

Abstract

The E-COMMERCE-WEBSITE is a modern, single-page web application designed to replicate the essential features of an online retail platform. Developed using React with TypeScript, and styled using Tailwind CSS, the project emphasizes a modular, scalable, and responsive architecture that aligns with current web development standards. The use of Vite as a build tool ensures faster development and optimized builds, improving both performance and developer productivity.

This project serves as a front-end blueprint for e-commerce applications, showcasing a clean UI layout that could include essential features such as product listings, filtering options, product detail pages, and a shopping cart interface (based on future expansions). The integration of PostCSS and ESLint further ensures code consistency, maintainability, and adherence to best practices.

The website is built with responsiveness in mind, allowing seamless navigation across desktop, tablet, and mobile devices. The choice of TypeScript improves code reliability and scalability, reducing the likelihood of runtime errors through static type-checking. Furthermore, Tailwind CSS allows rapid UI development with utility-first CSS classes, improving design efficiency.

While this version focuses primarily on the front-end, the project lays the groundwork for full-stack integration. Potential extensions include user authentication, order management, payment gateways, and real-time database connectivity, making it suitable for real-world e-commerce deployments. As an educational and portfolio project, it demonstrates proficiency in modern web development practices and can serve as a foundational component for future commercial or academic use.

Introduction of Ecommerce Website

ShopEase is an open-source, full-stack e-commerce web application designed to help small and medium-sized businesses (SMBs) establish a digital storefront quickly and affordably. Built using the **MERN stack** (MongoDB, Express.js, React.js, Node.js), it provides a modern, scalable solution with essential online shopping features.

As e-commerce becomes a vital part of the global economy, many smaller businesses are left behind due to the high cost and complexity of building their own platforms. ShopEase bridges this gap by offering a ready-to-use, customizable platform with no licensing costs — ideal for developers, students, startups, or retailers aiming to go online.

Key Technologies:

- **Frontend:** TypeScript
- **Styling:** Tailwind CSS
- **Build Tool:** Vite
- **Configuration Files:** eslint.config.js, postcss.config.js, tailwind.config.js, tsconfig.json, vite.config.ts

Features of Ecommerce Website:

1. User Authentication

- Secure login and registration using JWT
- Role-based access (User/Admin)

2. Product Management

- Browse, search, and view detailed product information
- Admins can add, update, or delete products

3. Shopping Cart & Wishlist

- Users can add products to a cart or wishlist
- View and modify cart before checkout

4. Order Placement & Tracking

- Seamless checkout process
- Order summary and history for users

5. Admin Dashboard

- Manage inventory, orders, and users
- Simple, intuitive UI for backend operations

6. Responsive Design

- Fully functional across devices (mobile, tablet, desktop)

Advantages of Ecommerce Website

- **Open Source & Free**
No licensing or subscription costs. Easily customizable for various business needs.
- **Quick Deployment**
Can be deployed on cloud platforms like Vercel, Render, or Heroku with minimal setup.
- **Scalable Architecture**
Built using industry-standard technologies (MERN stack) that support future scaling and integration.
- **Developer-Friendly**
Clean, modular code structure that can be extended or adapted by developers.
- **Empowers Small Businesses**
Eliminates dependence on third-party marketplaces. Businesses retain full control over branding, products, and customer data.
- **Educational Value**
Great learning project for students and developers interested in full-stack development.

Problem statement

Small and medium-sized businesses (SMBs) play a vital role in local economies and global trade, yet they are often underserved by the existing digital commerce infrastructure. They face a persistent challenge: how to establish a fully functional, branded online store without incurring high costs or relying on inflexible, third-party platforms.

The lack of affordable, technically accessible, and customizable e-commerce solutions creates a significant barrier to digital entry. Without the ability to manage their own platforms, these businesses are forced to compromise on user experience, lose control of customer data, and reduce profit margins due to commissions and fees from external marketplaces.

ShopEase directly addresses this challenge.

It offers a free, open-source, and full-stack web application designed specifically to empower small and medium businesses to launch and operate their own e-commerce platforms. Built using the MERN stack (MongoDB, Express.js, React.js, Node.js), ShopEase is:

- Technically robust yet easy to deploy
- Fully customizable for branding and business needs
- Scalable and extendable as the business grows
- Accessible to developers, startups, and students alike

By eliminating high upfront costs, simplifying deployment, and returning control to the business owner, ShopEase enables independent retailers to compete in the digital economy on their own terms — helping bridge the gap between opportunity and access.

Requirements

1. Software Requirement

These requirements specify the software dependencies and technologies needed to run the ShopEase platform.

- **Frontend:**
 - **React.js:** JavaScript library for building the user interface.
 - **Redux** (optional): For state management across the application.
 - **HTML5, CSS3:** For basic page structure and styling.
 - **Bootstrap** or **Material UI:** For responsive design and UI components.
- **Backend:**
 - **Node.js:** JavaScript runtime environment for building the server-side application.
 - **Express.js:** Framework for handling routes, middlewares, and HTTP requests.
 - **JWT (JSON Web Tokens):** For user authentication and session management.
 - **bcrypt.js:** For secure password hashing.
- **Database:**
 - **MongoDB:** NoSQL database to store product data, user information, orders, etc.
 - **Mongoose:** ODM (Object Data Modeling) library for MongoDB and Node.js.
- **API and Integrations:**
 - **Stripe** or **PayPal SDK:** For integrating payment processing.
 - **SendGrid** or **Mailgun:** For email notifications (e.g., order confirmations).
- **Development Tools:**
 - **Git:** Version control system to track changes.
 - **VS Code:** IDE for code editing.
 - **Postman:** For testing APIs.
- **Hosting and Deployment:**
 - **Heroku, Vercel, or DigitalOcean:** For cloud-based hosting.
 - **MongoDB Atlas:** Managed cloud database service.

2. System Requirement:

System requirements define the hardware and software configurations necessary to run the ShopEase platform.

Server-side Requirements:

1. Operating System:

- Linux (Ubuntu preferred), macOS, or Windows for development
- Linux (Ubuntu or CentOS) for production deployment

2. CPU:

- Minimum: Dual-core processor
- Recommended: Quad-core processor or higher

3. RAM:

- Minimum: 2GB
- Recommended: 4GB or more

4. Storage:

- Minimum: 10GB free space
- Recommended: 20GB or more (depending on expected traffic and data storage needs)

5. Network:

- Minimum: Broadband connection (for cloud-based systems)

Client-side Requirements:

1. Web Browser:

- Latest version of Chrome, Firefox, Safari, or Edge

2. Device Compatibility:

- Desktop, laptop, tablet, or smartphone with internet access

3. Functional Requirements

Functional requirements define the system's behavior and what the system should do.

1. User Authentication:

- Users must be able to register with an email and password.
- Users can log in and log out using email/password or third-party authentication (if integrated).
- Admins have access to a secure admin panel with role-based permissions.

2. Product Management:

- Admin users can add, update, or delete products.
- Each product must have a name, description, price, image, and category.
- Users should be able to view product details and search for products by category or keyword.

3. Shopping Cart:

- Registered users and guests can add products to the shopping cart.
- The cart should update in real-time with quantities, total price, and subtotal.
- Users can modify the quantity or remove items from the cart.

4. Checkout Process:

- Users can proceed to checkout, entering shipping and payment information.
- Payment options (Stripe/PayPal) should be available.
- An order confirmation should be sent to the user's email.

5. Order Management:

- Users can view past orders and track the status of current orders.
- Admins can view and manage all orders, update statuses, and handle order fulfillment.

6. Admin Panel:

- Admin users can view user registrations, orders, and product details.
- Admins can manage the entire inventory, including adding, editing, and removing products.

7. Search and Filters:

- Users can filter products by price range, category, and availability.

- A search function should allow users to find products quickly.

8. Responsive Design:

- The system should be fully responsive across desktop, tablet, and mobile devices.

4. Non-Functional Requirement

Non-functional requirements describe the system's qualities, such as performance, security, and scalability.

1. Performance:

- The system should load in under 3 seconds.
- The platform should handle at least 500 concurrent users without significant performance degradation.
- Pages should load quickly even with a large product catalog (fast page rendering, optimized images).

2. Scalability:

- The system should be scalable to accommodate future growth in both users and data.
- The database should be able to scale horizontally or vertically as needed.
- The platform should handle increasing traffic during sales events or promotions.

