-item-action"
                    >
                      Orders
                    </NavLink>
                    {/*                    <NavLink
                      to="/dashboard/admin/users
                      "
                      className="list-group-item list-group-item-action"
                    >
                      Users
                    </NavLink> */}
                  </div>
                </div>
              </>
            );
};

export default AdminMenu;
```

=======frontend/src/components/Layout/Footer.js========

```
import React from "react";
import { Link } from "react-router-dom";const Footer =
() => {
            return (
              <div className="footer">
                <h1 className="text-center">All Right Reserved &copy; Techinfoyt</h1>
                <p className="text-center mt-3">
                  <Link to="/about">About</Link>|<Link to="/contact">Contact</Link>|
                  <Link to="/policy">Privacy Policy</Link>
                </p>
              </div>
            );
};

export default Footer;
```

=======frontend/src/components/Layout/Header.js========

```
import React from "react";
import { NavLink, Link } from "react-router-dom"; import {
useAuth } from "../../context/auth"; import toast from "react-hot-
toast";
import SearchInput from "../Form/SearchInput"; import
useCategory from "../../hooks/useCategory";import { useCart }
from "../../context/cart";
import { Badge } from "antd";
const Header = () => {
           const [auth, setAuth] = useAuth();
           const [cart] = useCart();
           const      categories      =
           useCategory();          const
           handleLogout = () => { setAuth({
             ...auth,
             user: null,
             token: "",
            });
            localStorage.removeItem("auth");
            toast.success("Logout
            Successfully");
           };
           return (
            <>
             <nav className="navbar navbar-expand-lg bg-body-tertiary fixed-top">
              <div className="container-fluid">
               <button
                className="navbar-
                toggler"type="button"
                data-bs-toggle="collapse"
                data-bs-
                target="#navbarTogglerDemo01"   aria-
                controls="navbarTogglerDemo01"  aria-
                expanded="false"
                aria-label="Toggle navigation"
               >
                <span className="navbar-toggler-icon" />
               </button>
               <div className="collapse navbar-collapse" id="navbarTogglerDemo01">
                <Link to="/" className="navbar-brand>
               SHOPPING CART
                </Link>
                <ul className="navbar-nav ms-auto mb-2 mb-lg-0">
                 <SearchInput />
                 <li className="nav-item">
                  <NavLink to="/" className="nav-link ">
                   Home
```

```
  </NavLink>
</li>
<li className="nav-item dropdown">
  <Link
    className="nav-link dropdown-toggle"
    to={"/categories"}
    data-bs-toggle="dropdown"
  >
    Categories
  </Link>
  <ul className="dropdown-menu">
    <li>
      <Link className="dropdown-item" to={"/categories"}>
        All Categories
      </Link>
    </li>
    {categories?.map((c) => (
      <li>
        <Link
          className="dropdown-item"
          to={`/category/${c.slug}`}
        >
          {c.name}
        </Link>
      </li>
    ))}
  </ul>
</li>

{!auth?.user ? (
  <>
    <li className="nav-item">
      <NavLink to="/register" className="nav-link">
        Register
      </NavLink>
    </li>
    <li className="nav-item">
      <NavLink to="/login" className="nav-link">
        Login
      </NavLink>
    </li>
  </>
) : (
  <>
    <li className="nav-item dropdown">
```

```
                <NavLink
                  className="nav-link  dropdown-toggle"
                  href="#"
                  role="button"
                  data-bs-toggle="dropdown"
                  style={{ border: "none" }}
                >
                  {auth?.user?.name}
                </NavLink>
                <ul className="dropdown-menu">
                  <li>
                    <NavLink
                      to={`/dashboard/${
                        auth?.user?.role === 1 ? "admin" : "user"
                      }`}
                      className="dropdown-item"
                    >
                      Dashboard
                    </NavLink>
                  </li>
                  <li>
                    <NavLink
                      onClick={handleLogout}
                      to="/login"
                      className="dropdown-item"
                    >
                      Logout
                    </NavLink>
                  </li>
                </ul>
              </li>
            </>
          )}
          <li className="nav-item">
            <NavLink to="/cart" className="nav-link">
              <Badge  count={cart?.length}  showZero  offset={[10,  -5]}>
                Cart
              </Badge>
            </NavLink>
          </li>
        </ul>
      </div>
    </div>
  </nav>
</>
```

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

=======frontend/src/components/Layout/Layout.js========

```
import React from "react"; import Footer from
"./Footer"; import Header from "./Header";
import { Helmet } from "react-helmet"; import { Toaster
} from "react-hot-toast";
         const Layout = ({ children, title, description, keywords, author }) => {
          return (
            <div>
              <Helmet>
                <meta charSet="utf-8" />
                <meta name="description" content={description} />
                <meta name="keywords" content={keywords} />
                <meta name="author" content={author} />
                <title>{title}</title>
              </Helmet>
              <Header />
              <main style={{ minHeight: "70vh" }}>
                <Toaster />

                {children}
              </main>
              <Footer />
            </div>
          );
};

Layout.defaultProps = {
          title: "Ecommerce app - shop now",
          description: "mern stack project",
          keywords:
          "mern,react,node,mongodb",author: "",
};

export default Layout;
```

=======frontend/src/components/Layout/UserMenu.js========

```
import React from "react";
import { NavLink } from "react-router-dom";
          const UserMenu = () => {
           return (
            <div>
             <div className="text-center dashboard-menu">
              <div className="list-group">
               <h4>Dashboard</h4>
               <NavLink
                to="/dashboard/user/profile"
                className="list-group-item list-group-item-action"
               >
                Profile
               </NavLink>
               <NavLink
                to="/dashboard/user/orders
                "
                className="list-group-item list-group-item-action"
               >
                Orders
               </NavLink>
              </div>
             </div>
            </div>
           );
};

export default UserMenu;
```

=======frontend/src/components/ Prices.js========

```
export const Prices = [
           {
            _id: 0,
            name: "$0 to 19",
            array: [0, 19],
           },
           {
            _id: 1,
            name: "$20 to 39",
            array: [20, 39],
           },
           {
            _id: 2,
            name: "$40 to 59",
            array: [40, 59],
           },
          {
            _id: 3,
            name: "$60 to 79",
            array: [60, 79],
          },
          {
            _id: 4,
            name: "$80 to 99",
            array: [80, 99],
          },
          {
            _id: 4,
            name: "$100 or more",
            array: [100, 9999],
          },
];
```

=======frontend/src/components/ Spinner.js========

```
import React, { useState, useEffect } from "react";
import { useNavigate, useLocation } from "react-router-dom";const Spinner
= ({ path = "login" }) => {
          const      [count,      setCount]      =
          useState(3);   const     navigate     =
          useNavigate();
          const location = useLocation();

          useEffect(() => {
            const    interval    =    setInterval(()    =>    {
              setCount((prevValue) => --prevValue);
            }, 1000);
            count       ===       0       &&
              navigate(`/${path}`,         {
              state: location.pathname,
              });
            return () => clearInterval(interval);
          }, [count, navigate, location, path]);
          return (
            <>
              <div
                className="d-flex flex-column justify-content-center align-items-center"
                style={{ height: "100vh" }}
              >
                <h1 className="Text-center">redirecting to you in {count} second </h1>
                <div className="spinner-border" role="status">
```

```
                   <span className="visually-hidden">Loading...</span>
                  </div>
                 </div>
                </>
               );
};

export default Spinner;
```

=======frontend/src/components/Route/AdminRoute.js========

```
import { useState, useEffect } from "react"; import {
useAuth } from "../../context/auth";import { Outlet } from
"react-router-dom"; import axios from "axios";
import Spinner from "../Spinner";

          export default function PrivateRoute() {
           const [ok, setOk] = useState(false);
           const [auth, setAuth] = useAuth();

           useEffect(() => {
            const authCheck = async () => {
             const res = await axios.get("/api/v1/auth/admin-auth");
             if (res.data.ok) {
              setOk(true);
             } else {
              setOk(false);
             }
            };
            if (auth?.token) authCheck();
           }, [auth?.token]);

           return ok ? <Outlet /> : <Spinner path="" />;
}
```

=======frontend/src/components/Route/Private.js========

```
import { useState, useEffect } from "react"; import {
useAuth } from "../../context/auth";import { Outlet } from
"react-router-dom"; import axios from "axios";
import Spinner from "../Spinner"; export default

function PrivateRoute() {
        const [ok, setOk] = useState(false);
        const [auth, setAuth] = useAuth();

        useEffect(() => {
          const authCheck = async () => {
            const res = await axios.get("/api/v1/auth/user-auth");
            if (res.data.ok) {
              setOk(true);
            }      else      {
              setOk(false);
            }
          };
          if (auth?.token) authCheck();
        }, [auth?.token]);

        return ok ? <Outlet /> : <Spinner />;
}
```

===================frontend/src/index.css================@import url("https://fonts.googleapis.com/css2?family=Playfair+Display:wght@700&family
=Poppins:wght@300;400&display=swap");

```
* {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
}

/* font-family: 'Poppins', sans-serif; */
/* font-family: 'Playfair Display', serif; */

