TRIBHUVAN UNIVERSITY

Institute of Science and Technology

Case Study on Online Shopping System

Submitted To:

Department of Information and Technology

Texas International College

In partial fulfillment of the requirements for the Bachelors of Science in Computer Science and Information Technology

Mr. Sunil Kumar Shrestha

Software Engineering(CSC-364*)*

Acknowledgments

We would like to express our gratitude and appreciation to all who helped us to complete this project report. We sincerely thank Mr. Sunil Shrestha, the subject instructor, for giving us the chance to meet this report as required coursework and for all of the encouragement, support, and help throughout. We also sincerely thank you for the time you spent proofreading and correcting our many mistakes.

We would also like to acknowledge with much appreciation the crucial role of the materials available online and the books referred to us by our subject teacher.

Members

- Abhi Kafle
- Amos Limbu
- Anisha Shrestha
- Anusha Maharjan
- Hangthim Limbu

Table of Contents

Introduction

The rise of smartphones and mobile devices revolutionized e-commerce, giving birth to mobile commerce or e-commerce. With the advent of mobile apps and responsive website design, customers gained the ability to shop conveniently on their mobile devices. Mobile payment systems, such as digital wallets and mobile banking, further accelerated the growth of e-commerce.

Online shopping is an application of the internet as e-commerce (Like Amazon). From the business perspective, customers usually find the products more attractive on websites, as they get all the details available there. People in large numbers are doing online shopping today, and it is not only because it is convenient as one can shop from home, but also because there is an ample number of varieties available, with high competition of prices, and also it is easy to navigate for searching regarding any particular item. This project aimed to develop a scalable and feature-rich e-commerce platform to meet the demands of the digital market. The platform included functionalities such as product listing, shopping cart, secure payment gateways, order management, and user profiles.

The proposed online shopping system will encompass the entire customer journey, from product browsing and selection to secure payment processing and order fulfillment.

Objectives

The objective of this project is to provide a user-friendly, secure, and scalable platform for customers to browse and purchase products online. This system has been designed keeping in mind all the aspects such as loading the data, complexity, and maintaining the security of user credentials.

The major objectives of the Online Shopping System are:

- To create an intuitive user interface that allows customers to browse products, add them to the shopping cart, and make secure online payments.
- To develop a product management system for sellers to add, edit, and update their product listings.
- To implement a secure authentication and authorization system for customers and sellers.
- To build an efficient search and filtering mechanism to enable customers to find products based on their preferences.
- To design a responsive website that adapts to different screen sizes and devices.
- To integrate popular payment gateways to facilitate secure and seamless online transactions.
- To attract a larger customer base and increase overall sales volume.
- To implement order management to ensure efficient order processing and delivery.

These objectives serve as a guide for the online shopping project, helping to align the development and implementation efforts with the overarching goals of the business or organization.

Overview of Online Shopping System

An online shopping system is a digital platform that enables customers to browse, select, and purchase products or services over the Internet. It provides a virtual marketplace where businesses can showcase their offerings, and customers can conveniently shop from anywhere, at any time.

The online shopping system consists of various components and features that work together to create a seamless and user-friendly shopping experience. Here is an overview of the key components:

- **Product Catalog:**
  The product catalog is a central database that stores information about the products or services available for purchase. It includes details such as product descriptions, prices, images, and variations. The catalog is organized into categories and subcategories to facilitate easy navigation for customers.

- **User Registration and Authentication:**
  The online shopping system allows users to create accounts, providing them with personalized features and a secure environment for transactions. User registration involves capturing essential information, such as name, email address, and password. Authentication mechanisms, such as username/password or social media log in, verify the user's identity when logging in.

- **Search and Navigation:**
  The system provides search functionality and intuitive navigation options to help customers find desired products efficiently. Customers can search by keywords, apply filters based on specific criteria (e.g., price range, brand), and browse through different product categories.

- **Product Detail Pages:**
  Each product has a dedicated detail page that provides comprehensive information about the item. This includes product descriptions, specifications, customer ratings, related products, and any promotional offers. High-quality product images or videos are also displayed to give customers a clear understanding of the product's appearance.

- **Shopping Cart:**
  The shopping cart is a virtual container where customers can add selected products before proceeding to the checkout process. It allows customers to review their selections, update quantities, and remove items if necessary. The cart also calculates the total order value, including taxes and shipping fees.

- **Checkout Process:**
  The checkout process guides customers through the final steps of making a purchase. It typically includes entering billing and shipping information, selecting a payment method (e.g., credit card, E-sewa), and confirming the order. The system validates the entered information and securely processes the payment to complete the transaction.

- **Order Management:**
  Once an order is placed, the online shopping system manages the order's lifecycle. It generates order confirmations and provides order tracking information to customers. The system integrates with inventory management systems to ensure product availability and initiate fulfillment processes, including packaging, shipping, and delivery.

- **Payment Gateway Integration:**
  Payment gateway integration enables secure and seamless online transactions. It facilitates the transfer of payment information between the customer, the online shopping system, and the payment processor. The system supports various payment methods, ensuring a wide range of options for customers.

- **Customer Support:**
  The online shopping system may include customer support features, such as email support. These features enable customers to seek assistance, ask questions, and resolve issues related to their purchases or general inquiries.

- **Security and Data Protection:**
  Online shopping systems prioritize the security and protection of customer data. They implement encryption protocols, secure communication channels, and compliance with data protection regulations. Measures are taken to ensure that sensitive information, such as payment details, is stored and transmitted securely.

An effective online shopping system combines these components to provide a seamless and enjoyable experience for customers while offering businesses the ability to showcase their products, and generate sales

Workflow Diagrams of Online Shopping System

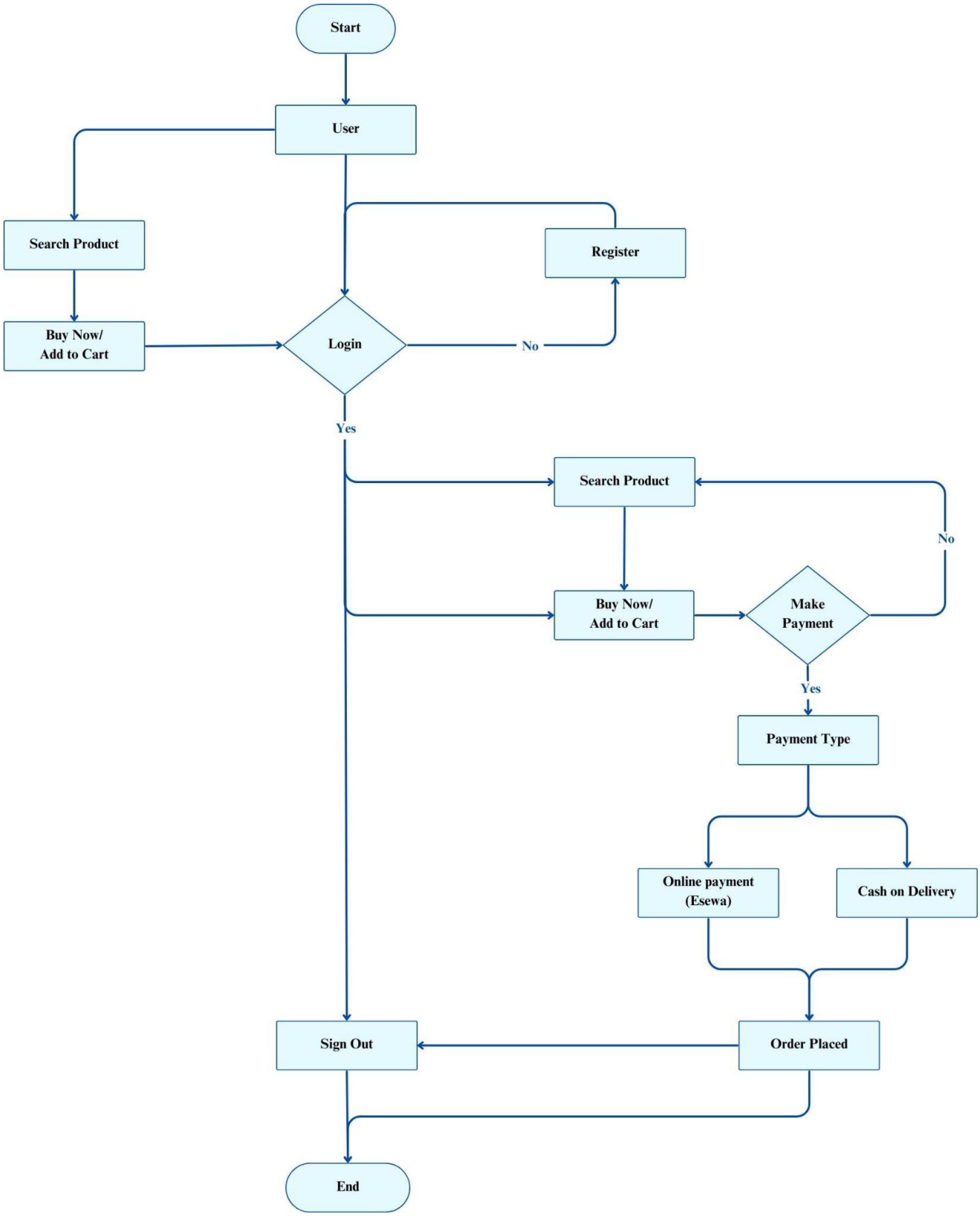


***Fig**: Workflow diagram of Online Shopping System*

Propose of System

The purpose of an online shopping system is to make shopping easier and more convenient for customers. It allows them to browse and buy products or services from anywhere, at any time. The system offers a wide selection of products, provides detailed information and customer reviews, and supports various secure payment options. Customers can track their orders and enjoy personalized recommendations. For businesses, the system expands their market reach, increases sales, and improves operational efficiency.

Actors

In an online shopping system, there are typically three main actors or entities involved:

- **Users:**
  Users are the primary actors in the online shopping system. They visit the website or platform, browse through products or services, add items to the shopping cart, and make purchases. Customers create accounts, provide personal information, and initiate transactions.

- **Seller and Admin**
  Sellers are the businesses or individuals who offer products or services for sale through the online shopping system. They create and manage product listings, set prices, update inventory, and handle order fulfillment.
  System admins are responsible for managing and maintaining the online shopping system. They oversee the technical aspects of the platform, ensure its proper functioning, and handle system updates and security measures.

Additionally, there may be other actors or entities involved in the online shopping system, depending on its specific features and functionalities. These can include:

- **Payment Gateway Providers:**
  Payment gateway providers are third-party services that facilitate secure online transactions. They integrate with the online shopping system to process payments and handle the transfer of funds between customers, merchants, and financial institutions.

- **Delivery and Logistics Providers:**
  In cases where physical products are involved, delivery and logistics providers play a role in the online shopping system. They handle the packaging, shipping, and delivery of products from Sellers to customers.

- **Customer Support Representatives:**
  Customer support representatives are responsible for addressing customer inquiries, concerns, and issues related to the online shopping system. They provide assistance through various channels such as live chat, email, or phone, helping customers with order tracking, returns, exchanges, and general inquiries.

These actors work together within the online shopping system to ensure a smooth and efficient shopping experience for customers and merchants alike.

Functional requirements

Functional requirements are a part of requirement analysis (also known as requirements engineering), which is an interdisciplinary field of engineering that concerns the design and maintenance of complex systems. It describes the desired end function of a system operating within normal parameters, so as to assure the design is adequate to make the desired product and the end product reaches its potential of the design in order to meet user expectations.

Here are the functional requirements of the online shopping system:

- **User Registration and Authentication:**
  The system should allow users to register for an account and authenticate their identity to access personalized features and make purchases.

- **Product Catalog Management:**
  The system should provide functionality for merchants to add, update, and manage product listings. Customers should be able to search and browse the product catalog effectively.

- **Shopping Cart:**
  The system should allow customers to add products to a shopping cart, view the cart, update quantities, and remove items before proceeding to checkout.

- **Order Placement and Processing:**
  Customers should be able to place orders, select shipping methods, and provide billing and shipping information. The system should process the order, generate order confirmations, and initiate the fulfillment process.

- **Payment Processing:**
  The system should support secure payment processing, integrating with payment gateways to facilitate various payment methods, such as credit cards, debit cards, and digital wallets.

- **Customer Ratings:**
  The system should allow customers to provide feedback, ratings, and reviews for products or services they have purchased.

- **Customer Support:**
  The system should provide customer support features, such as contact page and email support to assist customers with inquiries, returns, and other support-related issues.

Non-Functional Requirements

Non-functional requirements in software engineering refer to the characteristics of a software system that are not related to specific functionality or behavior. They describe how the system should perform, rather than what it should do. Here are the non-functional requirements:

- **Performance:**
  The system should be capable of handling a large number of concurrent users and transactions, ensuring fast page load times and responsiveness to provide a seamless shopping experience.

- **Security:**
  The system should employ security measures to protect customer data, including encryption of sensitive information, secure communication channels, and compliance with relevant data protection regulations.