3. Availability:

- The system should have 99.9% uptime to ensure high availability.
- Proper failover mechanisms should be in place to handle server downtime.

4. Security:

- User passwords should be securely hashed using bcrypt.
- All data exchanges should be encrypted via HTTPS.
- Sensitive user data (e.g., payment information) should be stored in compliance with security standards like PCI-DSS.
- Regular vulnerability assessments and updates should be carried out to mitigate security risks.

5. Usability:

- The application should be intuitive and easy to use for both customers

and admins.

- The interface should be designed for simplicity and ease of navigation, with clear CTAs (Call to Actions).

6. Compliance:

- The platform should comply with applicable data protection regulations such as GDPR for user privacy and data security.
- The payment integration should comply with PCI DSS (Payment Card Industry Data Security Standards).

7. Backup & Disaster Recovery:

- The system should have regular automated backups to ensure that data is recoverable in the event of a disaster.
- Backup data should be stored securely and available for recovery within a reasonable time frame.

Safety requirements

☐ **Data Integrity:**

- Ensure data consistency during transactions and updates (e.g., orders, payments).
- Use transactional mechanisms to prevent data corruption in case of failure.

☐ **Backup and Recovery:**

- Regular backups of the database and user data.
- Implement disaster recovery procedures to restore data quickly.

☐ **Error Handling:**

- Graceful handling of errors without exposing system internals to users.
- Provide meaningful error messages to users while logging technical errors for developers.

☐ **System Failover:**

- Implement automatic failover mechanisms to ensure the platform remains available if one server fails.

Security requirements

☐ **Authentication:**

- Secure user authentication with **JWT** (JSON Web Tokens).
- Implement multi-factor authentication (MFA) for admins and sensitive roles.

☐ **Password Protection:**

- Store user passwords securely using **bcrypt.js** for hashing.
- Enforce strong password policies (minimum length, special characters, etc.).

☐ **Data Encryption:**

- Use **SSL/TLS encryption** for secure data transmission over HTTPS.
- Ensure sensitive user data, such as payment details, is encrypted both in transit and at rest.

☐ **Authorization:**

- Implement role-based access control (RBAC) to limit access to the system based on user roles (e.g., admin, customer).
- Prevent unauthorized actions like modifying or deleting products without proper permissions.

☐ **Input Validation:**

- Sanitize and validate all user inputs to prevent **SQL injection**, **XSS (Cross-site Scripting)**, and other injection attacks.

☐ **Regular Security Audits:**

- Perform regular security audits to identify vulnerabilities in the application.
- Keep software dependencies and packages up-to-date with the latest security patches.

□ **Payment Security:**

- Integrate payment gateways like **Stripe** or **PayPal**, ensuring compliance with **PCI-DSS** (Payment Card Industry Data Security Standard).

Stakeholder and User Description

1. Stakeholder and User Description

Stakeholders

1. Small and Medium Business Owners (Primary Stakeholders)

- Goal: Use ShopEase to establish an online store
- Interests: Low cost, customization, control over customer data, simple admin interface

2. End Users / Customers

- Goal: Browse products, add to cart, place orders
- Interests: Simple, fast, responsive UI; secure login; easy checkout

3. Project Developers

- Goal: Build, maintain, and extend ShopEase
- Interests: Clean architecture, modular codebase, documentation, scalability

4. Open-source Contributors

- Goal: Contribute improvements, features, or bug fixes
- Interests: Community recognition, easy-to-understand code, good issue tracking

5. System Admins / DevOps

- Goal: Deploy and maintain infrastructure
- Interests: Reliable hosting, easy deployment, monitoring

Types of Stakeholders:

- **Internal Stakeholder:** An internal stakeholder is a person, group or a company that is directly involved in the project.
- **External Stakeholder:** An external stakeholder is the one who is linked indirectly to the project but has significant contribution in the successful completion of the project.

User Roles & Descriptions

1. Admin User

- Add, edit, or remove products
- View and manage customer orders
- Control platform settings

2. Registered Customer

- Sign up and log in
- Browse products, manage Wishlist and cart
- Place orders and view order history

3. Guest User

- Browse products only (no purchase or cart access)
- Prompted to sign up or log in to proceed further

2. Identify the appropriate Process Model

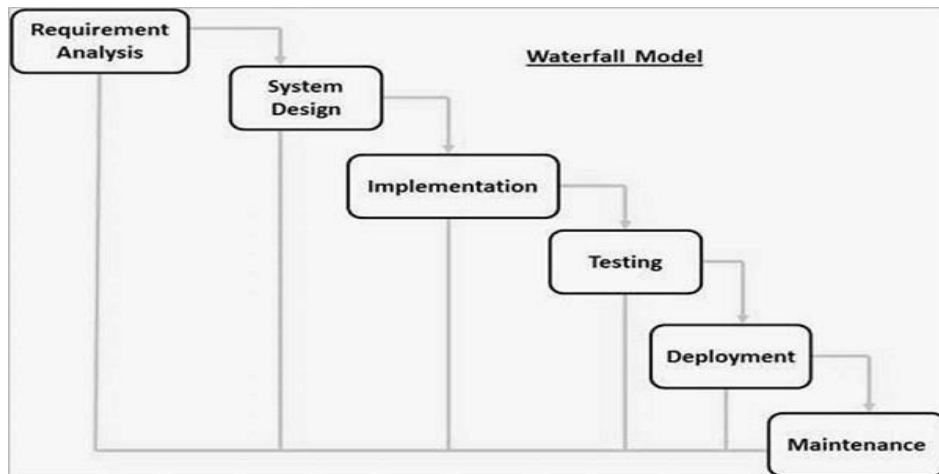
Recommended Process Model: Agile (Scrum) Why Agile?

ShopEase is a modern, full-stack web application being built in an open-source, collaborative environment. Agile is ideal for:

- Iterative development and feature expansion
- Continuous feedback from users and contributors
- Early delivery of a Minimum Viable Product (MVP)
- Adapting quickly to feature requests or bug reports

Agile Key Practices Applied to ShopEase:

- **Sprints:** Short 1–2-week cycles to deliver small increments (e.g., Cart Module, Authentication Module)
- **Daily Stand-ups:** For teams to sync up on progress, blockers, and goals
- **Product Backlog:** Prioritized list of features like "Add Wishlist", "Admin Panel", etc.
- **User Stories:** E.g., "As a customer, I want to add products to a Wishlist so I can purchase later."



3. Comparative study with Agile Model.

Aspect	Agile Model (Recommended)	Waterfall Model (Traditional)
Development Style	Iterative & Incremental	Sequential and Linear
Flexibility	Highly flexible — requirements can evolve	Rigid — requirements fixed early
User Involvement	Continuous feedback and involvement	User involved mostly at beginning and end
Delivery	Frequent small releases (MVP, iterations)	One-time full delivery at the end
Testing	Continuous testing within each sprint	Testing only after development phase
Best for	Dynamic projects with changing needs (like ShopEase)	Well-defined, stable requirements
Risk Management	Early detection of risks and errors	Late detection, higher risk impact

Objectives

1. **To develop a responsive and user-friendly e-commerce front-end interface** that works across devices.
2. **To utilize modern frontend technologies** like React, TypeScript, Tailwind CSS, and Vite for a fast and maintainable codebase.
3. **To create a scalable and modular architecture** that can be extended with backend services, authentication, and payment integration in the future.
4. **To simulate key e-commerce features** such as product display, shopping cart interface, and routing between pages.
5. **To practice real-world software engineering concepts** including UI design, component reusability, and application performance optimization.

Advantages

1. **Modern Tech Stack:** Uses industry-standard tools and frameworks like React, Vite, and TypeScript, making it relevant to current job markets.
2. **Responsive Design:** Tailwind CSS ensures the site adapts to various screen sizes and devices seamlessly.
3. **Scalable Codebase:** Component-based architecture makes it easy to add new features or modify existing ones.
4. **Faster Development:** Vite enables rapid development and hot module reloading for better developer experience.
5. **Clean UI:** Tailwind CSS allows for elegant and minimal design with quick customization options.
6. **Good Foundation for Full Stack Expansion:** The front-end is structured in a way that backend and APIs can easily be integrated.

Drawbacks

1. **No Backend Functionality:** Currently, it lacks real data handling, login systems, order processing, or database integration.
2. **No State Management System:** Lacks tools like Redux or Context API for managing application state in a scalable way.
3. **No Authentication or Authorization:** Security features like user login, admin roles, etc., are not implemented.
4. **Limited E-Commerce Logic:** Features like payment integration, order history, and dynamic inventory updates are not included.
5. **No Testing:** Absence of unit tests or integration tests makes it hard to validate functionality or prevent future bugs.