/* //navbar css */
.navbar {
        font-family: "Poppins", sans-serif;
        font-size: 17px;
        line-height: 26px;
        text-transform: uppercase;
        box-shadow: 0 8px 6px -6px gray;
        --webkit-box-shadow: 0 8px 6px -6px gray;
        border-bottom: solid gray !important;
}
```

```
.nav-link {
            font-weight: 300 !important;
}

.active {
            border-bottom: 2px solid black;
}

           .navbar-brand    {
            font-weight: 700;
            font-family:  "roboto",  sans-serif;
            letter-spacing: 3px;
}

           .search-form        {
            margin-right:  10px;
            margin-top: 4px;
}
           .search-form input {
            border:       none;
            border-radius: 0;
}
           .search-form     button     {
            background-color: #000000;
            border-radius: 0;
            color: white;
}

/* =============================
==========footer============= */

           .footer { color:
            white;
            padding: 25px;
            background: #000000; /* fallback for old browsers */
            background: -webkit-linear-gradient(
             to   right,
             #434343
             ,
             #000000
            ); /* Chrome 10-25, Safari 5.1-6 */
            background: linear-gradient(to right, #434343, #000000);
}
```

```
.footer a {
                text-decoration:
                none;  color:     white;
                padding: 10px;
}

               .footer a:hover {
                color: #ffefba;
                border-bottom: 1px solid #ffefba;
}
/* ==================================
======== page not found css ======= */
.pnf {
                display: flex;
                min-height: 65vh;
                flex-direction:  column;
                align-items:      center;
                justify-content: center;
}

.pnf-title {
                font-size:  100px;
                font-weight: 700;
}

.pnf-heading {
                font-weight: normal;
}

               .pnf-btn       {
                color: black;
                border: 1px solid black;
                text-decoration:   none;
                padding: 10px;
                margin-top: 10px;
}
               .pnf-btn:hover                 {
                background-color: black;
                color: white;
}
/* ========================================= */
/* =========contact us ========= */
```

```
          .contactus {
           margin: 0;
           padding:    0;
           height:  70vh;
           display: flex;
           align-items:      center;
           justify-content: center;
}

/* ==================== */
.product-link {
           text-decoration: none !important;
           color: black !important;
}

.cat-btn {
           padding: 40px 0px;
           font-size: 24px;
           text-transform: uppercase;
}
.cat-btn:hover {
           background-color: #434343;
           color: white;
}
/* ================
====dashboard
=================== */

.dashboard {
           margin-top: 100px !important;
}

.dashboard-menu h4 {
           background-color: #434343 !important;
           color: white;
           padding: 20px 0px;
}
```

=============frontend/src/styles/AuthStyles.css=============

```
/* ===================================
========= register page  */
          .form-container    {
           margin-top: 60px;
```

```
            display: flex;
            align-items:    center;
            justify-content: center;
            min-height: 70vh;
            flex-direction:     column;
            background-color: #ffdee9;
            background-image: linear-gradient(0deg, #ffdee9 0%, #b5fffc 100%);
}

.form-container form {
            box-shadow: rgba(0, 0, 0, 0.35) 0px 5px 15px;
            padding: 20px;
            background-color: #fff;
}

           .form-container form .title {
            text-align: center;
            margin-bottom: 15px;
            font-weight: bold;
            font-family: "Playfair Display", serif;
}

           .form-container form input {
            border: none;
            border-bottom: 1px solid #000;
            border-radius: 0;
}

           .form-container form button {
            border:   1px   solid   black;
            border-radius: 0;
            background-color: #000;
            color: white;
            width: 250px;
}

           .form-container form button:hover {
            border: 1px solid black;
            background: #000000; /* fallback for old browsers */
            background: -webkit-linear-gradient(
             to   right,
             #434343
             ,
             #000000
```

```
          ); /* Chrome 10-25, Safari 5.1-6 */
          background: linear-gradient(
           to    right,
           #434343
           ,
           #000000
          ); /* W3C, IE 10+/ Edge, Firefox 16+, Chrome 26+, Opera 12+, Safari 7+ */
}

.forgot-btn {
          background-color: #434343a4 !important;
          border: 0 !important;
}

         .forgot-btn:hover  { color:
          white !important;
}
```

===========frontend/src/styles/CartStyle.css================

```
         .cart-page           {
          margin-top: 64px;
}

.cart-page h1 {
          padding: 15px !important;
          font-family:  "roboto",  sans-serif;
          font-weight: normal;
          background-color: rgba(0, 0, 255, 0.072) !important;
}
         .cart-page h1 > p {
          font-size:     20px;
          margin-top: 10px;
}

         .cart-page       .card       {
          padding: 5px !important;
          height:      150px      !important;
          margin-bottom: 5px !important;
}
         .cart-page .cart-remove-btn {
          display: flex;
          align-items:      center;
          justify-content: center;
}
         .cart-page .cart-summary {
          text-align: center;
          /* margin-top: -140px !important; */
}
```

========frontend/src/styles/CategoryProductStyle.css==========

```
.category {
             flex-wrap: wrap;
             margin-top: 80px !important;
}
.category .card {
             background-color: rgba(128, 128, 128, 0.097);
             width: 18rem;
}
            .category .card-name-price {
             display: flex;
             flex-direction: row;
             justify-content: space-between;
}

            .category .card-price {
             color: green;
             font-weight: bold;
}
            .category button {
             border-radius: 0;
             width: 100%;
             border-top-left-radius:      5px;
             border-bottom-right-radius:  5px;
             font-size: 14px;
             font-weight: bold;
}

            .category   .card-text   {
             color: rgb(90, 89, 89);
}

            .category .card img:hover {
             transform: scale(0.9);
}
```

```
.category .card-img-top {
            height: 300px !important;
}
```

==============frontend/src/styles/homePage===============

```
           .banner-img        {
            margin-top: 60px;
}

           .home-page h1 {
            color: gray;
            font-family: "Playfair Display", serif;
}

           .home-page      {
            display:      flex;
            flex-wrap: wrap;
}
.home-page .card {
            background-color: rgba(128, 128, 128, 0.097);
            width: 18rem;
}
           .home-page  .card-name-price
            {display: flex;
            flex-direction: row;
            justify-content: space-between;
}

           .home-page .card-price {
            color: green;
            font-weight: bold;
}
           .home-page button {
            border-radius: 0;
            width: 100%;
            border-top-left-radius:        5px;
            border-bottom-right-radius: 5px;
            font-size: 14px;
            font-weight: bold;
}
```

```
                  .home-page .card-text {
                    color: rgb(90, 89, 89);
}

                  .home-page .card img:hover {
                    transform: scale(0.9);
}

                  .home-page .card-img-top {
                    height: 300px !important;
}
/* ==================
filters
============== */
.filters h4 {
                    margin-top:        100px;
                    color: gray !important;
                    border-bottom: 1px solid rgba(128, 128, 128, 0.337);
}

                  .ant-checkbox-wrapper:first-of-type  {
                    margin-left: 8px;
}

                  .ant-checkbox-wrapper {
                    margin: 2px;
}
.ant-radio-wrapper {
                    margin-bottom: 5px !important;
                    margin-left: 10px;
}

                  .filters          button          {
                    background-color: black;
                    width: 100%;
                    border:        none;
                    border-radius:   0;
                    margin-top: 20px;
}

                  .loadmore    {
                    color: green;
                    font-weight: bold;
                    font-size: 20px !important;
}
```

==========frontend/src/styles/ProductDetailsStyle============

```
.product-details {
            margin-top: 100px !important;
}

           .product-details-info  {
            display: flex;
            flex-direction: column;
            /* align-items: center; */
            justify-content: center;
}

.product-details-info h6 {
            margin-bottom: 20px !important;
}

.similar-products .card {
            background-color: rgba(128, 128, 128, 0.097);
            width: 18rem;
}

           .similar-products .card-name-price {
            display: flex;
            flex-direction: row;
            justify-content: space-between;
}

           .similar-products .card-price {
            color: green;
            font-weight: bold;
}
           .similar-products button {
            border-radius: 0;
            width: 100%;
            border-top-left-radius:      5px;
            border-bottom-right-radius: 5px;
            font-size: 14px;
            font-weight: bold;
}

           .similar-products .card-text {
            color: rgb(90, 89, 89);
}

          .similar-products .card img:hover {
           transform: scale(0.9);
}

          .similar-products .card-img-top {
           height: 300px !important;
}
```

==============frontend/public/index.html ================

```
<!DOCTYPE html>
<html lang="en">
           <head>
            <meta charset="utf-8" />
            <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
            <meta name="viewport" content="width=device-width, initial-scale=1" />
            <meta name="theme-color" content="#000000" />
            <meta
             name="description"
             content="Web site created using create-react-app"
            />
            <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
            <!--
             manifest.json provides metadata used when your web app is installed on
             auser's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
            -->
            <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
            <link
             href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
          alpha1/dist/css/bootstrap.min.css
             "              rel="stylesheet"
             integrity="sha384-
GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
             crossorigin="anonymous"
            />
            <!--
             Notice the use of %PUBLIC_URL% in the tags above.
             It will be replaced with the URL of the `public` folder during the build.
             Only files inside the `public` folder can be referenced from the HTML.
```

```
        Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" willwork
        correctly both with client-side routing and a non-root public URL. Learn how
        to configure a non-root public URL by running `npm run build`.
      -->
      <title>Ecommerce App</title>
    </head>
    <body>
      <noscript>You need to enable JavaScript to run this app.</noscript>
      <div id="root"></div>
      <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0
        -
  alpha1/dist/js/bootstrap.bundle.min.js
        "integrity="sha384-
w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN"
        crossorigin="anonymous"
      ></script>
      <!--
        This HTML file is a template.
        If you open it directly in the browser, you will see an empty page.