- **Usability and User Experience:**
  The system should have an intuitive user interface, easy navigation, and responsive design to enhance the user experience and ensure accessibility across different devices and screen sizes.

- **Scalability:**
  The system should be scalable to accommodate growth in users, products, and transactions without compromising performance and responsiveness.

- **Availability:**
  The system should have high availability and uptime, minimizing downtime and ensuring continuous access for customers to browse, shop, and make purchases.

- **Integration:**
  The system may need to integrate with external services, such as payment gateways, inventory management systems, or shipping providers, to streamline operations and provide seamless customer experiences.

- **Data Backup and Recovery:**
  The system should have mechanisms in place for regular data backups and disaster recovery procedures to ensure data integrity and minimize the risk of data loss.

- **Compliance:**
  The system should adhere to legal and regulatory requirements, such as data protection laws, privacy policies, and consumer protection regulations.

These functional and non-functional requirements collectively define the features, performance, security, and usability aspects of an online shopping system, ensuring its effectiveness and meeting the expectations of customers and merchants.

Hardware Platforms

Hardware platforms refer to the physical devices or components on which software systems or applications run. These platforms provide the necessary computational resources and infrastructure for executing software programs. Hardware platforms can vary in terms of their capabilities, specifications, and form factors.

This system shall work on most laptops and PCs and supports React.js, Tailwinds, Node.js, and MYSQL.

Processor	Intel i3, RYZEN5, or better
RAM	4GB
Hard Disk	Min 4GB

Software Platforms

Software platform refers to the underlying framework or environment on which software applications or systems are developed, deployed, and executed. It includes the combination of software tools, libraries, programming languages, operating systems, and other infrastructure components that support the development and execution of software.

Software platforms provide a standardized and structured environment for developers to create and deploy software applications. They offer a set of APIs (Application Programming Interfaces), services, and tools that facilitate the development process and provide a foundation for building and running software.

This system will run on Mozilla Firefox, Windows Edge, Google Chrome, etc.

Operating System	Windows, MacOS, Linux, Android, IOS
Database	MySQL
Front-end	React.js, Tailwinds, JavaScript
Back-end	Node.js

Requirements Analysis

Requirement analysis in an online shopping system involves understanding and defining the specific requirements of the system to ensure it meets the needs of users, customers, and stakeholders. The objective of requirement analysis is to understand and define what the software system needs to do in order to meet the desired objectives and fulfill the needs of stakeholders.

Requirement analysis should involve thorough stakeholder engagement, understanding user needs, considering industry best practices, and ensuring the system meets security, performance, usability, and scalability requirements.

System Models

In an online shopping system, system models are representations or abstractions of a software system that help in understanding, designing, and analyzing its various aspects. System models provide a visual or conceptual representation of the system's structure, behavior, and interactions within its environment. Here are some commonly used system models:

ER Diagram

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes, and relationships

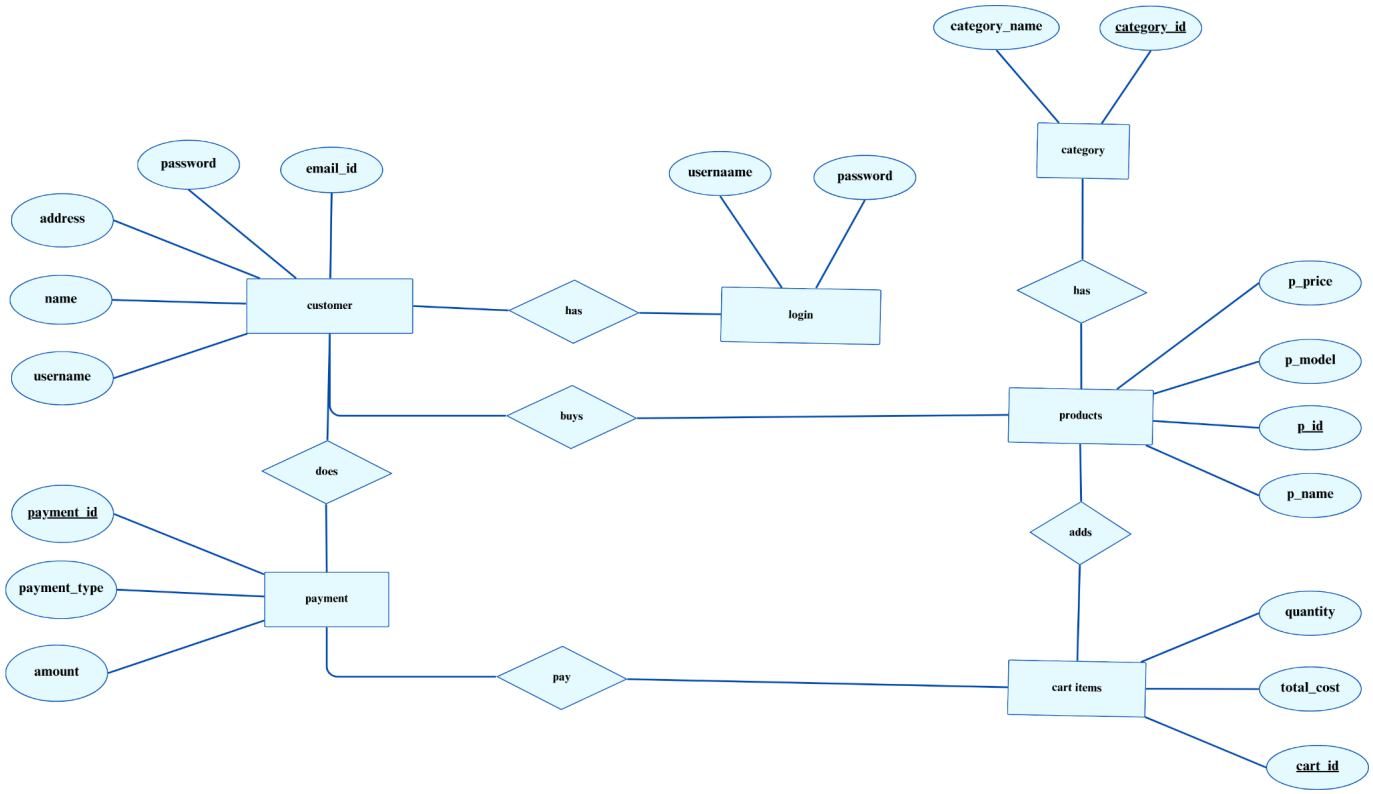


***Fig**: ER Diagram of Online Shopping System*

Context Diagram

A Software System Context Model is a type of system context model that explicitly depicts the boundary between the software system and *its* external environment: the hardware devices that the software system interacts with in order to engage with the environment.

The process of establishing the analysis framework by drawing and reviewing the context diagram inevitably involves some initial discussions with users regarding problems with the existing system and the specific requirements for the new system. These are formally documented along with any specific system requirements identified in previous studies.

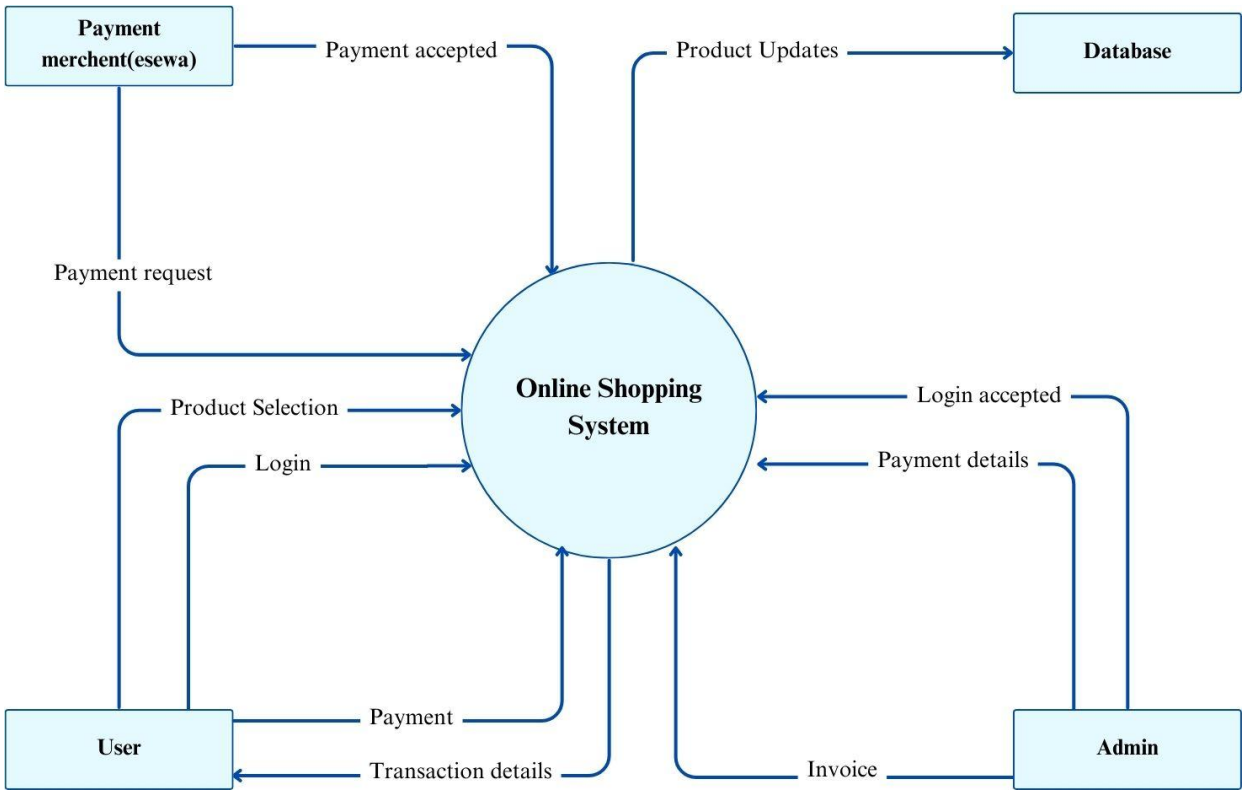


Fig: Context Model of Online Shopping System

The "Online Shopping System" represents the main system being developed. It encompasses all the components and functionalities of the online shopping system, including user interfaces, product catalogs, shopping carts, payment merchants, order management, and other relevant features.

The arrows indicate the flow of information and interactions between the users, payment merchants, admins, and the online shopping system. For example, the user can browse the product catalog, add items to the cart, proceed to checkout, make payments, and manage their account.

The context model provides a high-level overview of the system's boundaries and its interactions with external entities, setting the stage for further detailed analysis and modeling of the online shopping system.

Use Case Diagram

A use case model is a visual representation that captures the functionalities and interactions of a system from the perspective of its users. It is a key component of requirements analysis and system design in software engineering. The use case model describes the system's behavior by identifying and documenting the various use cases and actors involved.

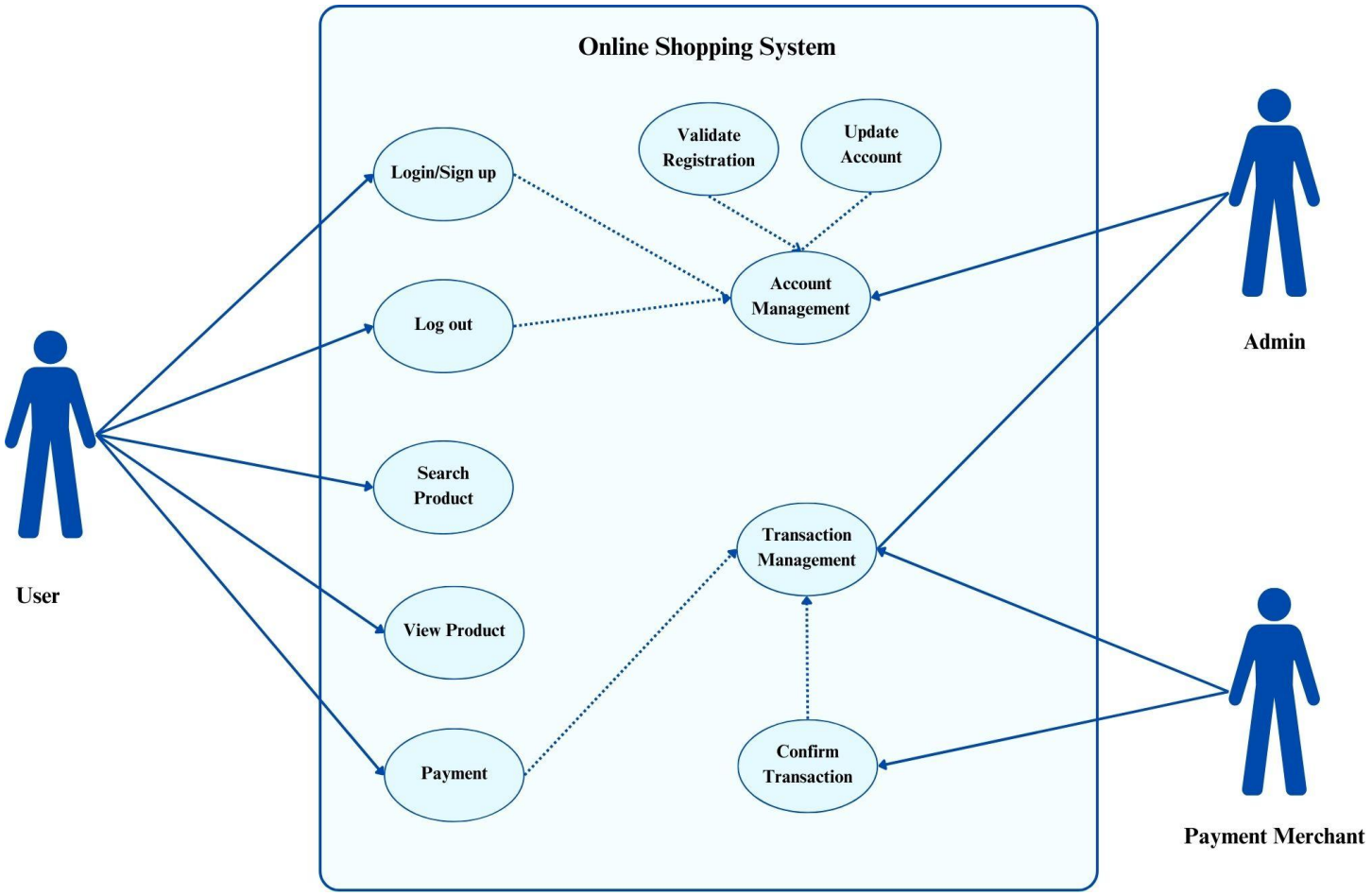


Fig: Use Case Model of Online Shopping System

Online Shopping System: Login	
Actors	Users/Customers
Descriptions	Users Log into the system by their authorized credentials
Data	Passwords and username
Stimulus	Credential input issued by user
Response	Authorized entry into the system after validating the credentials
Comment	Users must enter correct credentials to be given access to the system