Future Scope

Project Justification:

Objective: Develop and deploy an open-source, customizable e-commerce platform designed for small and medium-sized businesses (SMBs) to easily create their own online stores. The platform will feature user-friendly interfaces for both customers and admins, full e-commerce functionality (product management, checkout process, payment integration), and scalability for growing businesses.

Key Features in Scope:

- **Frontend:**
 - User Registration and Authentication (Login/Sign-Up)
 - Product Browsing and Search
 - Shopping Cart and Checkout
 - Order History and Tracking
 - Responsive Design (Mobile/Desktop compatibility)
- **Backend:**
 - Product Management (CRUD operations for products)
 - User Management (Admin and Customer Roles)
 - Order Management (Creation, Tracking, and Management)
 - Payment Integration (Stripe/PayPal)
 - Security Features (Password hashing, JWT for authentication)
- **Admin Panel:**
 - Product, Order, and User Management
 - Analytics Dashboard (Basic Sales Analytics)
- **Scalability:**
 - Ensure the platform can handle future user growth and increased traffic.

Thus, all the manual complications in controlling the library have been rectified by implementing this computerization system. The system has been developed to control the behaviour, jobs and requirements of each category of users.

1. Find jobs and roles responsibilities

Project Manager (PM)

- Responsibilities:
 - Oversee the overall project development and execution.
 - Define project milestones, deadlines, and deliverables.
 - Manage communication between stakeholders and development team.
 - Handle risk management and mitigation strategies.

Business Analyst

- Responsibilities:
 - Define user requirements and business objectives.
 - Work with stakeholders to gather feature requests.
 - Translate business needs into technical specifications for the development team.

Frontend Developer

- Responsibilities:
 - Build the user interface using React.js, HTML5, CSS3.
 - Ensure responsive design and mobile compatibility.
 - Implement shopping cart, product browsing, and checkout UI components.

Backend Developer

- Responsibilities:
 - Build the server-side application using Node.js and Express.js.
 - Integrate database management with MongoDB.
 - Develop product, user, and order management APIs.
 - Handle payment gateway integration (e.g., Stripe, PayPal).

UI/UX Designer

- Responsibilities:
 - Create wireframes, prototypes, and visual designs.

- Focus on ease of use and accessibility.
- Design intuitive and user-friendly interfaces for both customers and admins.

Database Administrator (DBA)

- Responsibilities:
 - Design and maintain the MongoDB database structure.
 - Ensure database security, backups, and optimization.
 - Handle data migration and scaling as the user base grows.

Quality Assurance (QA) Tester

- Responsibilities:
 - Write and execute test cases for functional and non-functional requirements.
 - Perform user acceptance testing (UAT).
 - Identify bugs and work with developers to resolve issues.

Security Specialist

- Responsibilities:
 - Implement security protocols such as HTTPS, JWT, and role-based access control (RBAC).
 - Perform vulnerability assessments and penetration testing.

DevOps Engineer

- Responsibilities:
 - Set up and manage the deployment pipeline.
 - Handle cloud hosting and infrastructure setup (e.g., Heroku, AWS).
 - Manage server health, uptime, and scalability.

2. Calculate project based on resources

The project can be divided into multiple phases. Below is a high-level breakdown of the timeline, estimated resources, and job responsibilities per phase.

Phase 1: Requirements Gathering and Planning (1 Month)

- Activities:
 - Define business requirements
 - Finalize features and functionalities
 - Create wireframes and prototypes
 - Set project milestones and timelines
- Resources Required:
 - Project Manager
 - Business Analyst
 - UI/UX Designer

Phase 2: Frontend and Backend Development (2 Months)

- Activities:
 - Frontend: Build React.js components (Product Listing, Cart, Checkout, Order History)
 - Backend: Set up Node.js server, MongoDB database, and implement APIs for product, user, and order management
 - Security: Implement basic authentication and authorization features (JWT, password hashing)
- Resources Required:
 - Frontend Developer (2)
 - Backend Developer (2)
 - UI/UX Designer (1)
 - Security Specialist (1)

Phase 3: Admin Panel and Payment Integration (1 Month)

- Activities:
 - Develop Admin Panel for product, order, and user management
 - Integrate Stripe/PayPal for payments
 - Implement basic analytics dashboard
- Resources Required:
 - Backend Developer (1)
 - Frontend Developer (1)
 - Database Administrator (1)
 - Quality Assurance Tester (1)

Phase 4: Testing and Deployment (1 Month)

- Activities:
 - Perform Unit Testing, Integration Testing, and User

- Acceptance Testing (UAT)
 - Set up deployment pipeline (CI/CD)
 - Deploy the app on the cloud (Heroku, AWS)
- Resources Required:
 - QA Tester (1)
 - DevOps Engineer (1)
 - Backend Developer (1)

Phase 5: Post-Launch Support and Updates (Ongoing)

- Activities:
 - Monitor system performance and usage
 - Implement feedback from initial users
 - Roll out updates and improvements
 - Ensure platform scalability as users increase
- Resources Required:
 - Project Manager (1)
 - Backend/Frontend Developer (1)
 - DevOps Engineer (1)

4. Resource-Based Calculation

Team Members and Estimated Hours:

- Project Manager: 80 hours (across all phases)
- Frontend Developer: 300 hours (over 2 months)
- Backend Developer: 300 hours (over 2 months)
- UI/UX Designer: 100 hours (in the first 2 phases)
- Quality Assurance Tester: 120 hours (during testing phase)
- Security Specialist: 60 hours (during Phase 2 for security implementation)
- DevOps Engineer: 80 hours (during deployment phase)
- Database Administrator: 40 hours (in Phase 2 for database design and scaling)

Total Estimated Hours:

- Total Hours = $80 + 300 + 300 + 100 + 120 + 60 + 80 + 40 = 1,080$ hours

5. Cost Estimation

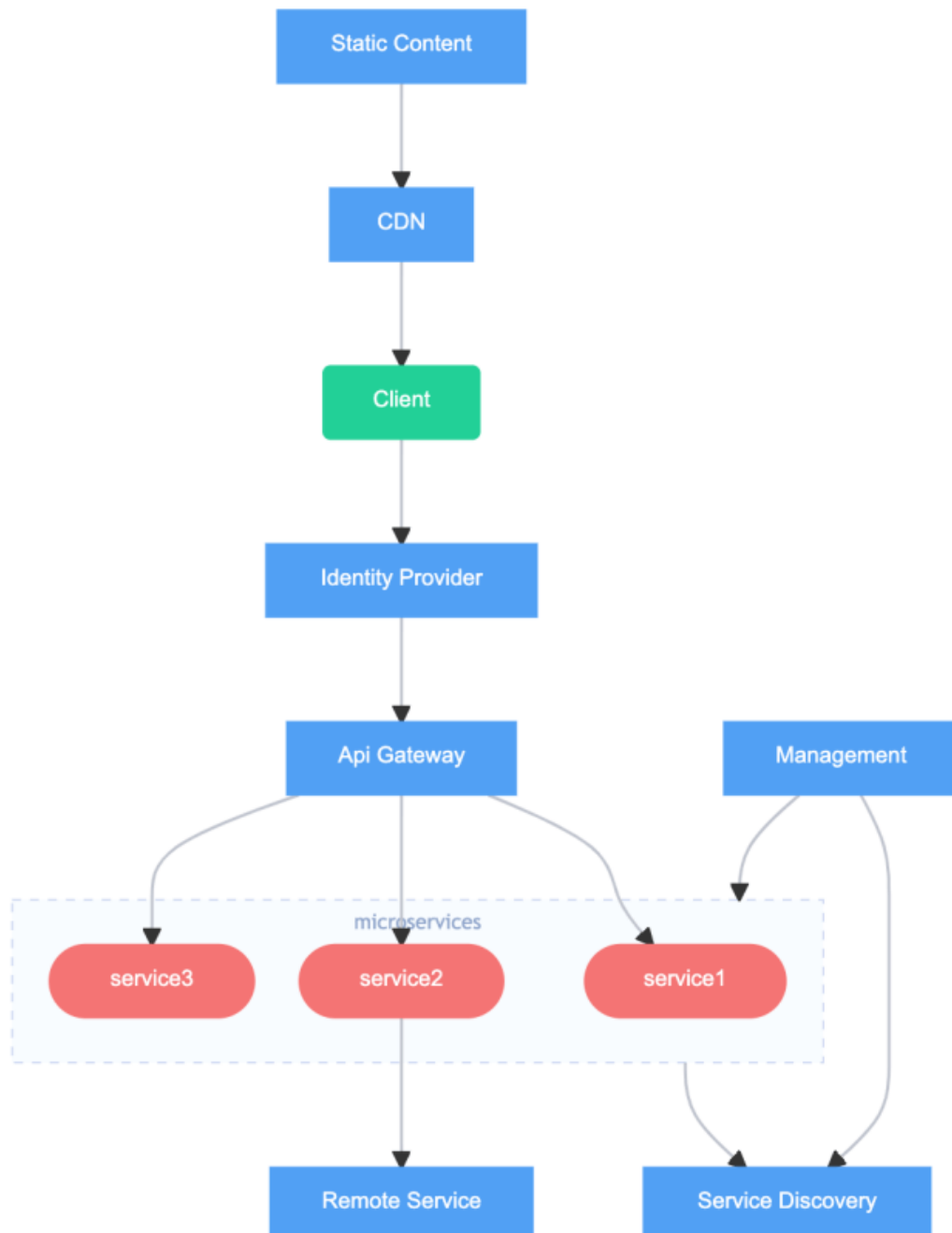
Assumptions:

- Hourly rates are averaged at \$40/hour for developers, designers, and testers, \$50/hour for Project Managers and DevOps Engineers.

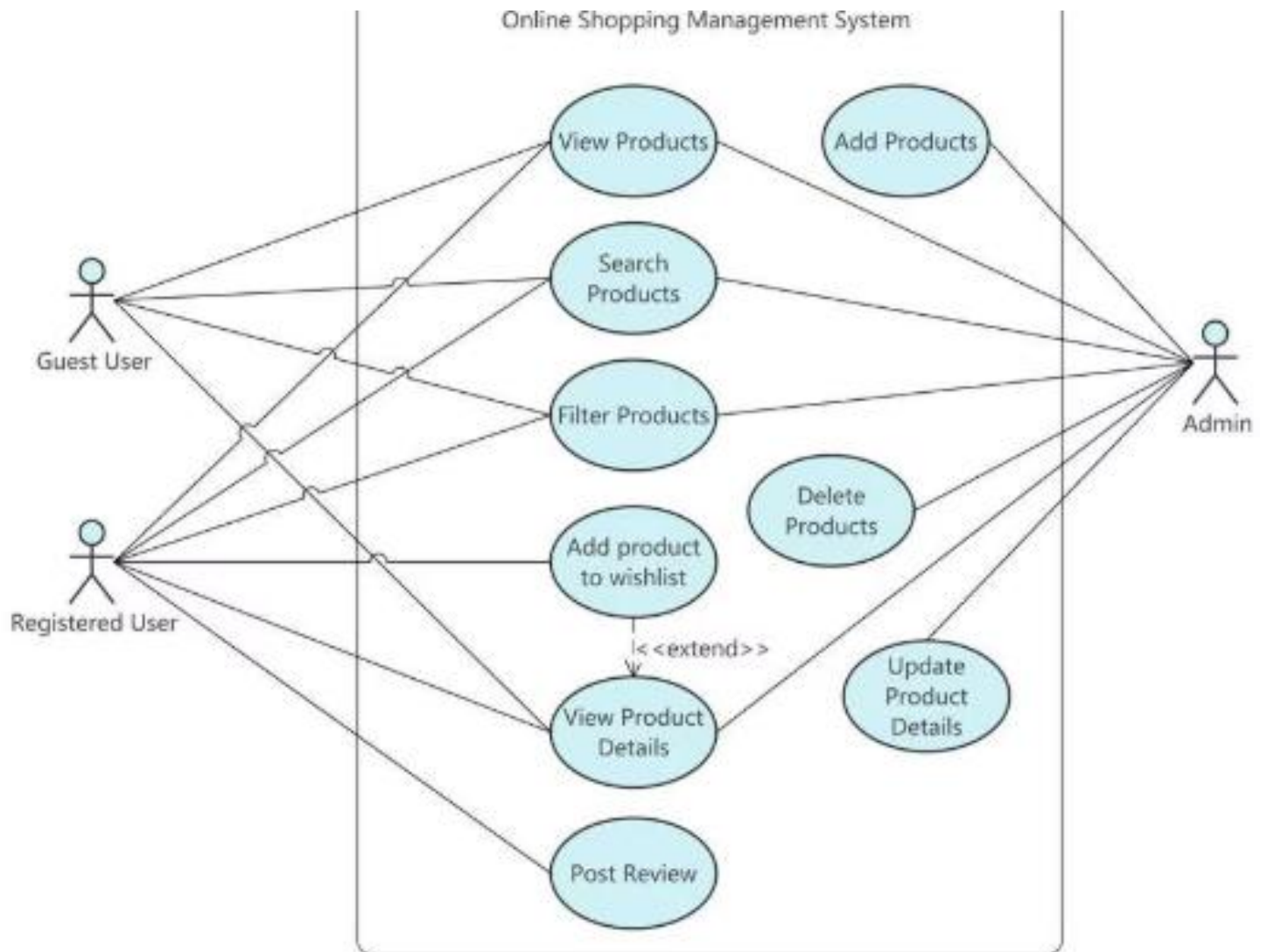
Total Cost = $1,080 \text{ hours} \times \$40/\text{hour} = \$43,200$