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

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

===============frontend/src/style/index.js================

```
import React from "react";
import ReactDOM from "react-dom/client"; import
"./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals"; import {
BrowserRouter } from "react-router-dom";import { AuthProvider
} from "./context/auth"; import { SearchProvider } from
"./context/search";
```

```
import { CartProvider } from "./context/cart"; import
"antd/dist/reset.css";

const root = ReactDOM.createRoot(document.getElementById("root"));root.render(
           <AuthProvider>
             <SearchProvider>
               <CartProvider>
                 <BrowserRouter>
                   <App />
                 </BrowserRouter>
               </CartProvider>
             </SearchProvider>
           </AuthProvider>
);

reportWebVitals();
```

~~~~~~~~~~~~~~~~~~~~~~Backend Codes~~~~~~~~~~~~~~~~~~~~~~~~

================== database.js=======================

```
import mongoose from "mongoose";

          const connectDB = async () => {
            try {
              const conn = await
              mongoose.connect(process.env.MONGO_URL);console.log(
                `Connected to Mongodb Database ${conn.connection.host}`.bgBlack.yellow
              );
            } catch (error) {
              console.log(`Error in Mongodb ${error}`.bgBlack.red);
            }
};

export default connectDB;
```

====================
server.js======================

```
import express from "express";import colors
from "colors"; import dotenv from "dotenv";
import morgan from "morgan";
import connectDB from "./config/db.js";
import authRoutes from "./routes/authRoute.js";
import categoryRoutes from "./routes/categoryRoutes.js"; import
productRoutes from "./routes/productRoutes.js"; import cors from
"cors";

//configure                    env
dotenv.config();

//databse                  config
connectDB();

//rest object
const app = express();

//middelwares          app.use(cors());
app.use(express.json());
app.use(morgan("dev"));

//routes
app.use("/api/v1/auth",                              authRoutes);
app.use("/api/v1/category",                 categoryRoutes);
app.use("/api/v1/product", productRoutes);

//rest api
                 app.get("/",        (req,        res)        =>       {
                   res.send("<h1>Welcome    to    SHOPPING
                   CART</h1>");
});

//PORT
const PORT = process.env.PORT || 8080;

//run listen app.listen(PORT, () => {
console.log(
                     `Server Running on ${process.env.DEV_MODE} mode on port
${PORT}`.bgCyan
```

```
            .white
          );
});
```

================ routes/authRoute.js=====================

```
import express from "express";import {
          registerController,
          loginController,
          testController,
          forgotPasswordController,
          updateProfileController,
          getOrdersController,
          getAllOrdersController,
          orderStatusController,
} from "../controllers/authController.js";
import { isAdmin, requireSignIn } from "../middlewares/authMiddleware.js";

//router object
const router = express.Router();

//routing
//REGISTER || METHOD POST
router.post("/register", registerController);

//LOGIN     ||     POST     router.post("/login",
loginController);

//Forgot Password || POST
router.post("/forgot-password", forgotPasswordController);

//test routes
router.get("/test", requireSignIn, isAdmin, testController);

//protected User route auth
         router.get("/user-auth", requireSignIn, (req, res) => {
          res.status(200).send({ ok: true });
});
//protected Admin route auth
         router.get("/admin-auth", requireSignIn, isAdmin, (req, res) => {
          res.status(200).send({ ok: true });
});

//update profile
router.put("/profile", requireSignIn, updateProfileController);

//orders
router.get("/orders", requireSignIn, getOrdersController);

//all orders
router.get("/all-orders", requireSignIn, isAdmin, getAllOrdersController);

// order status update router.put("/order-
status/:orderId",requireSignIn,
            isAdmin,
            orderStatusControlle
            r
);

export default router;
```

=================
routes/categoryRoutes.js================

```
import express from "express";
import { isAdmin, requireSignIn } from "./../middlewares/authMiddleware.js";import {
            categoryControlller,
            createCategoryController,
            deleteCategoryCOntroller,
            singleCategoryController,
            updateCategoryController
            ,
} from "./../controllers/categoryController.js";const router =

express.Router();

//routes
// create categoryrouter.post(
            "/create-category",
            requireSignIn,
            isAdmin,
            createCategoryController
);

//update category
```

```
router.put(
        "/update-category/:id",
        requireSignIn,
        isAdmin,
        updateCategoryController
);

//getALl category
router.get("/get-category", categoryControlller);

//single category
router.get("/single-category/:slug", singleCategoryController);

//delete          category
router.delete(
        "/delete-category/:id",
        requireSignIn,
        isAdmin,
        deleteCategoryCOntroller
);

export default router;
```

=================
routes/productRoutes.js==================

```
import express from "express"; import {
brainTreePaymentController,
braintreeTokenController,
createProductController,
deleteProductController,
getProductController,
getSingleProductController,
productCategoryController,
productCountController,
productFiltersController,
productListController,
productPhotoController,
realtedProductController,
searchProductController,
updateProductController,
} from "../controllers/productController.js";
import { isAdmin, requireSignIn } from "../middlewares/authMiddleware.js";
import formidable from "express-formidable";const router

= express.Router();

//routes   router.post(   "/create-
product",            requireSignIn,
isAdmin, formidable(),
                  createProductController
);
//routes router.put(
                  "/update-product/:pid",
                  requireSignIn,
                  isAdmin, formidable(),
                  updateProductController
);

//get products
router.get("/get-product", getProductController);

//single product
router.get("/get-product/:slug", getSingleProductController);

//get photo
router.get("/product-photo/:pid", productPhotoController);

//delete rproduct
router.delete("/delete-product/:pid", deleteProductController);

//filter product
router.post("/product-filters", productFiltersController);

//product count
router.get("/product-count", productCountController);

//product per page
router.get("/product-list/:page", productListController);

//search product
```

```
router.get("/search/:keyword", searchProductController);

//similar product
router.get("/related-product/:pid/:cid", realtedProductController);

//category wise product
router.get("/product-category/:slug", productCategoryController);

//payments routes
//token
router.get("/braintree/token", braintreeTokenController);

//payments
router.post("/braintree/payment",    requireSignIn,    brainTreePaymentController);   export

default router;
```

================ models/categoryModel.js==================

```
import mongoose from "mongoose";

            const          categorySchema          =          new
             mongoose.Schema({name: {
              type: String,
              // required: true,
              // unique: true,
             },
             slug: {
              type:      String,
              lowercase: true,
             },
});

export default mongoose.model("Category", categorySchema);
```

================ models/orderModel.js==================

```
import mongoose from "mongoose";

const orderSchema = new mongoose.Schema(
          {
            products: [
              {
                type:
                mongoose.ObjectId,   ref:
                "Products",
              },
            ],
            payment:  {},
            buyer: {
              type:
              mongoose.ObjectId,   ref:
              "users",
            },
            status:        {
              type: String,
              default: "Not Process",
              enum: ["Not Process", "Processing", "Shipped", "deliverd", "cancel"],
            },
          },
          { timestamps: true }
);

export default mongoose.model("Order", orderSchema);
```

================ models/productModel.js==================

```
import mongoose from "mongoose";

const productSchema = new mongoose.Schema(
          {
            name: {
              type:   String,
              required: true,
            },
            slug: {
              type:   String,
              required: true,
            },
            description:   {
              type:   String,
              required: true,
            },
            price: {
```

```
            type: Number,
            required: true,
          },
          category: {
            type:
            mongoose.ObjectId,   ref:
            "Category",
            required: true,
          },
          quantity: { type:
            Number,
            required: true,
          },
          photo: {
            data:           Buffer,
            contentType: String,
          },
          shipping:       {
            type: Boolean,
          },
        },
        { timestamps: true }
);

export default mongoose.model("Products", productSchema);
```

================ models/userModel.js==================

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
        {
          name: {
            type:   String,
            required: true,
            trim: true,
          },
          email:  { type:
            String,
            required: true,
            unique: true,
          },
          password:    {
            type:   String,
            required: true,
          },
          phone: {
            type:   String,
            required: true,
          },
          address:      {
            type:        {},
            required: true,
          },
          answer:      {
            type: String,
            required: true,
          },
          role: {
            type: Number,
            default: 0,
          },
        },
        { timestamps: true }
);

export default mongoose.model("users", userSchema);
```

============ middlewares/authMiddleware.js==============

```
import JWT from "jsonwebtoken";
import userModel from "../models/userModel.js";

//Protected Routes token base
  export const requireSignIn = async (req, res, next) => {
   try {
    const  decode  =  JWT.verify(
      req.headers.authorization,
      process.env.JWT_SECRE
      T
    );
    req.user            =
    decode;next();
   }   catch   (error)   {
    console.log(error);
   }
};
```

```
//admin acceeess
          export const isAdmin = async (req, res, next) => {
           try {
            const user = await userModel.findById(req.user._id);
            if (user.role !== 1) {
             return
              res.status(401).send({
              success: false,
              message: "UnAuthorized Access",
             });
            } else {
             next();
            }
           } catch (error) {
            console.log(error);
            res.status(401).send(
            {success: false,
             error,
             message: "Error in admin middelware",
            });
           }
};
```