Online Shopping System: Search Product	
Actors	Users/Customers
Descriptions	Users Search into the system by their keywords
Data	Keywords
Stimulus	Search input issued by user
Response	Returns and Display the product as per the input request
Comment	Users must enter correct keywords to display the respective item

Online Shopping System: Buy Now	
Actors	Users/Customers
Descriptions	Users selects the product and view the detail
Data	Product Details
Stimulus	Product selection input issued by user
Response	Returns the detail page consisting of the product details
Comment	Users must select correct product to view its detail

Online Shopping System: Check Out	
Actors	Users/Customers
Descriptions	Users selects the payment method
Data	Payment Details and Confirmation
Stimulus	Payment selection input issued by user
Response	Returns the payment page consisting of the pricing details
Comment	Users must input proper payment details and payment methods

Class Diagram

Class diagram is a static diagram. It represents the static view of an application. The class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

The purpose of a class diagram is to model the static view of an application. Class diagrams are the only diagrams that can be directly mapped with object-oriented languages and are thus widely used at the time of construction.

The class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.
Classes are Admin, Customer, Order, Items, Payment, and ShippingInfo.

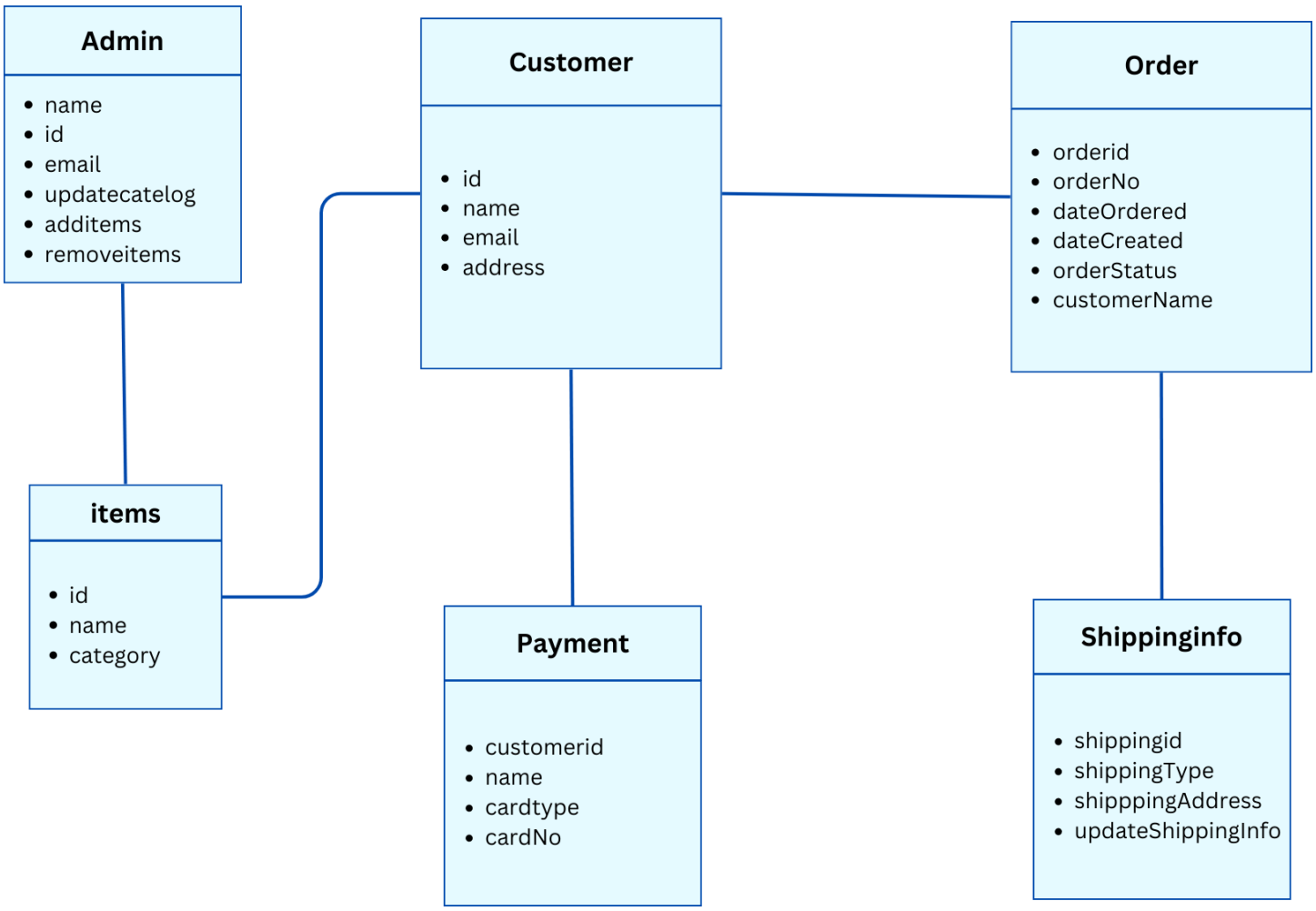


Fig: Class diagram of Online Shopping Model

Collaboration Diagram

It shows the object organization as seen in the following diagram. In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

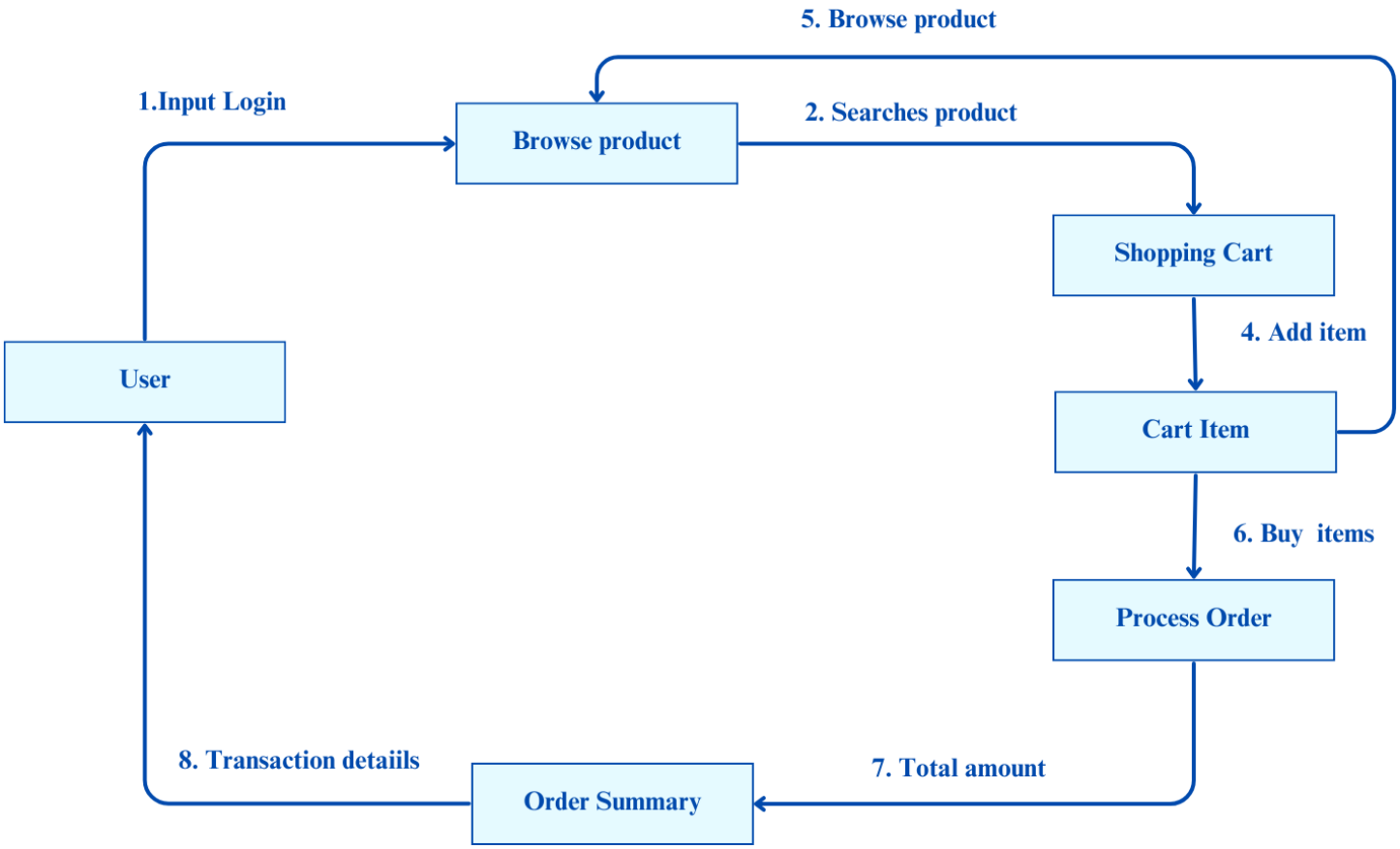


Fig: Collaboration Diagram of Online Shopping Model

Sequence Diagram

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent the time what messages are sent and when.