Software Design and System Modeling

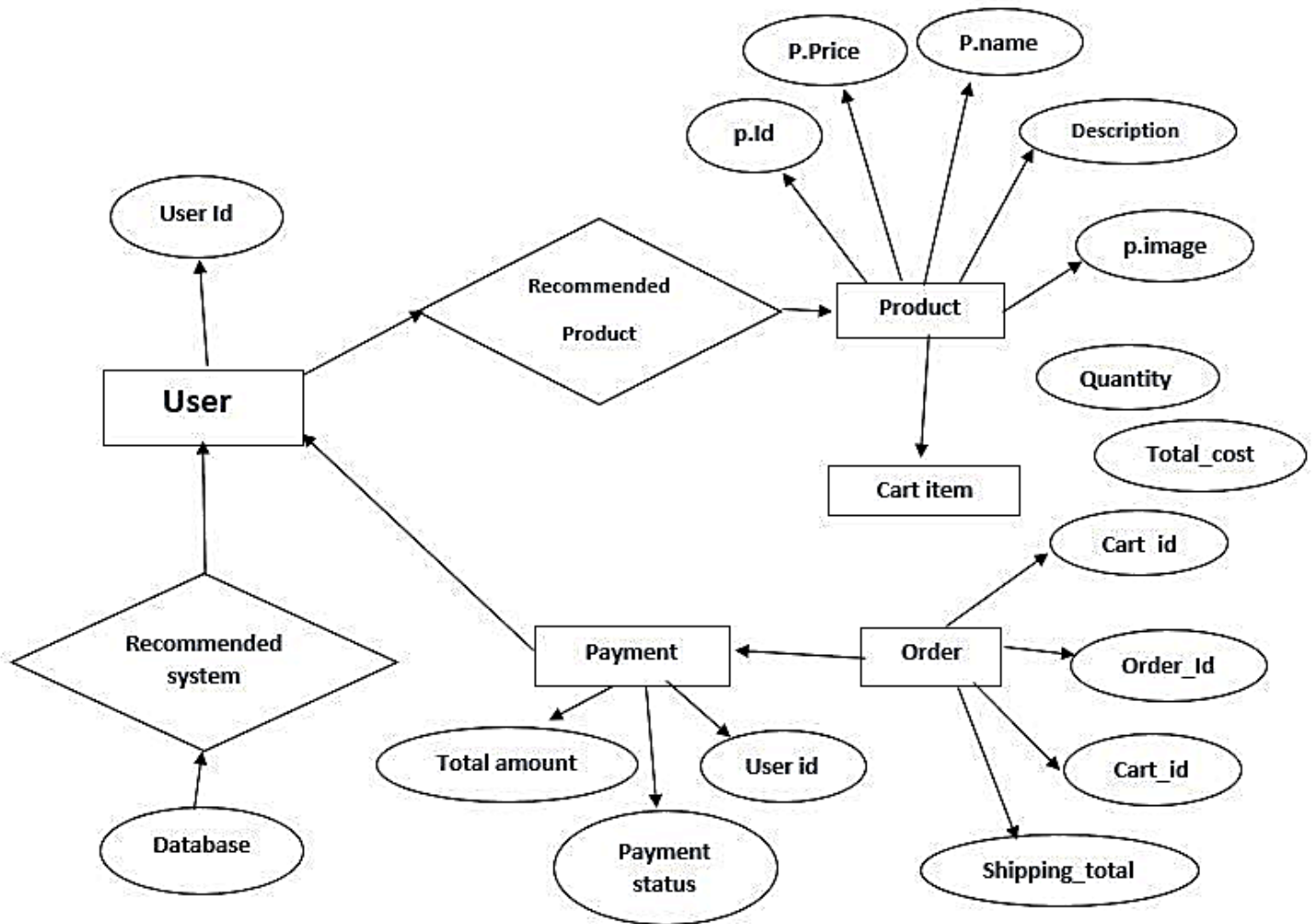
1. Design System Architecture



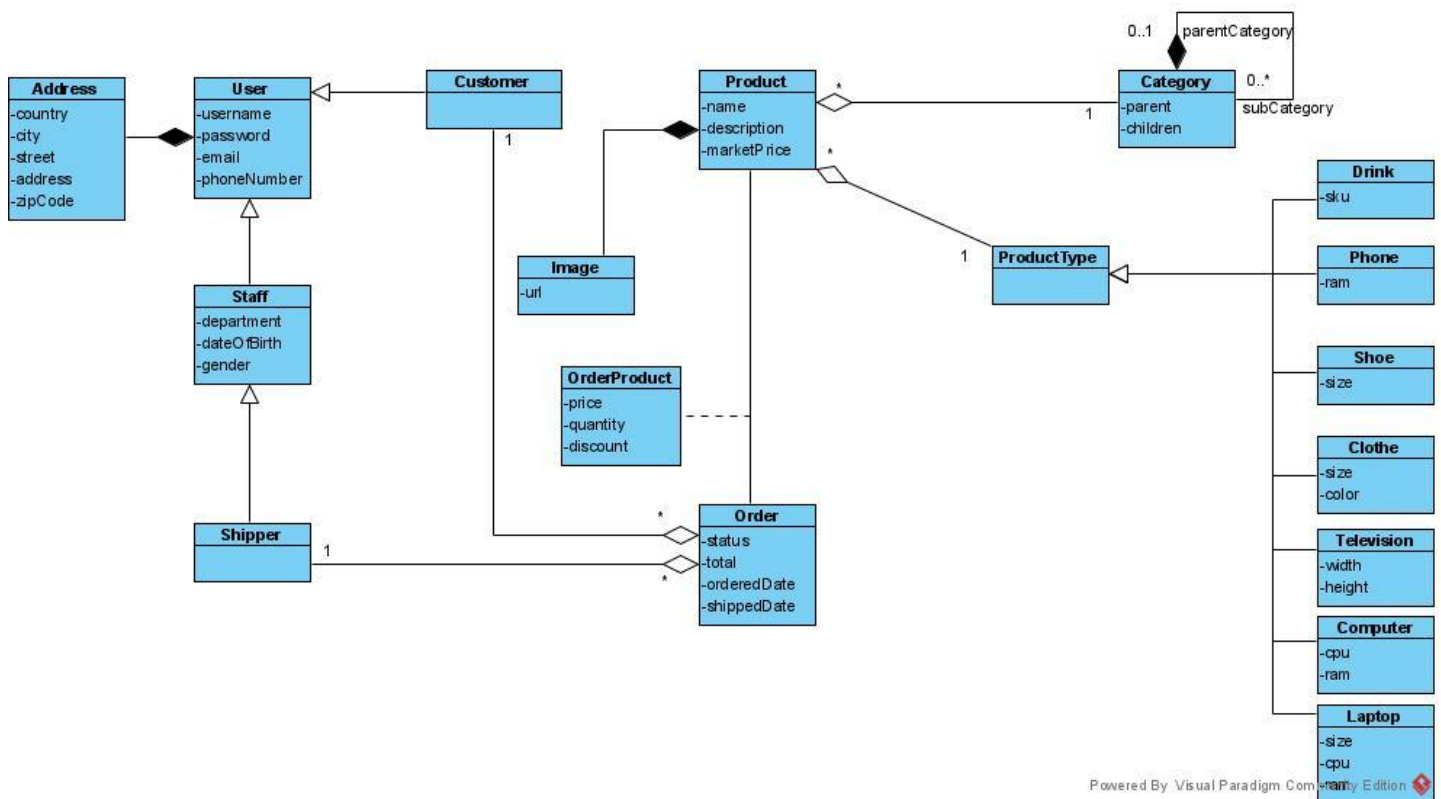
2. Use Case Diagram



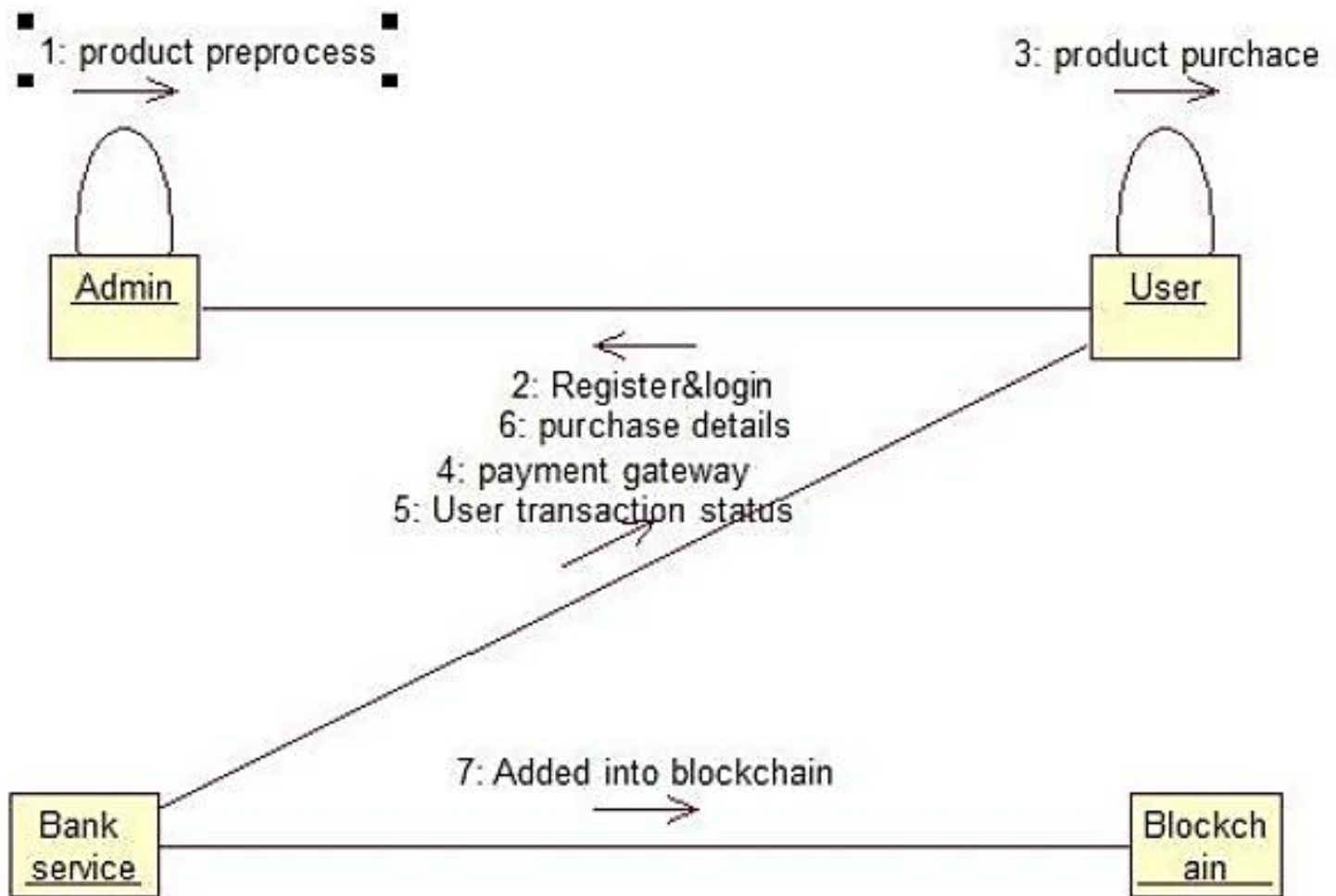