===================
helpers/authHelper.js=================

```
import bcrypt from "bcrypt";

          export const hashPassword = async (password) =>
           {try {
            const saltRounds = 10;
            const hashedPassword = await bcrypt.hash(password,
            saltRounds);return hashedPassword;
           } catch (error) {
            console.log(error);
           }
};

          export const comparePassword = async (password, hashedPassword) =>
           {return bcrypt.compare(password, hashedPassword);
};
```

============ controllers/authController.js==================

import userModel from "../models/userModel.js"; import orderModel from "../models/orderModel.js";

import { comparePassword, hashPassword } from "./../helpers/authHelper.js";import JWT from "jsonwebtoken";

```
export const registerController = async (req, res) => {
  try {
    const { name, email, password, phone, address, answer } = req.body;
    //validations
    if (!name) {
      return res.send({ error: "Name is Required" });
    }
    if (!email) {
      return res.send({ message: "Email is Required" });
    }
    if (!password) {
      return res.send({ message: "Password is Required" });
    }
    if (!phone) {
      return res.send({ message: "Phone no is Required" });
    }
    if (!address) {
      return res.send({ message: "Address is Required" });
    }
    if (!answer) {
      return res.send({ message: "Answer is Required" });
    }
    //check user
    const exisitingUser = await userModel.findOne({ email });
    //exisiting user
    if (exisitingUser) {
      return
        res.status(200).send({
        success: false,
        message: "Already Register please login",
      });
    }
    //register user
    const hashedPassword = await hashPassword(password);
    //save
```

```
            const user = await new userModel({
              name,
              email,
              phone,
              addres
              s,
              password:
              hashedPassword,answer,
            }).save();
            res.status(201).send(
              {success: true,
              message: "User Register
              Successfully",user,
            });
          } catch (error) {
            console.log(error);
            res.status(500).send(
            {success: false,
              message: "Errro in Registeration",
              error,
            });
          }
};

//POST LOGIN
        export const loginController = async (req, res) => {
          try {
            const { email, password } = req.body;
            //validation
            if (!email || !password) { return
              res.status(404).send({
              success: false,
                message: "Invalid email or password",
              });
            }
            //check user
            const user = await userModel.findOne({ email });
            if (!user) {
              return
                res.status(404).send({
                success: false,
                message: "Email is not registerd",
              });
            }
            const match = await comparePassword(password,
            user.password);if (!match) {
```

```
          return
            res.status(200).send({
            success: false,
            message: "Invalid Password",
          });
        }
        //token
        const token = await JWT.sign({ _id: user._id }, process.env.JWT_SECRET, {
          expiresIn: "7d",
        });
        res.status(200).send(
          {success: true,
          message: "login successfully",
          user: {
            _id:          user._id,
            name:  user.name,
            email:   user.email,
            phone:
            user.phone,
            address:
            user.address,     role:
            user.role,
          },
          token,
        });
      }     catch     (error)     {
        console.log(error);
        res.status(500).send(
        {success: false,
          message: "Error in login",
          error,
        });
      }
};
```

//forgotPasswordController

```
    export const forgotPasswordController = async (req, res) => {
      try {
        const { email, answer, newPassword } = req.body;
        if (!email) {
          res.status(400).send({ message: "Emai is required" });
        }
        if (!answer) {
          res.status(400).send({ message: "answer is required" });
```

```
            }
            if (!newPassword) {
             res.status(400).send({ message: "New Password is required" });
            }
            //check
            const user = await userModel.findOne({ email, answer });
            //validation
            if (!user) {
             return
              res.status(404).send({
              success: false,
              message: "Wrong Email Or Answer",
             });
            }
            const hashed = await hashPassword(newPassword);
            await userModel.findByIdAndUpdate(user._id, { password: hashed });
            res.status(200).send({
             success: true,
             message: "Password Reset Successfully",
            });
           }    catch    (error)    {
            console.log(error);
            res.status(500).send(
            {success: false,
             message: "Something went wrong",
             error,
            });
           }
};

//test controller
          export const testController = (req, res) => {
           try {
            res.send("Protected Routes");
           }   catch   (error)   {
            console.log(error);
            res.send({ error });
           }
};

//update prfole
          export const updateProfileController = async (req, res) => {
           try {
            const { name, email, password, address, phone } = req.body;
            const user = await userModel.findById(req.user._id);
```

```
        //password
        if (password && password.length < 6) {
          return res.json({ error: "Passsword is required and 6 character long" });
        }
        const hashedPassword = password ? await hashPassword(password) :
        undefined;const updatedUser = await userModel.findByIdAndUpdate(
          req.user._id,
          {
            name: name || user.name,
            password:    hashedPassword    ||
            user.password,phone: phone || user.phone,
            address: address || user.address,
          },
          { new: true }
        );
        res.status(200).send(
          {success: true,
          message: "Profile Updated SUccessfully",
          updatedUser,
        });
      }  catch  (error)  {
        console.log(error);
        res.status(400).send(
        {success: false,
          message: "Error WHile Update profile",
          error,
        });
      }
};

//orders
    export const getOrdersController = async (req, res) => {
      try {
        const orders = await orderModel
          .find({ buyer: req.user._id })
          .populate("products", "-photo")
          .populate("buyer", "name");
        res.json(orders);
      }  catch  (error)  {
        console.log(error);
        res.status(500).send(
        {success: false,
          message: "Error WHile Geting Orders",
          error,
        });
      }
};
//orders
export const getAllOrdersController = async (req, res) => {
      try {
        const orders = await orderModel
          .find({})
          .populate("products", "-photo")
          .populate("buyer", "name")
          .sort({  createdAt:  "-1"  });
        res.json(orders);
      }     catch     (error)     {
        console.log(error);
        res.status(500).send(
        {success: false,
          message: "Error WHile Geting Orders",
          error,
        });
      }
};

//order status
    export const orderStatusController = async (req, res) =>
      {try {
        const { orderId } = req.params;
        const { status } = req.body;
        const orders = await orderModel.findByIdAndUpdate(
          orderId,
          { status },
          { new: true }
        );
        res.json(orders);
      }     catch     (error)     {
        console.log(error);
        res.status(500).send(
        {success: false,
          message: "Error While Updateing Order",
          error,
        });
      }
};
```

============ controllers/categoryController.js==============

```
import categoryModel from "../models/categoryModel.js";import slugify
from "slugify";
export const createCategoryController = async (req, res) => {
        try {
          const { name } = req.body;
          if (!name) {
            return res.status(401).send({ message: "Name is required" });
          }
          const existingCategory = await categoryModel.findOne({ name });
          if (existingCategory) {
            return
              res.status(200).send({
              success: false,
              message: "Category Already Exisits",
            });
          }
          const category = await new categoryModel({
            name,
            slug: slugify(name),
          }).save();
          res.status(201).send(
          {success: true,
            message:    "new    category
            created",category,
          });
        }   catch   (error)   {
          console.log(error);
          res.status(500).send(
          {success: false,
            errro,
            message: "Errro in Category",
          });
        }
};

//update category
       export const updateCategoryController = async (req, res) =>
        {try {
          const { name } = req.body;
          const { id } = req.params;
          const category = await categoryModel.findByIdAndUpdate(
            id,
            { name, slug: slugify(name) },
            { new: true }
          );
          res.status(200).send(
            {success: true,
            messsage:          "Category          Updated
            Successfully",category,
          });
        }    catch    (error)    {
          console.log(error);
          res.status(500).send(
          {success: false,
            error,
            message: "Error while updating category",
          });
        }
};

// get all cat
      export const categoryControlller = async (req, res) => {
        try {
          const category = await categoryModel.find({});
          res.status(200).send({
            success: true,
            message: "All Categories List",
            category,
          });
        }    catch    (error)    {
          console.log(error);
          res.status(500).send(
          {success: false,
            error,
            message: "Error while getting all categories",
          });
        }
};

// single category
      export const singleCategoryController = async (req, res) => {
        try {
          const category = await categoryModel.findOne({ slug: req.params.slug });
          res.status(200).send({
            success: true,
            message:       "Get       SIngle       Category
            SUccessfully",category,
          });
```

```
           }    catch    (error)    {
            console.log(error);
            res.status(500).send(
             {success: false,
             error,
             message: "Error While getting Single Category",
            });
           }
};

//delete category
          export const deleteCategoryCOntroller = async (req, res) => {
           try {
            const { id } = req.params;
            await   categoryModel.findByIdAndDelete(id);
            res.status(200).send({
             success: true,
             message: "Categry Deleted Successfully",
            });
           }     catch     (error)     {
            console.log(error);
            res.status(500).send(
            {success: false,
             message: "error while deleting category",
             error,
            });
           }
};
```