A sequence diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

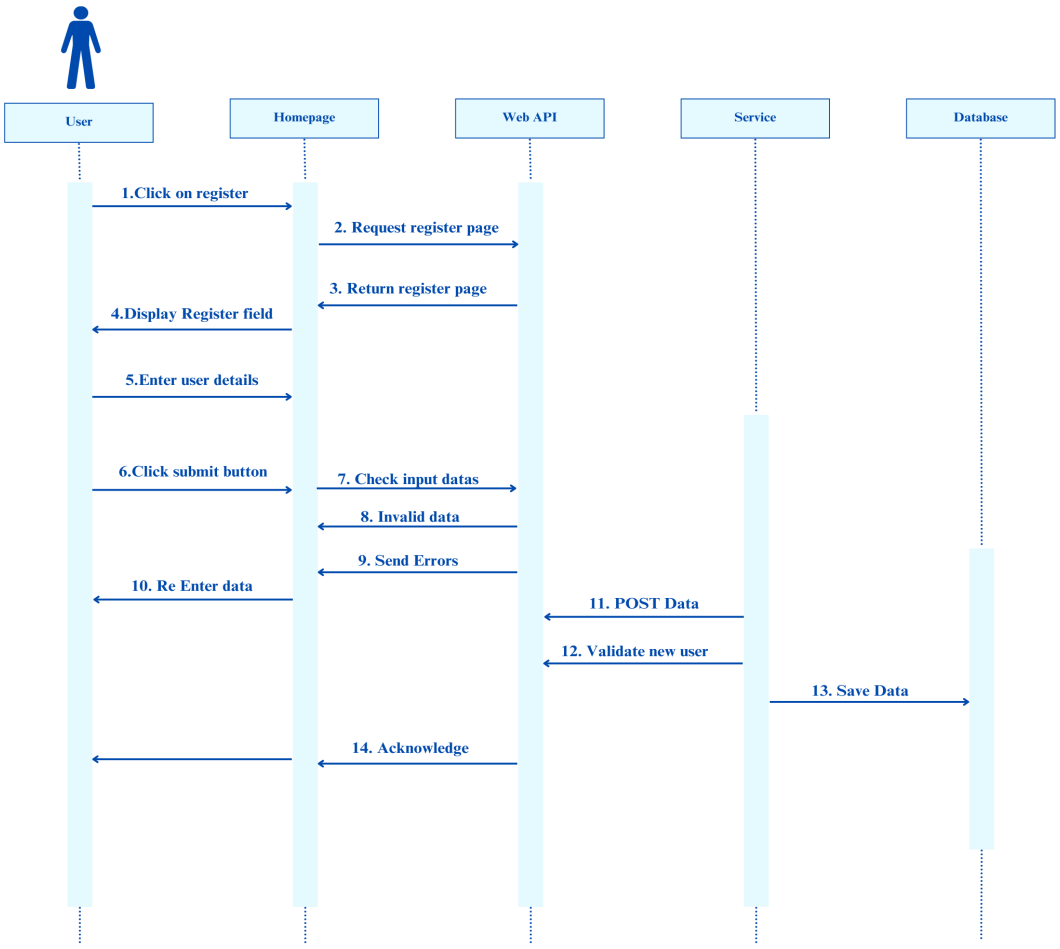


Fig: Sequence Diagram of Login/Register

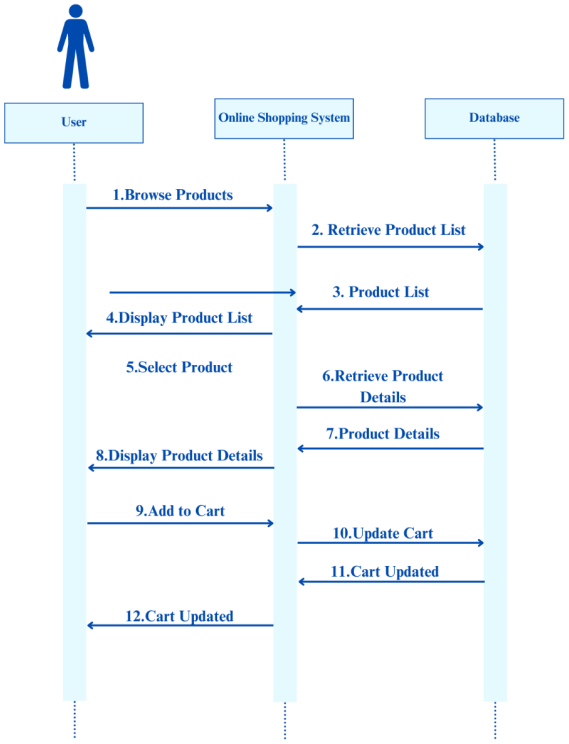


Fig: Sequence Diagram of Online Shopping Diagram

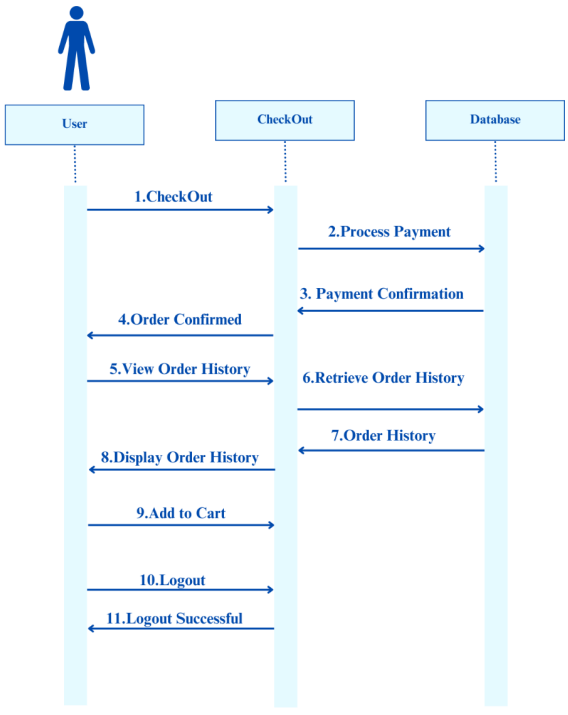


Fig: Sequence Diagram of Payment

State Diagram

It describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of the State diagram is to model the lifetime of an object from creation to termination.

State diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

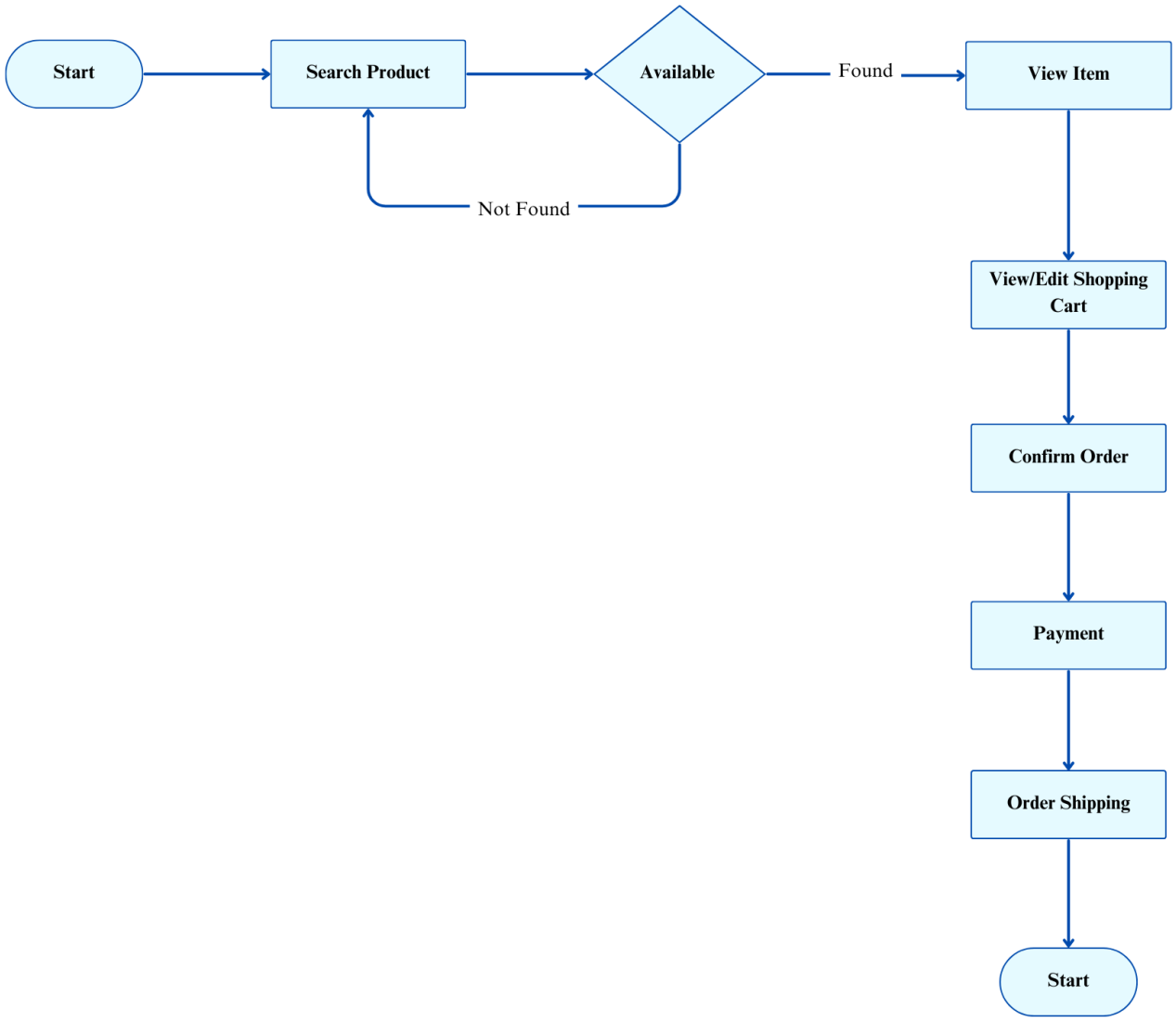


Fig: State Diagram of Online Shopping System

Component Diagram

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system.

The component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

The below Component Diagram shows the component of the Online Shopping System and Components are:-

- Webstore
- Warehouse
- Payment Merchant(Esewa)

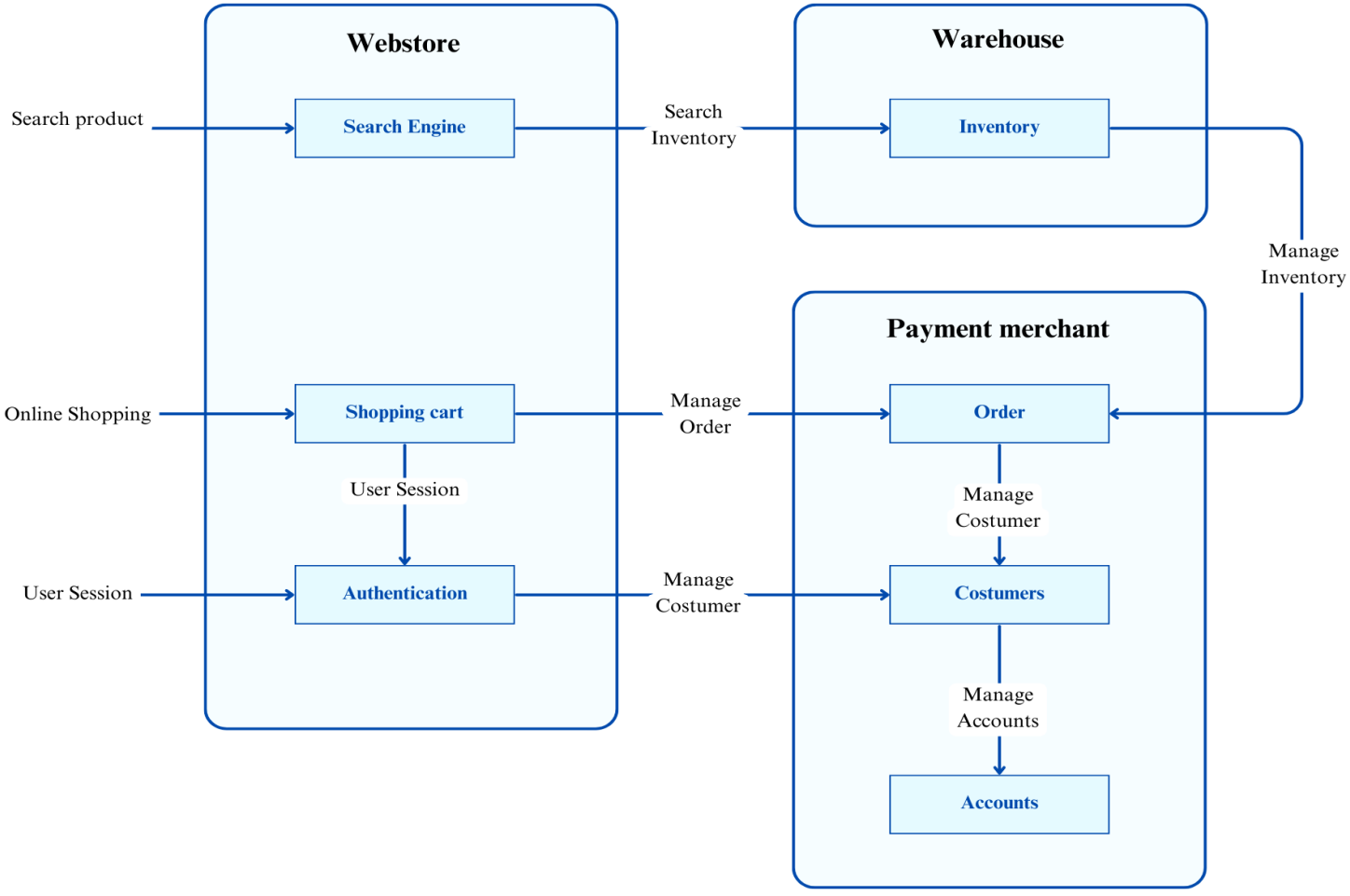


Fig: Component Diagram of Online Shopping System

Implementation

Codes:

Frontend Part : React , tailwind , Bootstrap

About Page:

```
import React from 'react'
import { Footer, Navbar } from "../components";
const AboutPage = () => {
  return ( <><Navbar />
     <div className="container my-3 py-3">
       <h1 className="text-center">About Us</h1>
       <hr /><p className="lead text-center">Our Products are
nice and high quality.</p><h2 className="text-center
py-4">Our Products</h2><div className="row">
         <div className="col-md-3 col-sm-6 mb-3 px-3">
           <div className="card h-100">
<img className="card-img-top img-fluid"
src="https://images.pexels.com/photos/298863/pexels-photo-2
98863.jpeg?auto=compress&cs=tinysrgb&w=600" alt=""
height={160} />
             <div className="card-body">
               <h5 className="card-title text-center">Mens's
Clothing</h5>
             </div>
           </div>
         </div>
         <div className="col-md-3 col-sm-6 mb-3 px-3">
           <div className="card h-100">
             <img className="card-img-top img-fluid"
src="https://images.pexels.com/photos/7679720/pexels-photo-
7679720.jpeg?auto=compress&cs=tinysrgb&w=600" alt=""
height={160} />

             <div className="card-body">
               <h5 className="card-title text-center">Women's
Clothing</h5>
             </div>
           </div>
         </div>
         <div className="col-md-3 col-sm-6 mb-3 px-3">
           <div className="card h-100">
             <img className="card-img-top img-fluid"
src="https://images.pexels.com/photos/1927259/pexels-photo-
1927259.jpeg?auto=compress&cs=tinysrgb&w=600" alt=""
height={160} />
             <div className="card-body">
               <h5 className="card-title
text-center">Jewelery</h5>
             </div>
           </div>
         </div>
         <div className="col-md-3 col-sm-6 mb-3 px-3">
           <div className="card h-100">
             <img className="card-img-top img-fluid"
src="https://images.pexels.com/photos/356056/pexels-photo-3
56056.jpeg?auto=compress&cs=tinysrgb&w=600" alt=""
height={160} />
             <div className="card-body">
               <h5 className="card-title
text-center">Electronics</h5>
             </div>
           </div>
         </div>
       </div>
     </div>
     <Footer />
   </>
  )
```

ProductDetail.jsx

```
import React, { useEffect, useState } from "react";
import Skeleton from "react-loading-skeleton";
import { Link, useParams } from "react-router-dom";
import Marquee from "react-fast-marquee";
import { useDispatch } from "react-redux";
import { addCart } from "../redux/action";

import { Footer, Navbar } from "../components";
const Product = () => {
  const { id } = useParams();
  const [product, setProduct] = useState([]);
  const [similarProducts, setSimilarProducts] = useState([]);
  const [loading, setLoading] = useState(false);
  const [loading2, setLoading2] = useState(false);
  const dispatch = useDispatch();
  const addProduct = (product) => {
    dispatch(addCart(product)); };
  useEffect(() => {
```

```
    const getProduct = async () => {
      setLoading(true);
      setLoading2(true);
      const response = await
fetch(`https://fakestoreapi.com/products/${id}`);
      const data = await response.json();
      setProduct(data);
      setLoading(false);
      const response2 = await fetch(

`https://fakestoreapi.com/products/category/${data.category}`)
; const data2 = await response2.json();
     setSimilarProducts(data2);
      setLoading2(false);};
    getProduct();
  }, [id]);
  const Loading = () => {
    return ( <>
       <div className="container my-5 py-2">
        <div className="row">
         <div className="col-md-6 py-3">
          <Skeleton height={400} width={400} />
         </div>
         <div className="col-md-6 py-5">
          <Skeleton height={30} width={250} />
          <Skeleton height={90} />
          <Skeleton height={40} width={70} />
          <Skeleton height={50} width={110} />
          <Skeleton height={120} />
          <Skeleton height={40} width={110} inline={true} />
          <Skeleton className="mx-3" height={40}
width={110} />
         </div>
        </div> </div></> ); };
  const ShowProduct = () => {
    return ( <>
       <div className="container my-5 py-2">
        <div className="row">
         <div className="col-md-6 col-sm-12 py-3">
          <img
           className="img-fluid"
           src={product.image}
           alt={product.title}
           width="400px"
           height="400px" /> </div>
         <div className="col-md-6 col-md-6 py-5">
          <h4 className="text-uppercase
text-muted">{product.category}</h4>
          <h1 className="display-5">{product.title}</h1>
          <p className="lead">
           {product.rating && product.rating.rate}{" "}
           <i className="fa fa-star"></i>
 </p> <h3 className="display-6
my-4">${product.price}</h3>
<p className="lead">{product.description}</p><button
 className="btn btn-outline-dark" onClick={() =>
addProduct(product)}>Add to Cart</button>
<Link to="/cart" className="btn btn-dark mx-3">
Go to Cart </Link></div>   </div></div></> };
  const Loading2 = () => { return ( <>
       <div className="my-4 py-4">
        <div className="d-flex">
         <div className="mx-4">
          <Skeleton height={400} width={250} /></div>
         <div className="mx-4">
          <Skeleton height={400} width={250} />
         </div>
         <div className="mx-4">
          <Skeleton height={400} width={250} />
         </div>
         <div className="mx-4">
          <Skeleton height={400} width={250} />
         </div> </div></div> </> );}
  const ShowSimilarProduct = () => {
    return (<><div className="py-4 my-4">
        <div className="d-flex">
         {similarProducts.map((item) => {
          return (
           <div key={item.id} className="card mx-4
text-center"><img
            className="card-img-top p-3"
            src={item.image}
            alt="Card"
            height={300}
            width={300} />
           <div className="card-body">
            <h5 className="card-title">
             {item.title.substring(0, 15)}...
            </h5>
           </div>
           {/* <ul className="list-group list-group-flush">
            <li className="list-group-item
lead">${product.price}</li>
           </ul> */}
           <div className="card-body">
            <Link
             to={"/product/" + item.id}
             className="btn btn-dark m-1"
            > Buy Now
            </Link> <button
             className="btn btn-dark m-1"
             onClick={() => addProduct(item)}
            >
Add to Cart
```

```
 </button></div> </div>
);})}</div></div></>);}; return (<>
 <Navbar /><div className="container">
      <div className="row">{loading ? <Loading /> :
<ShowProduct />}</div>
 <div className="row my-5 py-5">
        <div className="d-none d-md-block">
        <h2 className="">You may also Like</h2>
            <Marquee pauseOnHover={true}
              pauseOnClick={true}
              speed={50}>
              {loading2 ? <Loading2 /> : <ShowSimilarProduct />}
            </Marquee> </div> </div> </div><Footer /></> );};

export default Product;
```

Register

```
import React from "react";
import { Footer, Navbar } from "../components";
import { Link } from "react-router-dom";
const Register = () => {
  return (
    <>
      <Navbar />
      <div className="container my-3 py-5">
        <h4 className="text-center">Register</h4>
        <hr />
        <div class="row my-4 h-100">
          <div className="col-md-4 col-lg-4 col-sm-8
mx-auto">
            <form>
              <div class="form my-3">
                <label for="Name">Full Name</label>
                <input
                  type="email"
                  class="form-control"
                  id="Name"
                  placeholder="Enter Your Name"
                />
              </div>
              <div class="form my-3">
                <label for="Email">Email address</label>
                <input
                  type="email"
                  class="form-control"
                  id="Email"
                  placeholder="name@example.com"
                />
              </div>
              <div class="form  my-3">
                <label for="Password">Password</label>
                <input
                  type="password"
                  class="form-control"
                  id="Password"
                  placeholder="Password" />
              </div>
              <div className="my-3">
                <p>
                  Already has an account?{" "}
                  <Link
                    to="/login"
                    className="text-decoration-underline text-info">
                    Login</Link>{" "}</p>
              </div>
              <div className=""><bdisabled
                >
                  Register
                </button>
              </div>
            </form>
          </div>
        </div>
      </div>
      <Footer />
    </>
  );
};

export default Register;
```

Cart.jsx

```
import React from "react";
import { Footer, Navbar } from "../components";
import { useSelector, useDispatch } from "react-redux";
import { addCart, delCart } from "../redux/action";
import { Link } from "react-router-dom";

const Cart = () => {
  const state = useSelector((state) => state.handleCart);
```

```
  const dispatch = useDispatch();

  const EmptyCart = () => {
    return (
      <div className="container">
        <div className="row">
          <div className="col-md-12 py-5 bg-light text-center">
            <h4 className="p-3 display-5">Your Cart is
Empty</h4>
            <Link to="/home" className="btn  btn-outline-dark
mx-4">
              <i className="fa fa-arrow-left"></i> Continue
Shopping
            </Link>
          </div>
        </div>
      </div>
    );
  };

  const addItem = (product) => {
    dispatch(addCart(product));
  };
  const removeItem = (product) => {
    dispatch(delCart(product));
  };

  const ShowCart = () => {
    let subtotal = 0;
    let shipping = 30.0;
    let totalItems = 0;
    state.map((item) => {
      return (subtotal += item.price * item.qty);
    });

    state.map((item) => {
      return (totalItems += item.qty);
    });
    return (
      <>
        <section className="h-100 gradient-custom">
          <div className="container py-5">
            <div className="row d-flex justify-content-center
my-4">
              <div className="col-md-8">
                <div className="card mb-4">
                  <div className="card-header py-3">
                    <h5 className="mb-0">Item List</h5>
                  </div>
                  <div className="card-body">
                    {state.map((item) => {
                      return (
                        <div key={item.id}>
                          <div className="row d-flex
align-items-center">
                            <div className="col-lg-3 col-md-12">
                              <div
                                className="bg-image rounded"
                                data-mdb-ripple-color="light"
                              >
                                <img
                                  src={item.image}
                                  // className="w-100"
                                  alt={item.title}
                                  width={100}
                                  height={75}
                                />
                              </div>
                            </div>

                            <div className="col-lg-5 col-md-6">
                              <p>
                                <strong>{item.title}</strong>
                              </p>
                              {/* <p>Color: blue</p>
                              <p>Size: M</p> */}
                            </div>

                            <div className="col-lg-4 col-md-6">
                              <div
                                className="d-flex mb-4"
                                style={{ maxWidth: "300px" }}
                              >
                                <button
                                  className="btn px-3"
                                  onClick={() => {
                                    removeItem(item);
                                  }}
                                >
                                  <i className="fas fa-minus"></i>
                                </button>

                                <p className="mx-5">{item.qty}</p>

                                <button
                                  className="btn px-3"
                                  onClick={() => {
                                    addItem(item);
    }} ><i className="fas fa-plus"></i>
                                </button> </div><p className="text-start
text-md-center"><strong><span
className="text-muted">{item.qty}</span>{" "}
                                  x ${item.price}
                                </strong></p></div>
```

```
                  </div> <hr className="my-4" />
                 </div>
               ); })} </div> </div> </div>
          <div className="col-md-4">
            <div className="card mb-4">
              <div className="card-header py-3 bg-light">
                <h5 className="mb-0">Order Summary</h5>
              </div>
              <div className="card-body">
                <ul className="list-group list-group-flush">
                  <li className="list-group-item d-flex
justify-content-between align-items-center border-0 px-0
pb-0">
                    Products
({totalItems})<span>${Math.round(subtotal)}</span>
                  </li>
                  <li className="list-group-item d-flex
justify-content-between align-items-center px-0">
                    Shipping
                    <span>${shipping}</span>
                  </li>
                  <li className="list-group-item d-flex
justify-content-between align-items-center border-0 px-0
mb-3">
                    <div>
                      <strong>Total amount</strong>
                    </div>
                    <span>
                      <strong>${Math.round(subtotal +
shipping)}</strong>
                    </span>
                  </li>
                </ul>

                <Link
                  to="/checkout"
                  className="btn btn-dark btn-lg btn-block”  >
                  Go to checkout
  return (<Navbar />
    <div className="container my-3 py-3">
      <h4 className="text-center">Cart</h4>
      <hr />
      {state.length > 0 ? <ShowCart /> : <EmptyCart />}
    </div>
    <Footer />
  </>  );};
```

Backend part : Nodejs, Expressjs, Prisma, Postgres

Signup/Login:

```
import * as Boom from "@hapi/boom"
import { PrismaClient } from "@prisma/client"
import bcrypt from "bcryptjs"
import {
  createAccessToken,
  createRefreshToken,
  verifyRefreshToken,
} from "../utils/token.utils"
const prisma = new PrismaClient()


//Login Part
export const login = async (email: string, password: string) =>
{
  let user: any
  try {
    user = await prisma.user.findFirstOrThrow({
      where: {
        email,
      },
    })
  } catch (err: any) {
    if (err.code === "P2025") {
      throw Boom.notFound("Record does not exist")
    }
  }
  const passwordMatch = await bcrypt.compare(password,
user.password)

  if (!passwordMatch) {
    throw Boom.unauthorized("Password Does not match")
  }
  const accessToken = createAccessToken(user.id,
user.isAdmin)

  const refreshToken = createRefreshToken(user.id,
user.isAdmin)
  return { accessToken, refreshToken }}

//Signup Part
export const register = async (user: any) => {
  try {
    const {name, address,phoneNumber, email, password } =
user
```

```
    const users = await prisma.user.create({
      data: {
        name,
          address,
          phoneNumber,
        email,
        password: await bcrypt.hash(password as string, 10),
      },
      select: {
        email: true,
        addresses: true,
        phone_number: true,
        id: true,
      },
    })
    return users
  } catch (err: any) {
    if (err.code === "P2025") {
      throw Boom.badRequest("Email already exist")
    }
    if (err.code === "P2002") {
      throw Boom.badRequest("Email already exist")
    } else {
      throw err
    }
  }
}

//Refresh Token
export const refreshToken = async (refreshToken: any) => {
  try {
    const decodedToken = verifyRefreshToken(refreshToken)
    console.log(decodedToken)
  } catch (err) {
    throw Boom.unauthorized("User is not logged in")
  }
}
```