3. E-R Diagram



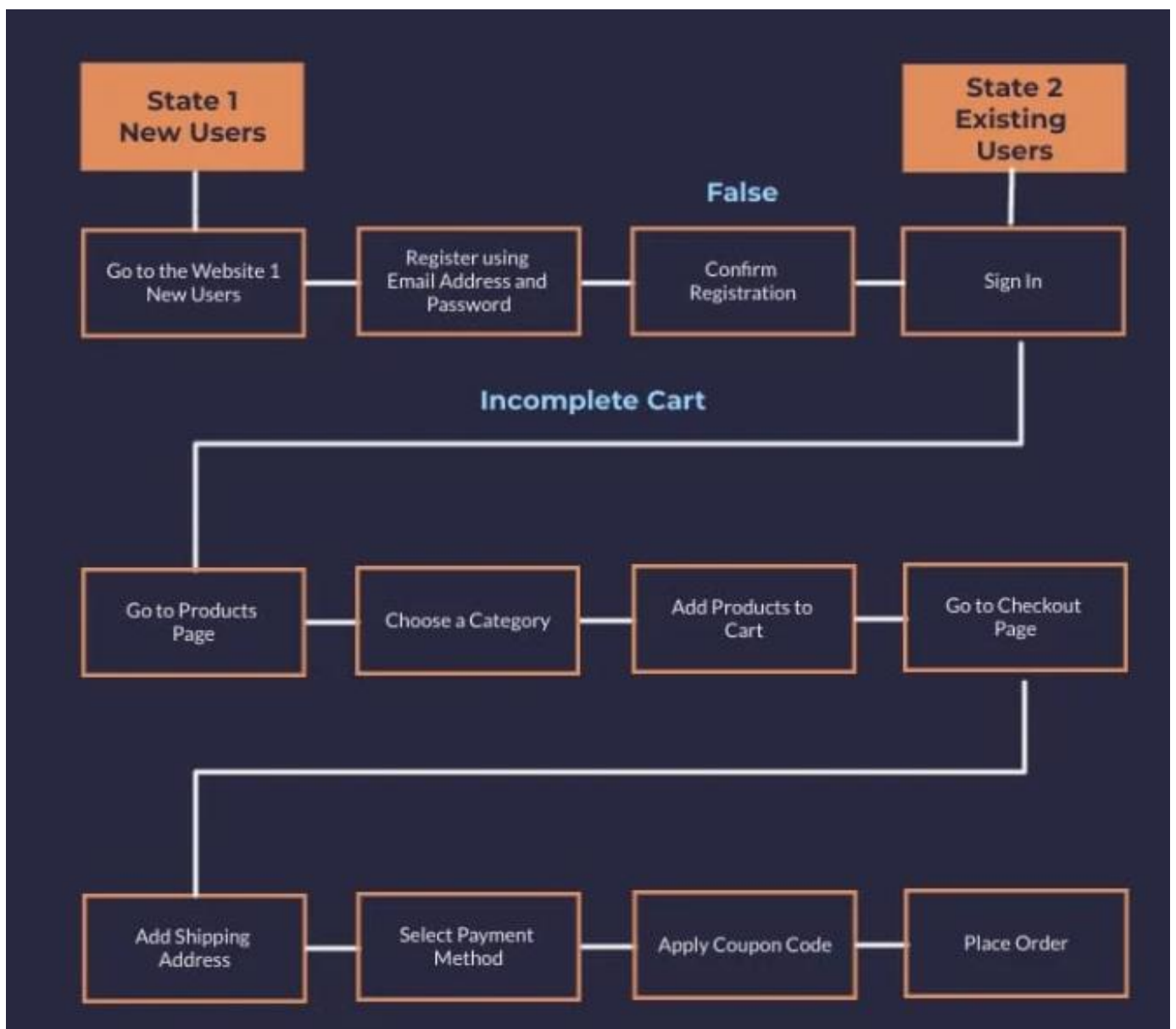
4. Class Diagram (Applied for OOPs based project)



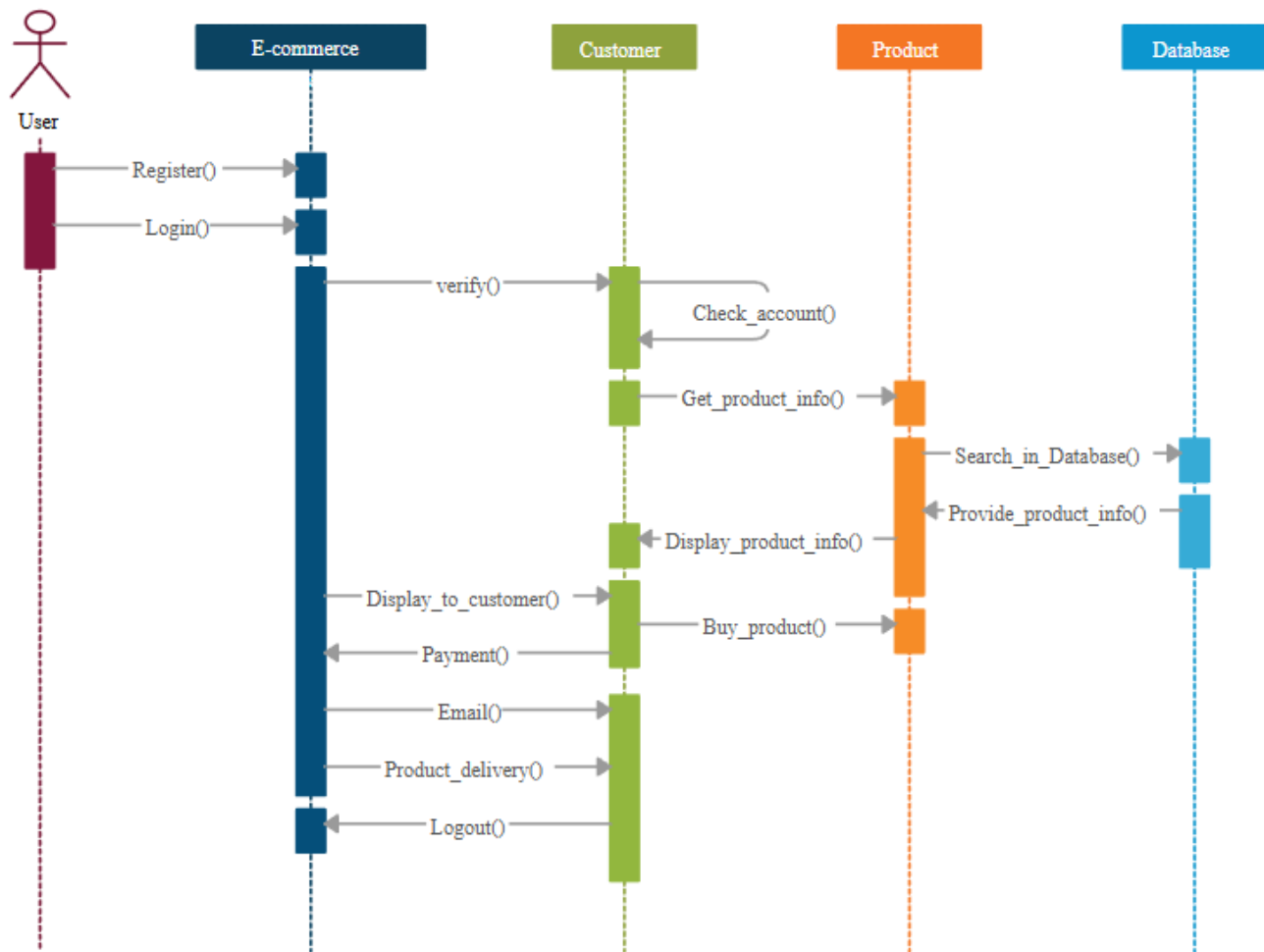
5. Collaboration Diagram (Applied For OOPS based Project)