============ controllers/productController.js==============

```
import productModel from "../models/productModel.js"; import
categoryModel from "../models/categoryModel.js"; import orderModel
from "../models/orderModel.js";

import fs from "fs";
import slugify from "slugify"; import braintree
from "braintree";import dotenv from "dotenv";

dotenv.config();

//payment gateway
          var gateway = new braintree.BraintreeGateway({
           environment: braintree.Environment.Sandbox,
           merchantId:   process.env.BRAINTREE_MERCHANT_ID,
           publicKey:        process.env.BRAINTREE_PUBLIC_KEY,
           privateKey: process.env.BRAINTREE_PRIVATE_KEY,
});

          export const createProductController = async (req, res) =>
           {try {
            const { name, description, price, category, quantity, shipping } =
             req.fields;
            const { photo } = req.files;
            //alidation
            switch  (true) {
            case !name:
              return res.status(500).send({ error: "Name is Required" });
             case !description:
              return res.status(500).send({ error: "Description is Required" });
             case !price:
              return res.status(500).send({ error: "Price is Required" });
             case !category:
              return res.status(500).send({ error: "Category is Required" });
             case !quantity:
              return res.status(500).send({ error: "Quantity is Required" });
             case photo && photo.size > 1000000:
              return res
               .status(500)
               .send({ error: "photo is Required and should be less then 1mb" });
            }

            const products = new productModel({ ...req.fields, slug: slugify(name) });if
            (photo) {
             products.photo.data                        =
             fs.readFileSync(photo.path);
             products.photo.contentType = photo.type;
            }
            await
            products.save();
            res.status(201).send(
            {success: true,
             message:        "Product        Created
             Successfully",products,
            });
           }   catch   (error)   {
            console.log(error);
            res.status(500).send(
            {success: false,
             error,
             message: "Error in crearing product",
            });
           }
};

//get all products
          export const getProductController = async (req, res) => {
           try {
            const products = await productModel
             .find({})
             .populate("category")
             .select("-photo")
             .limit(12)
             .sort({ createdAt: -1 });
            res.status(200).send({
            success: true,
             counTotal: products.length,
             message:  "ALlProducts  ",
             products,
            });
           }    catch    (error)    {
            console.log(error);
            res.status(500).send(
            {success: false,
             message: "Erorr in getting products",
             error: error.message,
            });
           }
};
// get single product
          export const getSingleProductController = async (req, res) => {
           try {
            const product = await productModel
             .findOne({ slug: req.params.slug })
             .select("-photo")
             .populate("category");
            res.status(200).send({
            success: true,
             message:      "Single      Product
             Fetched",product,
            });
           } catch (error) {
            console.log(error);
            res.status(500).send(
            {success: false,
             message: "Eror while getitng single product",
             error,
            });
           }
};

// get photo
          export const productPhotoController = async (req, res) => {
           try {
            const product = await productModel.findById(req.params.pid).select("photo");
            if (product.photo.data) {
             res.set("Content-type",
             product.photo.contentType);                         return
             res.status(200).send(product.photo.data);
            }
           }    catch     (error)     {
            console.log(error);
            res.status(500).send(
            {success: false,
             message: "Erorr while getting photo",
             error,
            });
           }
};

//delete controller
          export const deleteProductController = async (req, res) => {
           try {
            await   productModel.findByIdAndDelete(req.params.pid).select("-photo");
            res.status(200).send({
             success: true,
             message: "Product Deleted successfully",
            });
           }    catch     (error)     {
            console.log(error);
            res.status(500).send(
            {success: false,
             message: "Error while deleting product",
             error,
            });
           }
```

```
};

//upate producta
          export const updateProductController = async (req, res) =>
           {try {
            const { name, description, price, category, quantity, shipping } =
             req.fields;
            const { photo } = req.files;
            //alidation
            switch (true) {
            case !name:
              return res.status(500).send({ error: "Name is Required" });
             case !description:
              return res.status(500).send({ error: "Description is Required" });
             case !price:
              return res.status(500).send({ error: "Price is Required" });
             case !category:
              return res.status(500).send({ error: "Category is Required" });
             case !quantity:
              return res.status(500).send({ error: "Quantity is Required" });
             case photo && photo.size > 1000000:
              return res
               .status(500)
               .send({ error: "photo is Required and should be less then 1mb" });
            }

            const products = await productModel.findByIdAndUpdate(
             req.params.pid,
             { ...req.fields, slug: slugify(name) },
             { new: true }
            );
            if (photo) {
             products.photo.data =
             fs.readFileSync(photo.path);
             products.photo.contentType = photo.type;
            }
            await
            products.save();
            res.status(201).send(
            {success: true,
             message: "Product Updated
             Successfully",products,
            });
           } catch (error) {
            console.log(error);
```

```
                        res.status(500).send(
                          {success: false,
                          error,
                          message: "Error in Updte product",
                        });
                      }
};

// filters
                    export const productFiltersController = async (req, res) => {
                      try {
                        const { checked, radio } = req.body;
                        let args = {};
                        if (checked.length > 0) args.category = checked;
                        if (radio.length) args.price = { $gte: radio[0], $lte: radio[1] };
                        const products = await productModel.find(args);
                        res.status(200).send({
                          success:
                          true,
                          products,
                        });
                      } catch (error) {
                        console.log(error);
                        res.status(400).send(
                        {success: false,
                          message: "Error WHile Filtering Products",
                          error,
                        });
                      }
};

// product count
                    export const productCountController = async (req, res) => {
                      try {
                        const total = await productModel.find({}).estimatedDocumentCount();
                        res.status(200).send({
                          success:
                          true,total,
                        });
                      } catch (error) {
                        console.log(error);
                        res.status(400).send(
                        {
                          message: "Error in product count",
                          error,
             success: false,
            });
           }
};

// product list base on page
          export const productListController = async (req, res) => {
           try {
            const perPage = 6;
            const page = req.params.page ? req.params.page :
            1;const products = await productModel
             .find({})
             .select("-photo")
             .skip((page - 1) * perPage)
             .limit(perPage)
             .sort({ createdAt: -1 });
            res.status(200).send({
            success: true,
             products,
            });
           }   catch   (error)   {
            console.log(error);
            res.status(400).send(
            {success: false,
             message: "error in per page ctrl",
             error,
            });
           }
};

// search product
          export const searchProductController = async (req, res) =>
           {try {
            const  {  keyword  }  =  req.params;
            const resutls = await productModel
             .find({
              $or: [
               { name: { $regex: keyword, $options: "i" } },
               { description: { $regex: keyword, $options: "i" } },
              ],
             })
             .select("-photo");
            res.json(resutls);
          }     catch     (error)     {
            console.log(error);
            res.status(400).send(
            {success: false,
              message: "Error In Search Product API",
              error,
            });
          }
};

// similar products
          export const realtedProductController = async (req, res) => {
          try {
            const { pid, cid } = req.params;
            const products = await productModel
              .find({
                category: cid,
                _id: { $ne: pid },
              })
              .select("-photo")
              .limit(3)
              .populate("category");
            res.status(200).send({
            success: true,
              products,
            });
          }     catch     (error)     {
            console.log(error);
            res.status(400).send(
            {success: false,
              message: "error while geting related product",
              error,
            });
          }
};

// get prdocyst by catgory
          export const productCategoryController = async (req, res) => {
          try {
            const  category  =  await  categoryModel.findOne({  slug:  req.params.slug  });
            const products = await productModel.find({ category }).populate("category");
            res.status(200).send({
              success: true,
              category
              ,
              products
              ,
            });
          }     catch     (error)     {
            console.log(error);
            res.status(400).send(
            {success: false,
              error,
              message: "Error While Getting products",
            });
          }
};

//payment gateway api
//token
          export const braintreeTokenController = async (req, res) => {
          try {
            gateway.clientToken.generate({}, function (err, response) {
              if (err) {
                res.status(500).send(err);
              }          else          {
                res.send(response
                );
              }
            });
          }   catch    (error)    {
            console.log(error);
          }
};

//payment
          export const brainTreePaymentController = async (req, res) => {
          try {
            const { nonce, cart } = req.body;
            let total = 0;
            cart.map((i)  =>  {
              total += i.price;
            });
            let newTransaction = gateway.transaction.sale(
              {
                amount:                          total,
                paymentMethodNonce:
```

```
            nonce,      options:      {
            submitForSettlement: true,
            },
          },
          function (error, result) {if
            (result) {
              const order = new orderModel({
                products: cart,
                payment:      result,
                buyer: req.user._id,
              }).save();
              res.json({ ok: true });
            }         else         {
              res.status(500).send(error
              );
            }
          }
        );
      }   catch   (error)   {
        console.log(error);
      }
};
```

=============== package.json==================

```
{
      "name": "SHOPPING CART",
      "version": "1.0.0",
      "description": "ecommerce rest api",
      "main": "server.js",
      "type":   "module",
      "scripts": {
        "start":      "node      server.js",
        "server": "nodemon server.js",
        "client": "npm start --prefix ./client",
        "dev": "concurrently \"npm run server\" \"npm run client\""
      },
      "keywords":         [],
      "author":   "Techinfoyt",
      "license":        "MIT",
      "dependencies": {
        "@sendgrid/mail": "^7.7.0",
        "bcrypt": "^5.1.0",
        "braintree": "^3.13.0",
        "colors": "^1.4.0",
        "concurrently": "^7.6.0",
        "cors": "^2.8.5",
        "dotenv": "^16.0.3",
        "express": "^4.18.2",
        "express-formidable":  "^1.2.0",
        "jsonwebtoken": "^9.0.0",
        "mongoose": "^6.8.4",
        "morgan": "^1.10.0",
        "nodemon": "^2.0.20",
        "react-icons": "^4.7.1",
        "slugify": "^1.6.5"
      }
}
```

Implementation of Security Mechanisms at Various Levels

Security is a paramount concern for any e-commerce system, as it deals with sensitive user data (personal information, payment details) and critical business operations (transactions, inventory). A multi-layered approach to security will be implemented, covering the frontend, backend, database, and network communication.