##Category:

```
import * as Boom from "@hapi/boom"
import { PrismaClient } from "@prisma/client"
const prisma = new PrismaClient()
//get one product
export const getProduct = async (id: number) => {
  try {
    return await prisma.productCategory.findFirstOrThrow({
      where: {
        id,
      },
      select: {
        id: true,
        category_name: false,
      },
    })
  } catch (err: any) {
    if (err.code === "P2025") {
      throw Boom.notFound("Record not found")
    } else {
      throw err
    }
  }
}
//get all products
export const getAll = async () => {
  try {
    console.log("This is get all")
    const product = await
prisma.productCategory.findMany({
      select: {
        id: true,
        category_name: true,
      },
    })
    return product
  } catch (err: any) {
    if (err.code === "P2025") {
      throw Boom.notFound("No records found")
    }
  }
}
//post products
export const setProduct = async (product: any) => {
  // eslint-disable-next-line no-useless-catch
  try {
    const { category_name } = product
    const products = await prisma.productCategory.create({
      data: {
        id: Math.ceil(Math.random() * 1000),
        category_name,
      },
      select: {
        id: true,
        category_name: true,
      },
    })
    return products
  } catch (err) {
    throw err
  }
}

//updatéproducts
export const updateProduct = async (id: number, product: any)
=> {
```

```
    try {
        const { category_name } = product
        const products = await prisma.productCategory.update({
            where: {
                id,
            },
            data: {
                category_name,
            },
        })
        return products
    } catch (err: any) {
        if (err.code === "P2025") {
            throw Boom.notFound("Record doesnot match")
        } else {
            throw err
        }
    }
}
//delete products
export const deleteProduct = async (id: number) => {
    return await prisma.product.delete({ where: { id } })
}
```

Database Model (Using Prism):

```
generator client {
  provider      = "prisma-client-js"
  binaryTargets = ["native", "windows","linux-musl"]
}

datasource db {
  provider = "postgresql"
  url = env("DATABASE_URL")}
model Product {
  id            Int            @id @default(autoincrement())
  category      ProductCategory @relation(fields: [category_id],
references: [id])
  category_id   Int
  name          String
  description   String?
  product_image String
  product_items ProductItem[]
}
model ProductItem {
  id                 Int                 @id @default(autoincrement())
  product            Product             @relation(fields:
[product_id], references: [id])
  product_id         Int
  SKU                String
  quantity_in_stock  Int
  product_image      String
  price              Float
  configurations     ProductConfiguration[]
  shopping_cart_items ShoppingCartItem[]
  UserReview         UserReview[]
}
model ProductCategory {
  id                 Int             @id @default(autoincrement())
references: [id])
  parent_category_id Int?
  category_name      String
}

model OrderStatus {
  id        Int        @id @default(autoincrement())
  status    String
}
model ShoppingCart {
  id                 Int             @id @default(autoincrement())
  user               User            @relation(fields: [user_id],
references: [id])
  user_id            Int
}
model User {
  id          Int          @id @default(autoincrement())
  email       String       @unique
  phone_number String?
  password    String
  isAdmin     Boolean      @default(false)
  orders      ShopOrder[]
}
```

Admin CRUD

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import { PrismaClient } from "@prisma/client"
import * as Boom from "@hapi/boom"
import bcrypt from "bcryptjs"

const prisma = new PrismaClient()
//display one user
export const displayOne = async (id: number) => {
    try {
        const users = await prisma.user.findFirstOrThrow({
            select: {
                id: true,
                email: true,
                phone_number: true,
                addresses: true,
            },
```

```
            where: { id: Number(id) },
        })
        return users
    } catch (err: any) {
        if (err.code === "P2025") {
            throw Boom.notFound("Record not found")
        } else {
            throw err
        }
    }
}
//display all user
export const display = async () => {
    try {
        const users = await prisma.user.findMany({
            select: {
                id: true,
                email: true,
                phone_number: true,
                password: false,
            },
        })
        return users
    } catch (err: any) {
        if (err.code === "P2025") {
            throw Boom.notFound("Post not found")
        } else {
            throw err
        }
    }
}
//create user
// eslint-disable-next-line @typescript-eslint/no-explicit-any
export const createUser = async (
    email: string,
    password: string,
    isAdmin: boolean,
    phone_number: string
) => {
    try {
        const users = await prisma.user.create({
            data: {
                email,
                isAdmin,
                id: Math.ceil(Math.random() * 100),
                phone_number,
                password: await bcrypt.hash(password as string, 10),
            },
        })

        return users
    } catch (err: any) {
        if (err.code === "P2002") {
            throw Boom.badRequest("Email already exist")
        } else {
            throw err
        }
    }
}
//delete user
export const deleteUser = async (id: number) => {
    try {
        const user = await prisma.user.delete({
            where: {
                id,
            },
        })
        return user
    } catch (error: any) {
        if (error.code === "P2025") {
            throw Boom.notAcceptable("Record not found")
        } else {
            throw error
        }
    }
}
//update user
export const updateUser = async (id: string, user: any) => {
    try {
        const { email, password, phone_number } = user
        const users = await prisma.user.update({
            where: {
                id: Number(id),
            },
            data: {
                email,
                id: Math.ceil(Math.random() * 100),
                phone_number,
                password: await bcrypt.hash(password as string, 10),
            },
        })
        return users
    } catch (err: any) {
        if (err.code == "P2025") {
            throw Boom.notFound("Record not found")
        }
        if (err.code === "P2002") {
            throw Boom.notAcceptable("Email already is in use")
        } else {
            throw Error
        }
    }
}
```

```
//insert user
export const postUserss = async (user: any) => {
   try {
      const { email, password, phone_number } = user
      const users = await prisma.user.create({
         data: {
            email,
            id: Math.ceil(Math.random() * 100),
            phone_number,
            password: await bcrypt.hash(password as string, 10),
         },
      }
      return users
   } catch (err: any) {
      if (err.code === "P2002") {
         throw Boom.badRequest("Email already exist")
      } else {
         throw err
      }
   }
}
```

Esewa Integration Code:

```
const esewa = new Esewa({
  key: 'YOUR_ESEWA_MERCHANT_KEY',
  secret: 'YOUR_ESEWA_MERCHANT_SECRET',
  environment: 'test', // 'test' for sandbox environment,
'production' for live environment
});

// Route for initiating the payment
app.get('/payment', async (req, res) => {
  try {
    // Generate a unique transaction ID and amount
    const txId = Date.now().toString();
    const amount = 100; // Replace with the actual amount to be
charged

    // Create the payment URL
    const paymentUrl = await esewa.initiatePayment(txId,
amount, 'Payment for Order XYZ');

    // Redirect the user to the eSewa payment page
    res.redirect(paymentUrl);
  } catch (error) {
    console.error('Error initiating payment:', error);
    res.status(500).send('Error initiating payment');
  }
});

// Route for handling the callback from eSewa after payment
app.get('/payment/callback', async (req, res) => {
  try {
    // Verify the payment response
    const verified = await esewa.verifyPayment(req.query);

    if (verified) {
      // Payment is successful
      res.send('Payment successful');
    } else {
      // Payment verification failed
      res.send('Payment verification failed');
    }
  } catch (error) {
    console.error('Error verifying payment:', error);
    res.status(500).send('Error verifying payment');
  }
});
```

Output

```
SELECT * FROM "Product" LIMIT 100
```

Search results | Free | Cost: 6ms | 1 2 3 4 5 | Total 402

	* id integer	* category_id integer	* name text	description text	* product_image text
1	1	1	Product 1	Product 1 description	product1.jpg
2	2	2	Product 2	Product 2 description	product2.jpg
3	3	1	Product 1-1	Product 1-1 description	product1_1.jpg
4	4	1	Product 1-2	Product 1-2 description	product1_2.jpg
5	5	1	Product 1-3	Product 1-3 description	product1_3.jpg
6	6	1	Product 1-4	Product 1-4 description	product1_4.jpg
7	7	1	Product 1-5	Product 1-5 description	product1_5.jpg
8	8	1	Product 1-6	Product 1-6 description	product1_6.jpg
9	9	1	Product 1-7	Product 1-7 description	product1_7.jpg
10	10	1	Product 1-8	Product 1-8 description	product1_8.jpg
11	11	1	Product 1-9	Product 1-9 description	product1_9.jpg
12	12	1	Product 1-10	Product 1-10 description	product1_10.jpg
13	13	1	Product 1-11	Product 1-11 description	product1_11.jpg
14	14	1	Product 1-12	Product 1-12 description	product1_12.jpg
15	15	1	Product 1-13	Product 1-13 description	product1_13.jpg

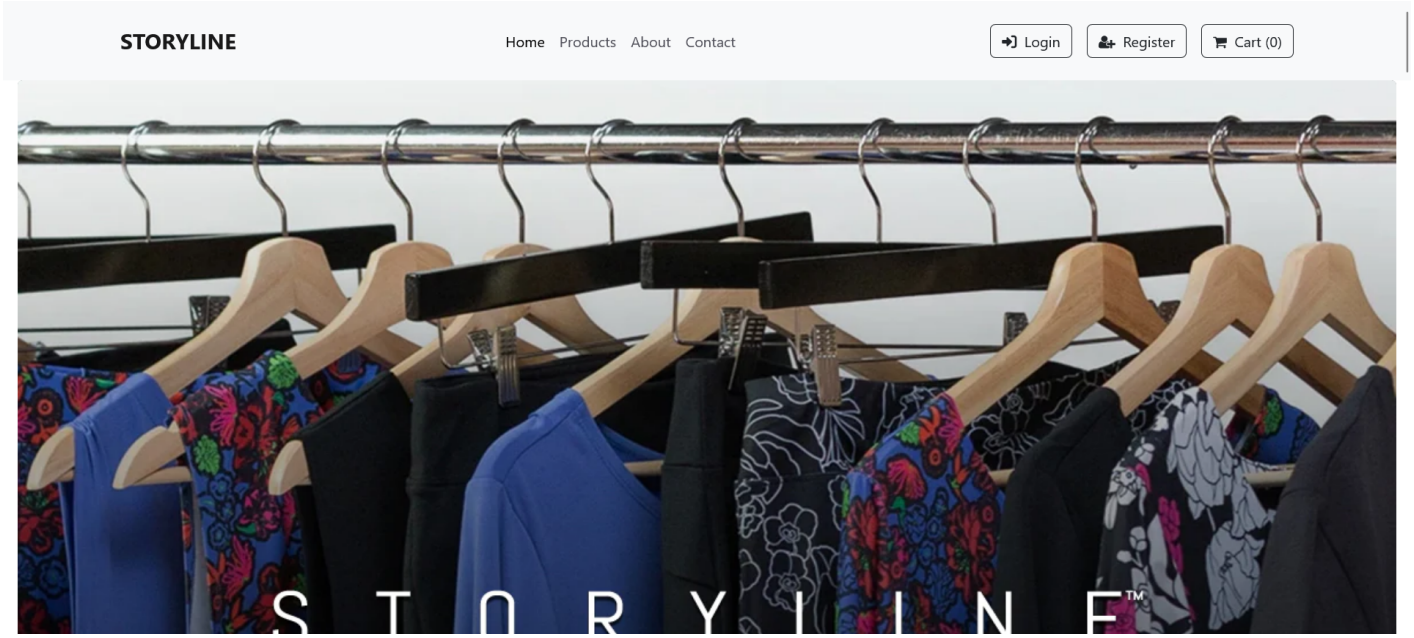
Product database