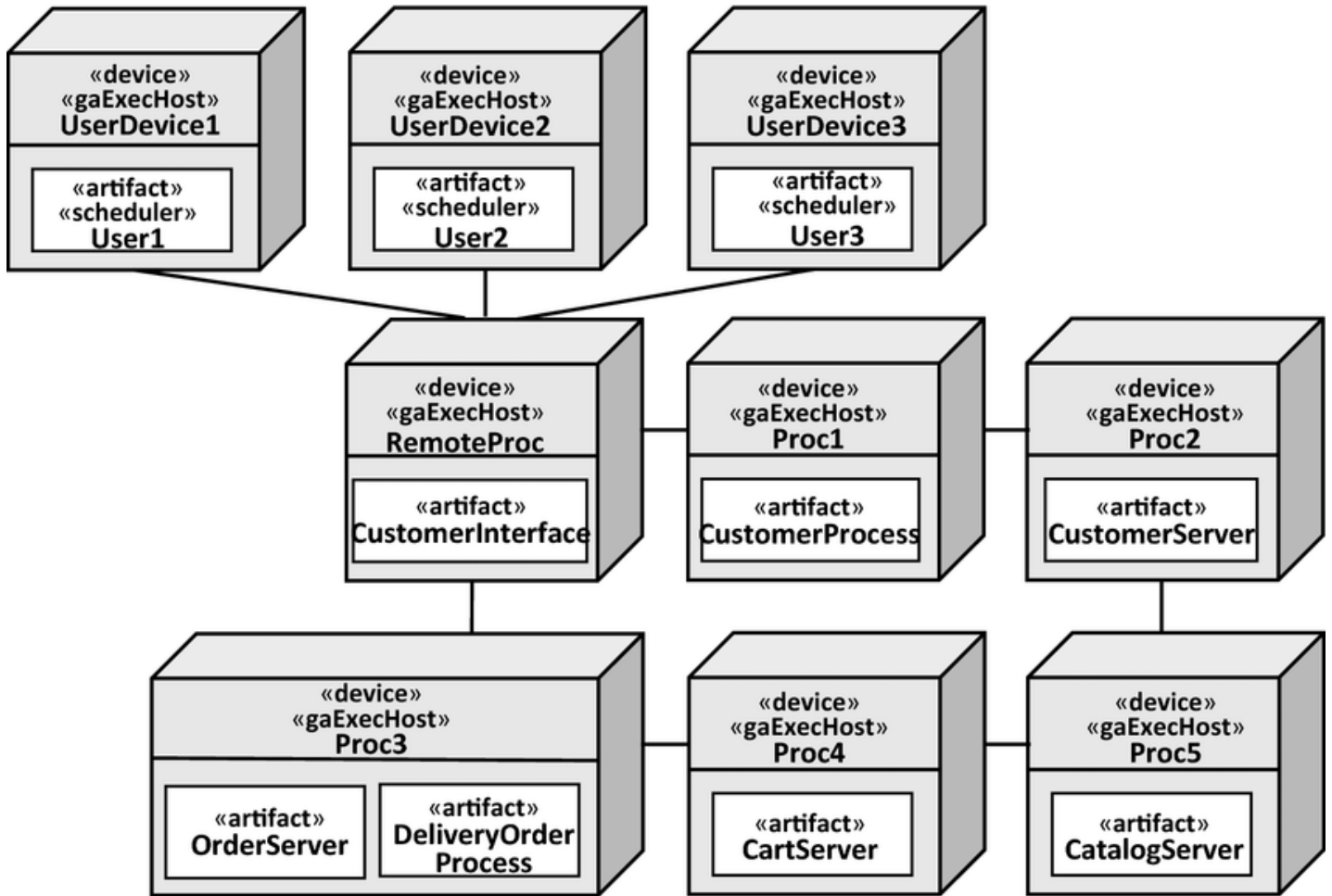
6. Design State Diagram



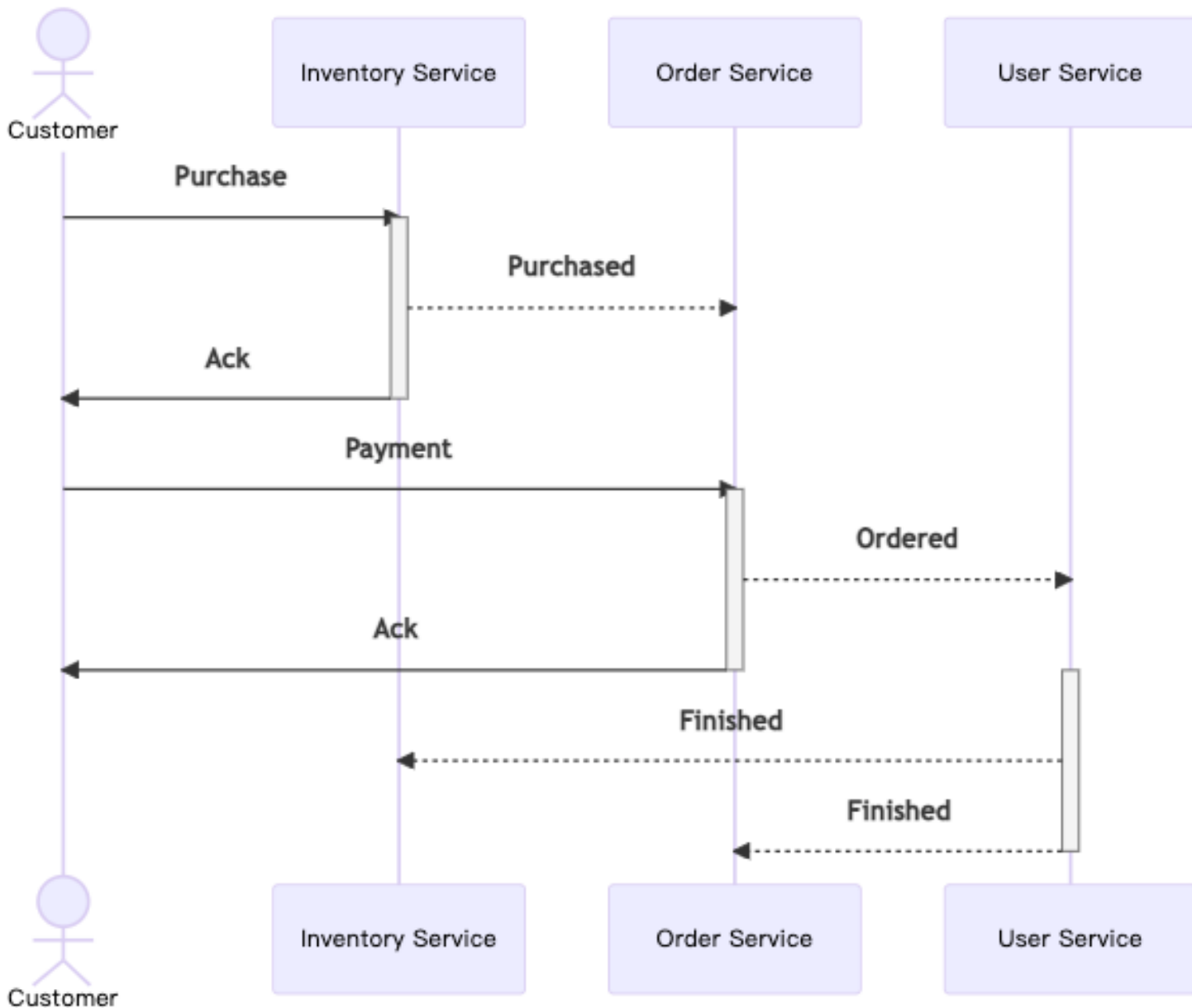
7. Design Sequence Diagram



8. Deployment Diagram



9. Sample Frontend Design



Implementation

1. Product Card Component (TypeScript + Tailwind)

```
// src/components/ProductCard.tsx

import React from "react";

interface Product {
  id: number;
  name: string;
  price: number;
  imageUrl: string;
}

const ProductCard: React.FC<{ product: Product }> = ({ product })
=> {
  return (
    <div className="border rounded-xl shadow-md p-4 transition
    hover:scale-105">

      <img src={product.imageUrl} alt={product.name} className="w-
      full h-48 object-cover rounded-md" />

      <h2 className="text-lg font-semibold mt-2">{product.name}</h2>

      <p className="text-green-600 font-bold text-
      sm">₹{product.price}</p>

      <button className="mt-2 w-full bg-blue-500 hover:bg-blue-600 text-
      white py-1 rounded">
        Add to Cart
      </button>

    </div>
  );
};
```

```
};
```

```
export default ProductCard;
```