A. Frontend Security

While the frontend primarily deals with presenting data and collecting user input, it's the first line of defense against client-side attacks and plays a crucial role in user trust.

6. **Input Validation and Sanitization:**
    - **Purpose: Prevent malicious data from reaching the backend and causing vulnerabilities like XSS.**
    - **Implementation: All user inputs (e.g., search queries, form fields like name, email, address, product descriptions) will be validated on the client-side using JavaScript. This includes checking data types, length constraints, format (e.g., email regex), and preventing special characters that could lead to injection.**
    - **Techniques: Utilize React's state management and event handlers to perform real-time validation feedback to the user. Libraries like `formik` or `react-hook-form` with validation schemas (e.g., `yup`) can streamline this process. Although client-side validation enhances UX, it must always be complemented by server-side validation, as client-side validation can be bypassed.**
7. **Cross-Site Scripting (XSS) Prevention:**
    - **Purpose: Prevent attackers from injecting malicious scripts into web pages viewed by other users.**
    - **Implementation: When displaying user-generated content (e.g., product reviews, comments), ensure that it is properly escaped or sanitized. React inherently provides some protection against XSS by escaping content by default when rendering JSX, but for dynamically inserted HTML (e.g., using `dangerouslySetInnerHTML`), explicit sanitization libraries (e.g., `DOMPurify`) must be used.**
8. **Content Security Policy (CSP):**

   - **Purpose: Mitigate XSS attacks by specifying trusted sources of content (scripts, styles, images, etc.).**
   - **Implementation: Configure CSP headers on the server-side to instruct the browser about legitimate sources for various resources. This helps prevent the browser from loading malicious scripts injected from untrusted domains.**
9. **Secure Local Storage/Session Storage Usage:**
   - **Purpose: Avoid storing sensitive information client-side.**
   - **Implementation: Avoid storing sensitive user data (e.g., raw tokens, passwords, private keys) directly in `localStorage` or `sessionStorage`. JWTs (JSON Web Tokens) should be stored securely, ideally in HTTP-only cookies, which are less susceptible to XSS attacks than `localStorage` when managed correctly, or if stored in `localStorage`, additional measures (e.g., token refresh mechanisms, short expiry times) must be considered.**
10. **Protection against Clickjacking:**
    - **Purpose: Prevent malicious websites from tricking users into clicking on hidden elements of the e-commerce site.**
    - **Implementation: Implement X-Frame-Options or Content-Security-Policy with `frame-ancestors` directive to prevent the site from being embedded in iframes on other domains.**

B. Backend Security

The backend is the core of the application where most business logic and data interactions occur. Securing the backend is critical to protecting data integrity and confidentiality.

7. **Authentication and Authorization:**
   - **Purpose: Verify user identity and control access to resources based on roles.**
   - **Implementation:**
     - **User Registration: Implement strong password policies (minimum length, complexity requirements). Store passwords using strong, one-way hashing algorithms like Bcrypt with sufficient salt rounds. Never store plain text passwords.**
     - **Login: Implement rate limiting on login attempts to prevent brute-force attacks. Provide clear but generic error messages for failed login attempts to avoid leaking information about valid usernames.**
     - **JSON Web Tokens (JWT): Use JWTs for session management. Upon successful login, a short-lived access token and a longer-lived refresh token will be issued. The access token will be sent with each request to authorize access.**
     - **Role-Based Access Control (RBAC): Implement middleware in Express.js to verify user roles (e.g., `customer`, `admin`) before allowing access to specific routes or performing certain actions. For example, only an admin user can access `/api/admin/products` to add or edit products.**
     - **Token Management: Access tokens should have short expiry times to minimize the window of opportunity for token compromise. Refresh tokens should be used to obtain new access tokens, and they should be stored securely (e.g., in an HTTP-only cookie). Blacklisting compromised tokens will also be considered.**
8. **API Security:**
   - **Purpose: Protect API endpoints from unauthorized access and malicious use.**
   - **Implementation:**

- **HTTPS/SSL/TLS: All communication between the frontend and backend will be encrypted using HTTPS to prevent eavesdropping and Man-in-the-Middle (MitM) attacks.**
- **CORS (Cross-Origin Resource Sharing): Properly configure CORS headers in Express.js to specify which origins are allowed to access the API. This prevents unauthorized domains from making requests to the backend.**
- **Rate Limiting: Implement rate limiting on sensitive or resource-intensive API endpoints (e.g., login, registration, password reset, search) to prevent denial-of-service (DoS) attacks and abuse. Libraries like `express-rate-limit` can be used.**
- **Input Validation and Sanitization (Server-side): This is the most critical validation layer. All data received from the client must be validated and sanitized on the server before processing or storing it in the database. This prevents various injection attacks (e.g., NoSQL injection for MongoDB, if not careful with query construction) and ensures data integrity.**
- **Parameter Tampering Prevention: Ensure that critical parameters (e.g., product price, order total) are never sent from the client or, if sent, are re-validated and re-calculated on the server-side to prevent users from manipulating them.**

9. **Error Handling and Logging:**
    - **Purpose: Identify and respond to security incidents and system anomalies.**
    - **Implementation: Implement centralized error handling in Express.js to catch unhandled exceptions and prevent sensitive information from being exposed in error responses. Use robust logging mechanisms (e.g., Winston, Morgan) to record significant events, including authentication attempts, failed requests, and system errors. Logs should be regularly reviewed and securely stored.**
10. **Dependency Management:**
    - **Purpose: Mitigate vulnerabilities introduced by third-party libraries.**
    - **Implementation: Regularly update Node.js, npm packages, and other dependencies to their latest stable versions to benefit from security patches. Use tools like `npm audit` or `Snyk` to scan for known vulnerabilities in dependencies.**
11. **Environment Variables:**
    - **Purpose: Securely manage sensitive configuration data.**
    - **Implementation: Store all sensitive configuration (e.g., database connection strings, API keys, JWT secrets) in environment variables rather than hardcoding them in the codebase. Use a `.env` file for local development and proper environment variable management in production.**
12. **HTTP Headers Configuration:**
    - **Purpose: Enhance overall application security.**
    - **Implementation: Configure various HTTP security headers (e.g., `Strict-Transport-Security`, `X-Content-Type-Options`, `X-Frame-Options`, `X-XSS-Protection`) using Express.js middleware like `helmet` to protect against common web vulnerabilities.**

C. Database Security (MongoDB)

Securing the database is critical as it holds all application data.

7. **Authentication and Authorization:**

    - **Purpose: Control who can access the database and what operations they can perform.**
    - **Implementation: Enable authentication in MongoDB. Create dedicated database users with specific roles and privileges (e.g., read-only access for some operations, read/write for others). Avoid using the default root user for application interactions.**
8. **Network Security:**
    - **Purpose: Restrict network access to the database.**
    - **Implementation: Configure network rules (firewalls, security groups) to only allow connections from the application server's IP address. Do not expose the MongoDB port to the public internet. Use a Virtual Private Cloud (VPC) for secure database deployment.**
9. **Encryption:**
    - **Purpose: Protect data at rest and in transit.**
    - **Implementation:**
        - **TLS/SSL: Enable TLS/SSL for all connections to the MongoDB database to encrypt data in transit.**
        - **Encryption at Rest: For highly sensitive data, consider encryption at rest (e.g., MongoDB's WiredTiger storage engine offers native encryption, or file-system level encryption). For specific sensitive fields within documents, application-level encryption can be implemented before storing data in MongoDB.**
10. **Input Validation (Mongoose Schema):**
    - **Purpose: Ensure data integrity and prevent malformed data.**
    - **Implementation: Utilize Mongoose schemas to define the structure, data types, and validation rules for documents stored in MongoDB. This adds another layer of validation before data is persisted.**
11. **Regular Backups:**
    - **Purpose: Data recovery in case of data loss or corruption.**
    - **Implementation: Implement a regular backup strategy for the MongoDB database, storing backups securely and off-site.**
12. **Least Privilege Principle:**
    - **Purpose: Minimize potential damage from compromised accounts.**
    - **Implementation: Configure database users with the minimum necessary privileges required for their operations. For example, the application user only needs read/write access to specific collections, not administrative privileges.**

D. Server/Infrastructure Security

Securing the underlying infrastructure hosting the MERN stack.