```
SELECT * FROM "User" LIMIT 100
```

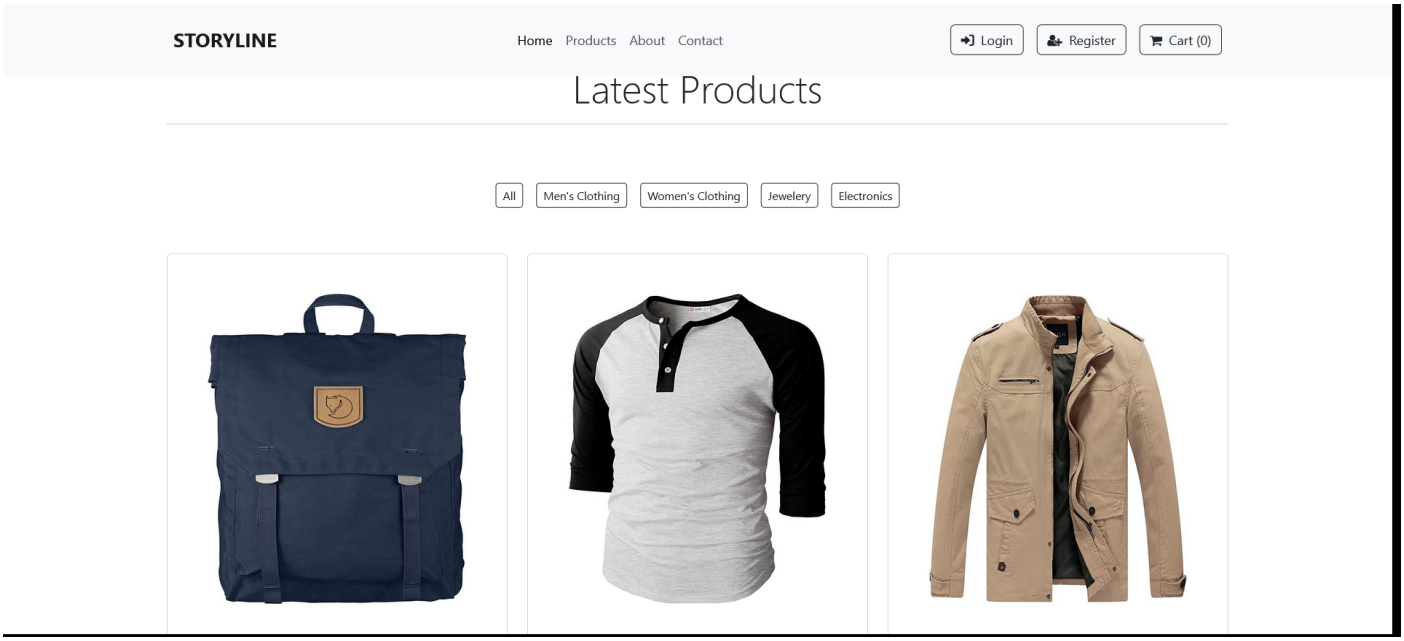
Search results | Free | Cost: 4ms | 1 | Total 20

	* id integer	* email text	phone_number text	* password text	* isAdmin boolean
1	1	user1@example.com	1234567891	password1	true
2	2	user2@example.com	1234567892	password2	true
3	3	user3@example.com	1234567893	password3	true
4	4	user4@example.com	1234567894	password4	true
5	5	user5@example.com	1234567895	password5	true
6	6	user6@example.com	1234567896	password6	false
7	7	user7@example.com	1234567897	password7	false
8	8	user8@example.com	1234567898	password8	false
9	9	user9@example.com	1234567899	password9	false
10	10	user10@example.com	12345678910	password10	false
11	11	user11@example.com	12345678911	password11	false
12	12	user12@example.com	12345678912	password12	false
13	13	user13@example.com	12345678913	password13	false
14	14	user14@example.com	12345678914	password14	false
15	15	user15@example.com	12345678915	password15	false

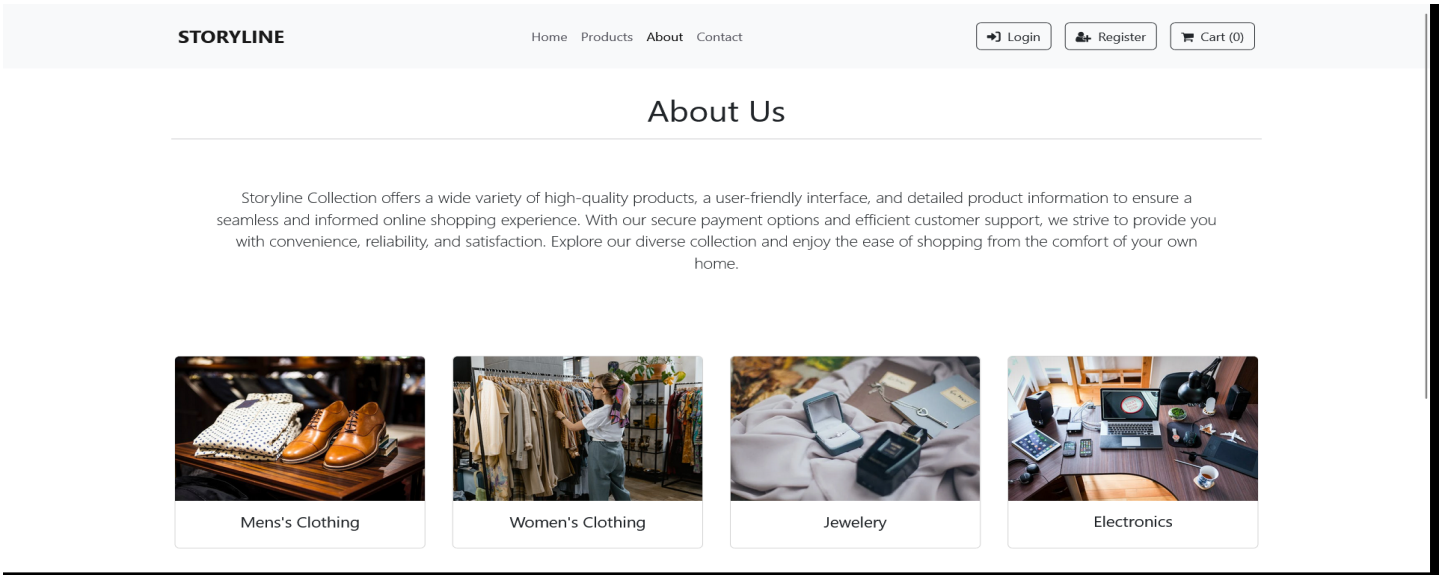
User Database



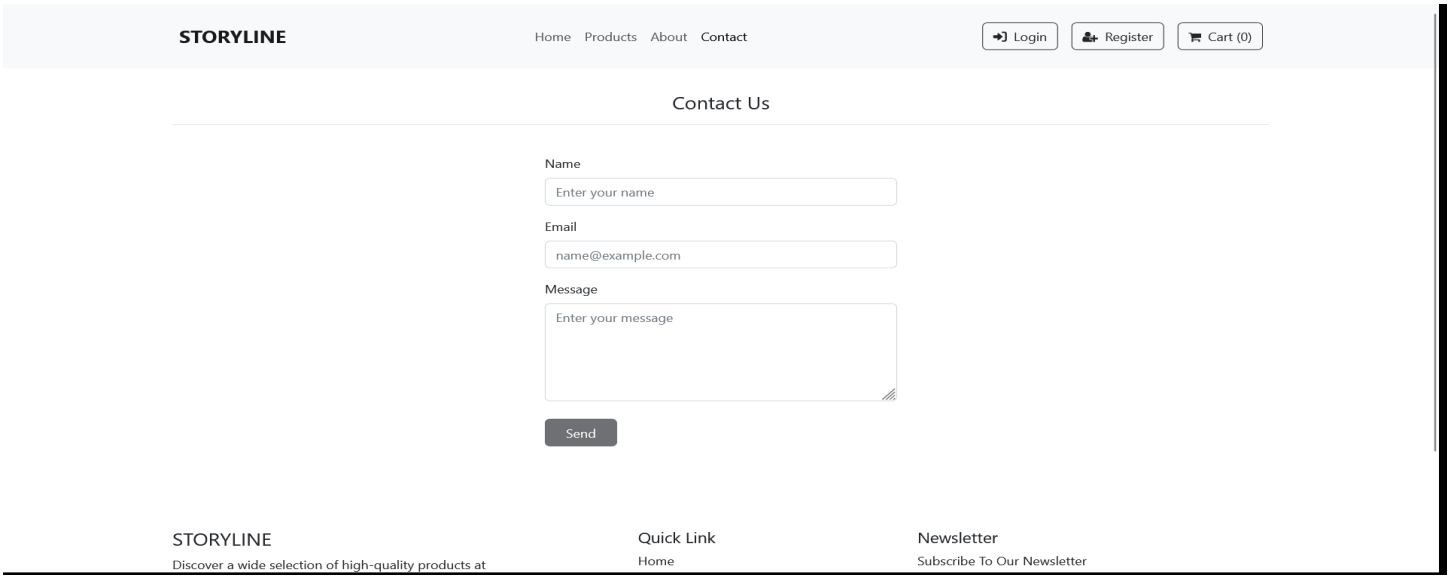
HomePage



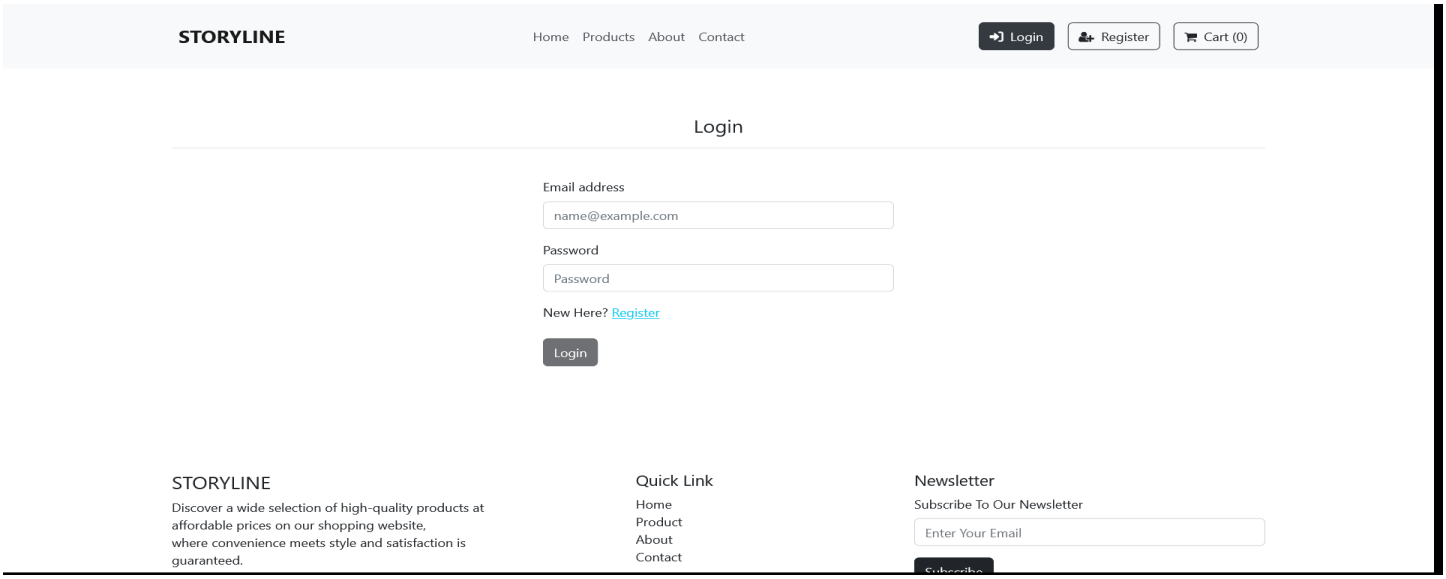
Latest Products



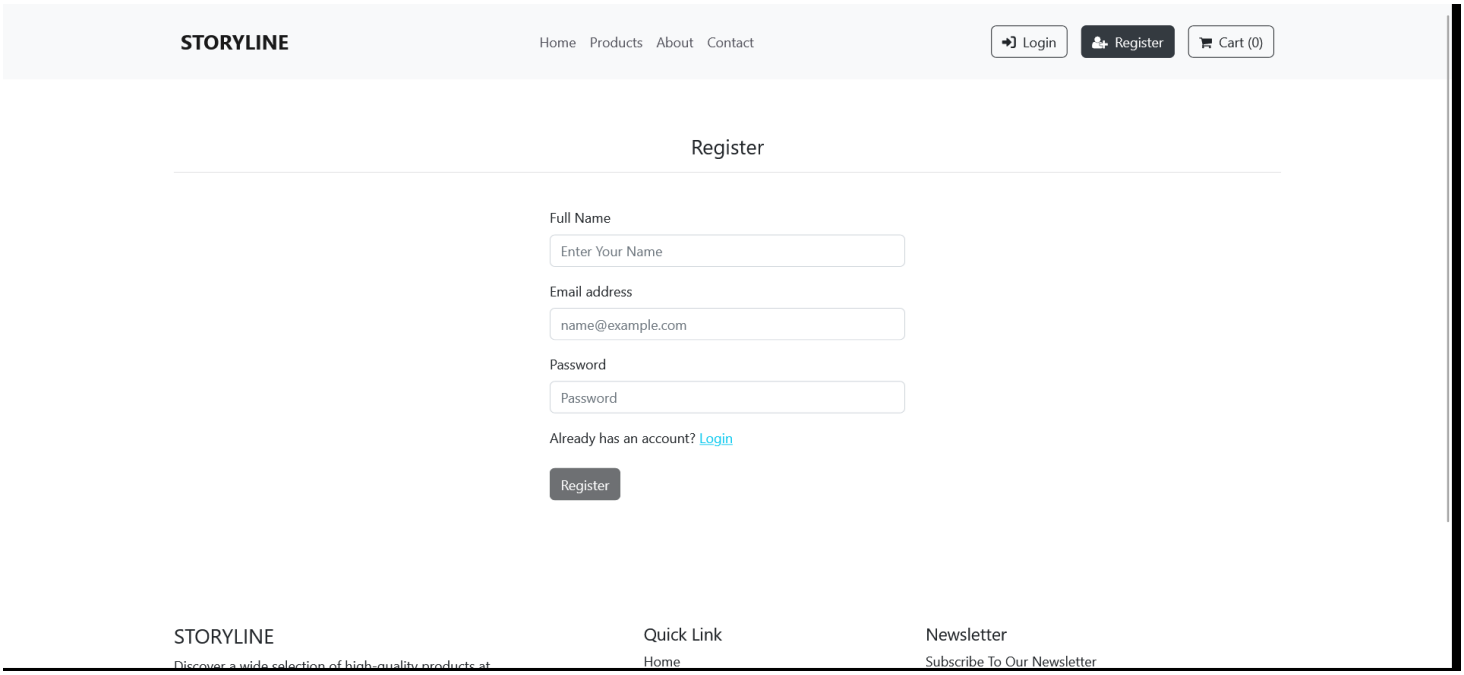
About Us Page



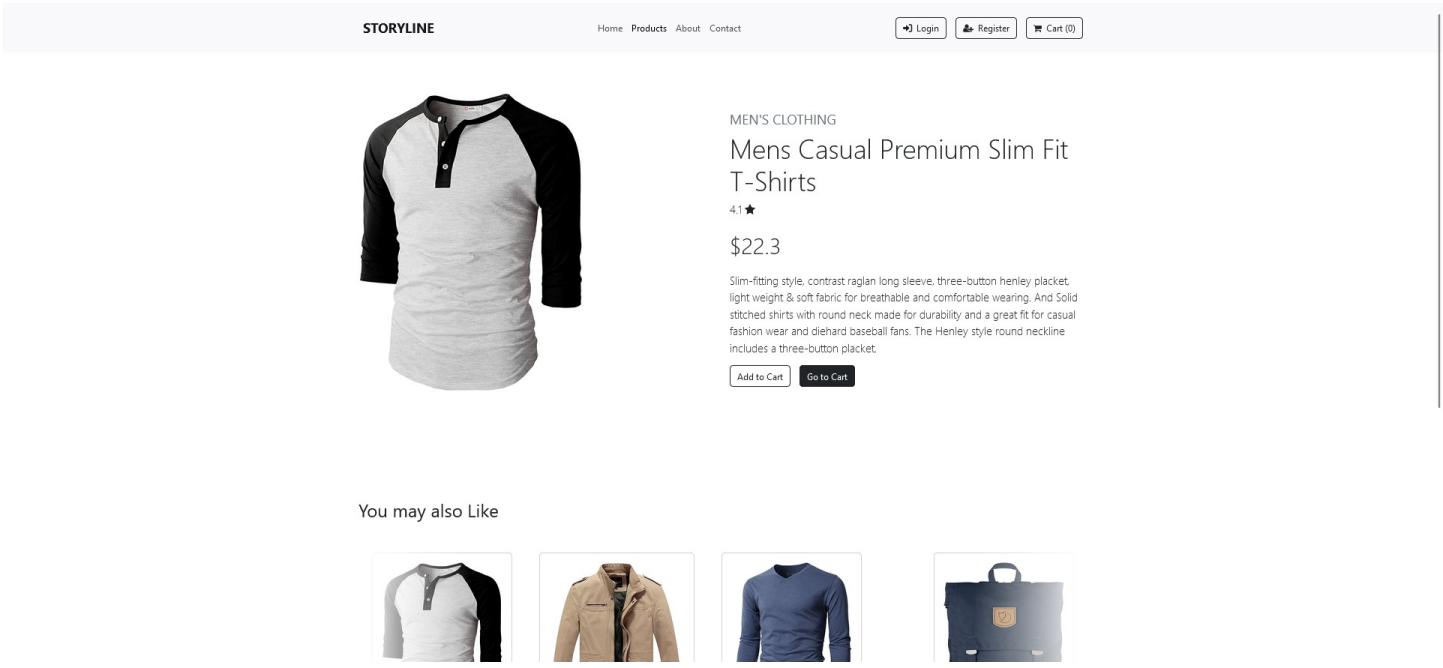
Contact Us Page



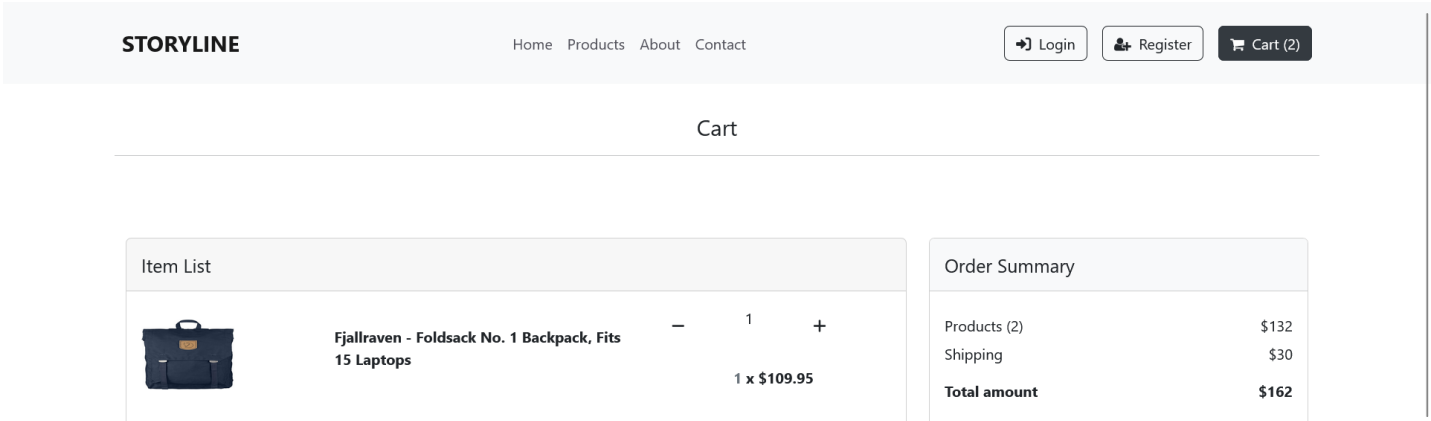
Login Page



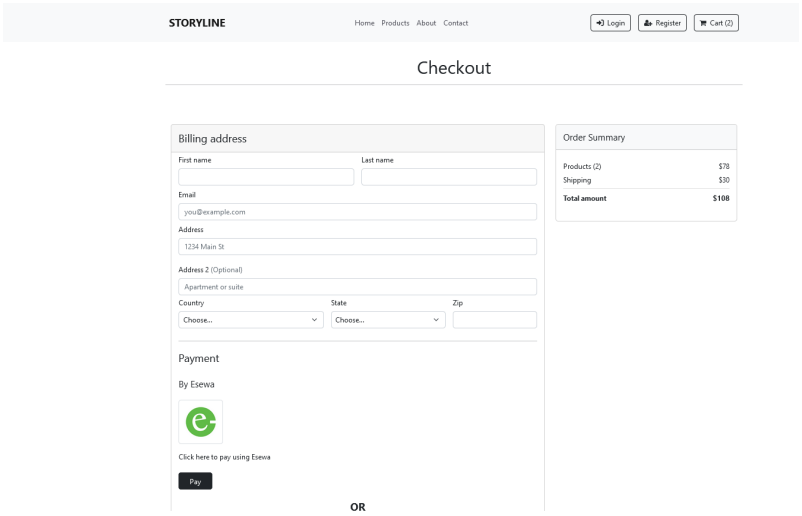
Registration page

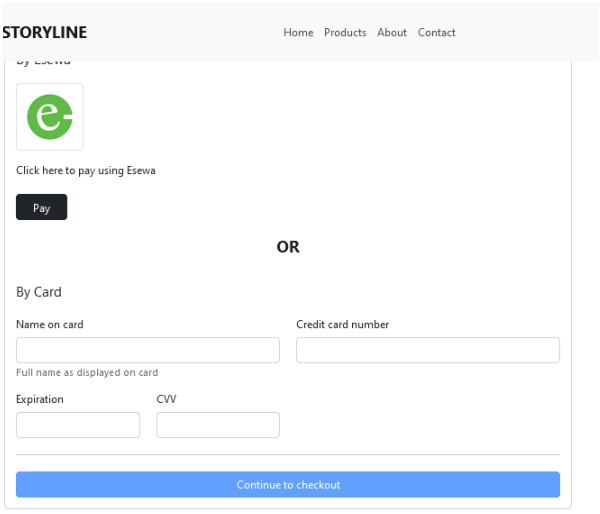


Detail page



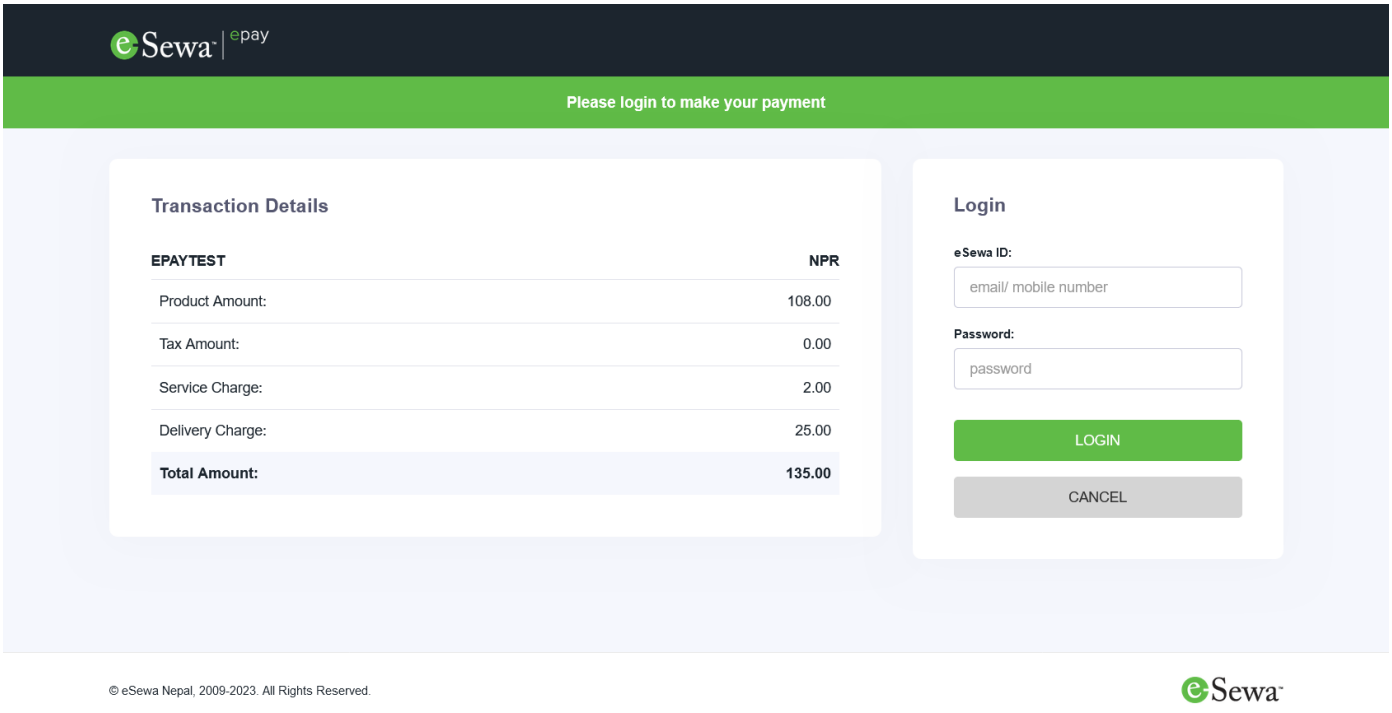
Cart page





Checkout page

Payment page



Payment login page

Future Recommendations

In the future, there are several recommendations for improving our online shopping system:

Enhanced Personalization: Develop advanced algorithms and machine learning techniques to provide highly personalized recommendations to customers based on their browsing history, purchase behavior, and preferences. This can improve the overall shopping experience and increase customer satisfaction.