2. Product List Page

```
// models/Order.js
```

```
// src/pages/Home.tsx
```

```
import React from "react";
```

```
import ProductCard from "../components/ProductCard";
```

```
const products = [  
  { id: 1, name: "Shirt", price: 499, imageUrl: "/images/shirt.jpg" },  
  { id: 2, name: "Shoes", price: 1299, imageUrl: "/images/shoes.jpg" },  
];
```

```
const Home: React.FC = () => {  
  return (  
    <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3  
gap-4 p-6">  
      {products.map((product) => (  
        <ProductCard key={product.id} product={product} />  
      ))}  
    </div>  
  );  
};
```

```
export default Home;
```

3. Cart Context (Global State with React Context API)

```
// src/context/CartContext.tsx
```

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

```
interface CartItem {
```

```
  id: number;
```

```
  name: string;
```

```
  price: number;
```

```
  quantity: number;
```

```
}
```

```
const CartContext = createContext<{
```

```
  cart: CartItem[];
```

```
  addToCart: (item: CartItem) => void;
```

```
  cart: [],
```

```
  addToCart: () => {},
```

```
});
```

```
export const useCart = () => useContext(CartContext);
```

```
export const CartProvider: React.FC<{ children: React.ReactNode }>  
= ({ children }) => {
```



```

const [cart, setCart] = useState<CartItem[]>([]);

const addToCart = (item: CartItem) => {
  setCart((prev) => {
    const exists = prev.find((p) => p.id === item.id);
    if (exists) {
      return prev.map((p) => p.id === item.id ? { ...p, quantity:
p.quantity + 1 } : p);
    }
    return [...prev, { ...item, quantity: 1 }];
  });
};

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

```

4. Cart Summary Component

```

// src/components/CartSummary.tsx
import React from "react";
import { useCart } from "../context/CartContext";

const CartSummary: React.FC = () => {

```

```

const { cart } = useCart();

const total = cart.reduce((acc, item) => acc + item.price *
item.quantity, 0);

return (
  <div className="p-4 border rounded-lg">
    <h2 className="text-xl font-semibold mb-2">Cart
Summary</h2>
    <ul className="text-sm">
      {cart.map((item) => (
        <li key={item.id}>
          {item.name} × {item.quantity} = ₹ {item.price *
item.quantity}
        </li>
      ))}
    </ul>
    <p className="mt-2 font-bold text-green-600">Total:
₹ {total}</p>
  </div>
);
};

export default CartSummary;

```

5. Routing with React Router

```

// src/App.tsx
import { BrowserRouter as Router, Routes, Route } from "react-
router-dom";

```

```

import Home from "./pages/Home";
import CartPage from "./pages/CartPage";

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/cart" element={<CartPage />} />
      </Routes>
    </Router>
  );
};
export default App;

```

6. package.json Sample Scripts

```

{
  "name": "shop-ease",
  "version": "1.0.0",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.14.0"
  },
  "devDependencies": {
    "vite": "^4.5.0",
    "typescript": "^5.3.3",
    "tailwindcss": "^3.3.2"
  }
}

```

OUTPUT:

ShopEase

Search products...

Categories Sign in

Sign In to Your Account

Email Address

Enter your email

Password

Enter your password

Sign in

Don't have an account? Sign up

For demo purposes, use:
Email: john@example.com
Password: password123

Create an Account

Full Name

123456

Email Address

123@GMAIL.COM

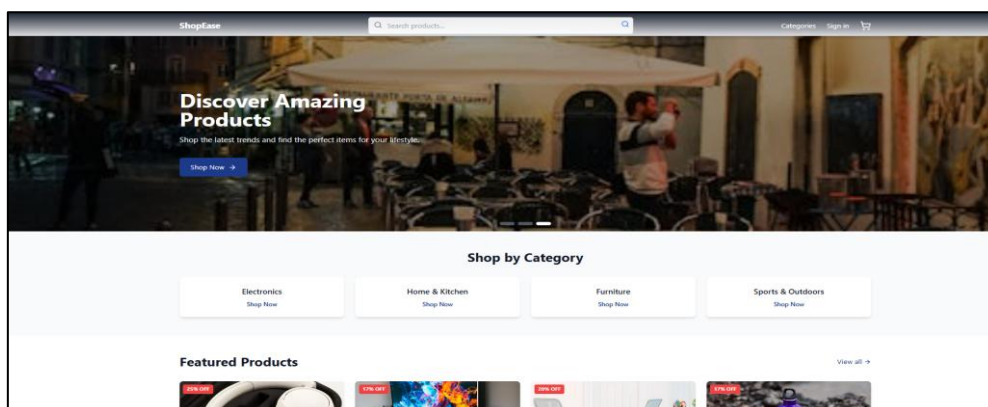
Password

.....

Create Account


Already have an account? Sign in

For demo purposes, use:
Email: john@example.com
Password: password123



Featured Products

20% OFF



4.7

Wireless Bluetooth Headphones


Premium noise-cancelling headphones with 20-hour battery life and superior sound quality.

₹12,374 ~~₹15,469~~

🛒

 Add to Cart

17% OFF



4.8

Ultra HD 4K Smart TV - 55"


Immerse yourself in stunning 4K resolution with this smart TV featuring HDR and built-in streaming apps.

₹41,249 ~~₹49,699~~

🛒

 Add to Cart

20% OFF



4.6

Ergonomic Office Chair


Work in comfort with this adjustable ergonomic chair designed for long hours of sitting.

₹16,499 ~~₹20,624~~

🛒

 Add to Cart

17% OFF



4.7

Stainless Steel Water Bottle

Eco-friendly insulated water bottle that keeps drinks cold for 24 hours or hot for 12 hours.

₹2,062 ~~₹2,474~~


🛒

 Add to Cart

View all →

New Arrivals

20% OFF



4.7


Wireless Bluetooth Headphones

Premium noise-cancelling headphones with 20-hour battery life and superior sound quality.

₹12,374 ~~₹15,469~~

🛒

 Add to Cart



4.5

Smart Fitness Watch


Track your fitness goals with this advanced smartwatch. Features heart rate monitoring, GPS...

₹12,374

🛒

 Add to Cart

17% OFF



4.8


Ultra HD 4K Smart TV - 55"

Immerse yourself in stunning 4K resolution with this smart TV featuring HDR and built-in streaming apps.

₹41,249 ~~₹49,699~~

🛒

 Add to Cart



4.3

Premium Coffee Maker

Start your day right with this programmable coffee maker that brews the perfect cup every time.

₹6,599

🛒

 Add to Cart

View all →

ShopEase

Search products...

Categories Sign in

Shop by Category

Electronics

Home & Kitchen

Furniture

Sports & Outdoors

Featured Collections

Summer Deals

Up to 50% off on selected items.

Shop Now →

New Arrivals

Check out our newest products.

Shop Now →

Clearance

Last chance to buy before they're gone.

Shop Now →


ShopEase

Search products...

Categories Sign in

Shopping Cart

1 Item



Stainless Steel Water Bottle

Eco-friendly insulated water bottle that keeps drinks cold for 24 hours or hot for 12 hours.

1

₹2,062

₹2,474

Clear Cart

Continue Shopping

Order Summary

Subtotal

₹2,062

Shipping GST (18%)

₹24.17

₹4,498

Total

₹853.658

Enter coupon code

Apply

Proceed to Checkout →

Secure payments powered by Stripe

37

Conclusion

The E-Commerce Website project successfully demonstrates the foundational structure and design of a modern online shopping platform using a robust front-end technology stack. By leveraging React, TypeScript, Tailwind CSS, and Vite, the application ensures a high-performance, modular, and responsive user interface that aligns with current industry standards.

The project exhibits a well-organized architecture, promoting scalability and ease of maintenance. Its use of utility-first styling (Tailwind CSS) and type safety (TypeScript) reflects adherence to best practices in frontend development. Although the current implementation primarily focuses on the client-side experience, it provides a solid base for further enhancements such as backend integration, user authentication, payment gateways, inventory management, and advanced analytics.

From a software engineering perspective, the system can be further analysed and extended using various modelling tools such as UML diagrams, ER diagrams, and DFDs, ensuring clear communication of system functionality and behaviour.

In summary, this project not only strengthens the understanding of building user-centric web applications but also provides a scalable blueprint for real-world e-commerce solutions. It serves as both a technical demonstration of frontend proficiency and a foundation for comprehensive full-stack development.