5. **Operating System Security:**
    - **Purpose: Protect the server from direct attacks.**
    - **Implementation: Keep the operating system (Linux, Windows Server) updated with the latest security patches. Disable unnecessary services, use strong SSH key authentication instead of passwords, and configure firewalls to restrict access to necessary ports only.**
6. **Process Isolation:**
    - **Purpose: Isolate different services to prevent a compromise in one from affecting others.**
    - **Implementation: Run Node.js applications as non-root users. If using microservices (future scope), consider containerization (Docker) for process isolation.**

7. **Monitoring and Alerting:**
   - **Purpose: Detect and respond to suspicious activities promptly.**
   - **Implementation: Implement system-level monitoring for CPU usage, memory, disk I/O, network traffic, and logs. Set up alerts for unusual patterns or potential security breaches.**
8. **Regular Security Audits and Penetration Testing:**
   - **Purpose: Proactively identify vulnerabilities.**
   - **Implementation: Conduct regular security audits and penetration testing by independent security experts to uncover vulnerabilities that might be missed during development.**

By integrating these security mechanisms at every level of the "Shopping Cart System," the project aims to provide a robust, secure, and trustworthy e-commerce platform for both users and administrators.

Future Scope and Further Enhancements

The "Shopping Cart System" project, developed using the MERN stack, lays a strong foundation for a functional e-commerce platform. However, the rapidly evolving nature of online commerce demands continuous innovation and enhancement. This section outlines the potential future scope and further enhancements that can be incorporated to evolve the system into a more comprehensive, intelligent, and user-centric solution. These enhancements are categorized to provide a structured roadmap for future development.

A. Core E-commerce Feature Enhancements

11. **Advanced Search and Filtering:**
    - **Current: Basic keyword search and category filtering.**
    - **Future: Implement faceted search (filtering by attributes like brand, price range, color, size, ratings), autocomplete suggestions, and "did you mean" functionality. Integration with search engines like Elasticsearch or Algolia could provide highly performant and relevant search results, handling large product catalogs efficiently.**
12. **Personalization and Recommendation Engine:**
    - **Current: No personalization.**
    - **Future: Implement algorithms to recommend products based on user browsing history, purchase history, items in the cart, popular products, or collaborative filtering (e.g., "customers who bought this also bought..."). This significantly enhances user engagement and sales. Data collected from user interactions would feed into this system, leveraging MongoDB's flexible schema for storing user behavior data.**
13. **Customer Reviews and Ratings System:**
    - **Current: Basic placeholder for reviews.**
    - **Future: Develop a robust system for customers to submit star ratings and written reviews for products. Include features like review moderation by administrators, sorting reviews (e.g., most helpful, newest), and filtering by star rating. This builds trust and aids purchasing decisions.**
14. **Wishlist Functionality:**
    - **Current: Not explicitly defined beyond "wishlists" in introduction.**

- **Future: Allow users to save products to a personal wishlist for later purchase. Include options to share wishlists, move items from wishlist to cart, and receive notifications when wishlist items go on sale or are back in stock.**

15. **Multi-Vendor Support / Marketplace:**
    - **Current: Single vendor system.**
    - **Future: Transform the platform into a marketplace where multiple vendors can register, list their products, and manage their own inventories and orders. This would require dedicated vendor dashboards, commission management, and more complex order routing logic.**
16. **Inventory Management Enhancements:**
    - **Current: Basic deduction upon purchase.**
    - **Future: Implement real-time inventory tracking, low-stock alerts for administrators, stock reservation for items in cart, and bulk import/export functionalities for product data. Integration with barcode scanners could further streamline operations for physical inventory.**
17. **Order Tracking and Notifications:**
    - **Current: Basic order status viewing.**
    - **Future: Provide detailed order tracking (e.g., "Order placed," "Processing," "Shipped," "Out for delivery," "Delivered"). Implement automated email/SMS notifications for status changes, shipping updates, and delivery confirmations.**
18. **Promotions and Discount Management:**
    - **Current: No explicit mention.**
    - **Future: Develop a system for administrators to create and manage various types of promotions: percentage discounts, fixed amount discounts, "buy one get one free" offers, free shipping, coupon codes, and loyalty programs. This requires sophisticated rule-based pricing logic.**
19. **Product Variants and Options:**
    - **Current: Assumed simple products.**
    - **Future: Support for products with multiple variants (e.g., T-shirt in different sizes and colors) and configurable options, each with its own SKU, price, and inventory.**
20. **Abandoned Cart Recovery:**
    - **Current: Not addressed.**
    - **Future: Implement a system to track abandoned carts and send automated follow-up emails to remind users about their unpurchased items, potentially offering incentives.**

B. User Experience (UX) and Interface (UI) Improvements

5. **Enhanced User Dashboard:**
    - **Current: Basic profile and order history.**
    - **Future: A more comprehensive user dashboard displaying recent activity, saved payment methods, loyalty points, personalized recommendations, and easy access to support.**
6. **Improved Checkout Experience:**
    - **Current: Standard multi-step checkout.**
    - **Future: Implement guest checkout, one-page checkout, express checkout options (e.g., using saved addresses and payment methods), and integration with digital wallets (Apple Pay, Google Pay).**
7. **Advanced Image and Media Handling:**
    - **Current: Basic image display.**

  - **Future: Implement image optimization (lazy loading, responsive images), image galleries, product videos, and 360-degree product views to enhance visual merchandising.**

8. **Accessibility (A11Y) Compliance:**
   - **Current: Basic web standards.**
   - **Future: Ensure the entire application adheres to WCAG (Web Content Accessibility Guidelines) standards, making it usable for individuals with disabilities (e.g., screen reader compatibility, keyboard navigation).**

Scalability and Performance Enhancements

6. **Caching Mechanisms:**
   - **Current: Not explicitly mentioned beyond general performance optimization.**
   - **Future: Implement various caching strategies:**
     - **Client-side caching: Browser caching for static assets.**
     - **Server-side caching: Redis or Memcached for frequently accessed data (e.g., product listings, categories) to reduce database load.**
     - **CDN (Content Delivery Network): For serving static assets (images, CSS, JS) to users from geographically closer servers, improving load times.**
7. **Load Balancing and Horizontal Scaling:**
   - **Current: Single server deployment assumption.**
   - **Future: Deploy the Node.js backend behind a load balancer (e.g., Nginx) to distribute incoming traffic across multiple server instances. This allows for horizontal scaling to handle increased user loads and provides high availability.**
8. **Database Optimization and Sharding:**
   - **Current: Basic MongoDB usage.**
   - **Future: As data grows, implement advanced MongoDB indexing, optimize complex queries, and consider sharding strategies to distribute data across multiple servers, improving performance and scalability.**
9. **Microservices Architecture:**
   - **Current: Monolithic MERN application.**
   - **Future: Decompose the monolithic application into smaller, independent microservices (e.g., separate services for user management, product catalog, order processing, payment) communicating via APIs. This enhances scalability, fault isolation, and independent deployment.**
10. **Serverless Functions:**
    - **Current: Traditional Node.js server.**
    - **Future: Utilize serverless functions (e.g., AWS Lambda, Google Cloud Functions) for specific, event-driven tasks (e.g., image resizing upon upload, sending order confirmation emails) to reduce server overhead and costs.**

Integration with External Services

7. **CRM (Customer Relationship Management) Integration:**
   - **Current: No CRM.**
   - **Future: Integrate with CRM systems (e.g., Salesforce, HubSpot) to manage customer interactions, support, and sales pipelines more effectively.**
8. **Email Marketing and Automation:**
   - **Current: No email capabilities.**
   - **Future: Integrate with email marketing platforms (e.g., Mailchimp, SendGrid) for transactional emails (order confirmations, shipping updates) and marketing campaigns (newsletters, promotions).**
9. **Analytics Tools:**
   - **Current: No explicit analytics.**
   - **Future: Integrate with Google Analytics, Mixpanel, or other analytics platforms to gain insights into user behavior, traffic sources, conversion rates, and sales performance.**

10. **Live Chat/Customer Support Integration:**
    - **Current: Manual customer support.**
    - **Future: Embed live chat widgets (e.g., Intercom, Zendesk Chat) or integrate with customer support ticketing systems for real-time assistance.**
11. **SMS Integration:**
    - **Current: No SMS.**
    - **Future: For critical notifications like order updates, delivery confirmations, or password resets.**
12. **Social Media Integration:**
    - **Current: No social login/sharing.**
    - **Future: Implement social login (e.g., Google, Facebook) for easier registration. Enable social sharing of products.**

Advanced Business Intelligence and Reporting

3. **Comprehensive Admin Dashboard and Reporting:**
   - **Current: Basic product/order/user management.**
   - **Future: Develop a more sophisticated admin dashboard with advanced analytics and reporting features, including:**
     - **Sales Reports: Daily, weekly, monthly, yearly sales, sales by product, sales by category, top-selling products.**
     - **Customer Reports: New customers, active customers, customer lifetime value.**
     - **Inventory Reports: Stock levels, low-stock items, fast/slow-moving items.**
     - **Marketing Reports: Campaign performance, coupon usage.**
     - **Dashboards with Visualizations: Use charting libraries (e.g., Recharts for React, D3.js) to provide interactive data visualizations for key performance indicators (KPIs).**
4. **A/B Testing Framework:**
   - **Current: No A/B testing.**
   - **Future: Integrate a framework that allows for A/B testing of different UI elements, product presentations, or pricing strategies to optimize conversion rates.**

DevOps and Deployment Enhancements

3. **CI/CD Pipeline:**
   - **Current: Manual deployment.**
   - **Future: Implement a Continuous Integration/Continuous Deployment (CI/CD) pipeline using tools like Jenkins, GitLab CI/CD, or GitHub Actions. This automates testing and deployment processes, ensuring faster and more reliable releases.**
4. **Containerization (Docker) and Orchestration (Kubernetes):**
   - **Current: Direct server deployment.**
   - **Future: Containerize the MERN application components using Docker. Deploy and manage these containers using Kubernetes for robust orchestration, scaling, and self-healing capabilities.**

By planning for these future enhancements, the "Shopping Cart System" can evolve into a dynamic, feature-rich, and highly competitive e-commerce solution capable of adapting to market demands and providing a superior user experience. This phased approach allows for continuous delivery of value and ensures the long-term viability and success of the project.