Improved Product Visualization: Invest in high-quality product imagery and videos, allowing customers to see products from different angles and zoom in for finer details. Additionally, 360-degree product views or virtual tours can provide a more immersive experience.

Faster and More Flexible Delivery Options: Enhance the logistics and delivery system to provide faster shipping options, including same-day or next-day delivery. Additionally, offer flexible delivery options such as time slots or alternative pickup locations to accommodate customers' preferences and schedules.

Simplified Checkout Process: Streamline the checkout process by minimizing the number of steps required and offering guest checkout options. Implementing one-click payment methods or digital wallets can further simplify and expedite the payment process.

Continuous Feedback and Improvement: Collect customer feedback and reviews to identify areas for improvement and enhance the overall online shopping experience. Regularly analyze data and metrics to identify pain points and implement iterative updates and optimizations.

References

Shopping System ER Diagram | FreeProjectz. (2017, July 17). https://www.freeprojectz.com/entity-relationship/shopping-system-er-diagram

UML - Component Diagrams. (n.d.). https://www.tutorialspoint.com/uml/uml_component_diagram.htm

UML Collaboration Diagram (UML2.0) | Diagramming Software for Design UML Collaboration Diagrams | UML Collaboration Diagram. Design Elements | Collaboration Diagram For Online Shopping Project. (n.d.). https://www.conceptdraw.com. https://www.conceptdraw.com/examples/collaboration-diagram-for-online-shopping-project

eSewa Document. (n.d.). https://developer.esewa.com.np/#/