Bibliography

The information and concepts presented in this reformulated synopsis draw upon established principles of software engineering, web development best practices, and the specifics of the MERN stack. The following resources, categories of resources, and academic principles form the basis of the methodologies and technical details discussed.

A. General Software Engineering & Project Management

1. **Pressman, R. S., & Lowe, D. (2019). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.**
   - **Provides foundational knowledge on software development life cycles, requirement engineering, design, testing, and project management methodologies, including Agile principles.**
2. **Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall.**
   - **Offers insights into Use Case and Sequence Diagrams, object-oriented analysis, and iterative development processes like those used in Agile.**
3. **PMBOK Guide (Project Management Body of Knowledge). Project Management Institute.**
   - **Reference for standard project management processes, planning, scheduling, risk management, and quality assurance.**
4. **Beck, K., et al. (2001). *Manifesto for Agile Software Development*.**
   - **The foundational document outlining the values and principles behind Agile methodologies.**

B. Web Development & MERN Stack Specifics

1. **MongoDB Documentation (docs.mongodb.com):**
   - **Official documentation for MongoDB, covering schema design, querying, indexing, replication, sharding, and security configurations.**
2. **Express.js Documentation (expressjs.com):**
   - **Official guide for building web applications and APIs with Express.js, including routing, middleware, and error handling.**
3. **React Documentation (react.dev):**
   - **Official and comprehensive resource for learning and understanding React, including components, state management, hooks, and best practices for building user interfaces.**
4. **Node.js Documentation (nodejs.org):**

    - Official documentation for Node.js, detailing its runtime environment, modules, and asynchronous I/O model.
5. Crockford, D. (2008). *JavaScript: The Good Parts*. O'Reilly Media.
    - Although an older text, it provides fundamental insights into JavaScript, which is the cornerstone of the MERN stack.
6. MDN Web Docs (developer.mozilla.org/en-US/docs/Web):
    - A comprehensive resource for web technologies, including HTML, CSS, JavaScript, Web APIs, and web security concepts.
7. Tailwind CSS Documentation (tailwindcss.com):
    - Reference for utility-first CSS framework used for responsive design and rapid UI development.

C. Security & Data Protection

1. OWASP Top 10 (owasp.org/www-project-top-ten/):
    - The Open Web Application Security Project (OWASP) Top 10 list of the most critical web application security risks. This serves as a primary guide for identifying and mitigating vulnerabilities.
2. Saltzer, J. H., & Schroeder, M. D. (1975). *The Protection of Information in Computer Systems*. Communications of the ACM, 17(7), 384-402.
    - A seminal paper introducing foundational security principles like the "principle of least privilege."
3. Ferguson, N., Schneier, B., & Kohno, T. (2010). *Cryptography Engineering: Design Principles and Practical Applications*. John Wiley & Sons.
    - For understanding cryptographic principles applied to password hashing (Bcrypt) and secure communication (TLS/SSL).
4. NIST Special Publication 800-63-3: Digital Identity Guidelines (pages.nist.gov/800-63-3/):
    - Provides guidelines on digital identity, including authentication and password management.

D. Scalability & Performance

1. Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media.
    - Explores concepts of data storage, processing, and system design for scalability and reliability, relevant for MongoDB and overall architecture.
2. Google Web Vitals (web.dev/vitals/):
    - Guidelines and metrics for web performance, influencing optimization strategies.

E. Specific Libraries and Tools

1. Mongoose (mongoosejs.com):
    - Official documentation for the ODM library used to interact with MongoDB from Node.js applications, including schema validation.
2. JWT (jwt.io):
    - Official site for JSON Web Tokens, explaining their structure and usage for authentication.
3. Bcrypt.js (github.com/dcodeIO/bcrypt.js):
    - Documentation for the JavaScript library used for password hashing.
4. Helmet.js (helmetjs.github.io/):
    - Middleware for Express.js that sets various HTTP headers to improve security.
5. Express-rate-limit (express-rate-limit.cyclic.app/):

- **Middleware for Node.js to limit repeated requests to public APIs and/or endpoints.**

1. **Security:** Like with any web application, security measures must be implemented toprotect sensitive customer data, prevent unauthorized access, and guard against common vulnerabilities such as cross-site scripting (XSS) or SQL injection attacks. Proper authentication and authorization mechanisms should be implemented to ensure secure transactions.

2. **Scalability and maintenance:** While the MERN stack is scalable, additional considerations need to be made for horizontal scaling, load balancing, and managinglarge databases. Proper maintenance and monitoring of the application's infrastructure are necessary to ensure smooth operation as traffic increases.

FUTURE SCOPE

The future scope of an eCommerce web application is promising, as the eCommerce industry continues to grow and evolve. Here are some key areas that are likely to shape the future of eCommerce web applications:

1. **Mobile Commerce:** With the increasing use of smartphones and mobile devices, eCommerce applications will need to be optimized for mobile platforms. Mobile commerce, or m-commerce, is expected to play a significant role in the future, with more users shopping and making purchases through their mobile devices. This includesresponsive web design, mobile apps, and mobile payment solutions.

2. **Personalization and Customization:** eCommerce applications will focus on providing personalized experiences to customers. Advanced algorithms and machine learning techniques will be used to analyze user data, preferences, and behavior to offertailored product recommendations, personalized offers, and customized user interfaces.

3. **Voice Commerce:** Voice assistants like Amazon Alexa, Google Assistant, and Apple Siri are becoming increasingly popular. eCommerce applications will need to integratevoice recognition technology to enable voice-based shopping experiences. Customerswill be able to search for products, place orders, and track shipments using voice commands.

4. **Augmented Reality (AR) and Virtual Reality (VR):** AR and VR technologies will enhance the eCommerce experience by allowing customers to visualize products in a virtual environment. Customers will be able to try on clothes, see how furniture looksin their homes, or even experience virtual walkthroughs of physical stores.

5. **Social Commerce:** Social media platforms are already playing a significant role in eCommerce. In the future, eCommerce applications will integrate more closely with social media platforms, allowing users to discover products, make purchases, and sharetheir shopping experiences with friends and followers.

6. **Artificial Intelligence (AI) and Chatbots:** AI-powered chatbots will continue to improve customer support and enhance the overall shopping experience. Chatbots canassist customers in real-time, provide product recommendations, answer queries, and even process transactions.

7. **Blockchain Technology:** Blockchain technology can improve security, transparency,and trust in eCommerce transactions. It can be used for secure payments, supply chain management, and verifying the authenticity of

products.

8. **Omnichannel Experience:** The future of eCommerce will involve seamless integration across multiple channels and touchpoints, including online stores, mobile apps, social media platforms, and physical stores. Customers will expect a consistent and personalized experience, regardless of the channel they choose.

9. **Sustainability and Ethical Practices:** As environmental and ethical concerns gain importance, eCommerce applications will need to address these issues. Customers willseek out platforms that promote sustainable practices, offer eco-friendly products, andsupport social causes.

10. **Data Security and Privacy:** With increasing cyber threats and data breaches, eCommerce applications must prioritize data security and user privacy. Robust securitymeasures, encryption techniques, and compliance with data protection regulations willbe crucial.

These are just a few examples of the future scope of eCommerce web applications. As technology advances and consumer demands evolve, eCommerce platforms will need to adapt and innovate to stay competitive in the market.

LIMITATION OF PROJECT

1. Review and Feedback feature is not provided.
2. Product Reviews feature is not provided yet.
3. In support section Chat Bot System is not provided.
4. In Profile section many details of user are not featured.
5. After checkout bill generation system is not provided.
6. Admin report generation system is not provided.
7. Mobile number and Email authentication features are not implemented.

BIBLIOGRAPHY

1. Stack Overflow – Stack Overflow is a popular question-and-answer website forprogrammers and developers helps us to get rid of different types of error while working with different programming languages. (https://stackoverflow.com/)

2. The npmjs website is the official website for the npm (Node Package Manager)registry. It provides several features and services to developers working with JavaScript and Node.js. (http://npmjs.com)

3. Bootstrap: Its help to build proper responsive UI by using its documentation.(https://getbootstrap.com/)

4. Uxwing: This website gives free icons in .SVG, .PNG that can be usedcommercially. (https://uxwing.com/)

5. Reactdev: This website gives an introduction to the 80% of React concepts that youwill use on a daily basis. (https://react.dev/learn/)

YouTube: Video Tutorial helps me build and guide at different phases while