

Course: Web Application Development

Course ID: IT093IU



**INTERNATIONAL UNIVERSITY -
VIETNAM NATIONAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE AND
ENGINEERING**

Clothing E-Commerce Website Project

Group member:

NAME	STUDENT ID
Vũ Nhật Duy	ITITIU17047
Hoàng Minh	ITITIU19029
Nguyễn Gia Phúc	ITITIU19041

Table of Contents

Clothing E-Commerce Website Project	0
Table of Contents	1
ABSTRACT	3
Chapter 1: Introduction to the Project	4
1.1. Overview of the Project	4
1.2. Purpose and Goals	4
1.2.1. Main Goals of the Project	4
1.2.2. Target Audience	4
Chapter 2: Details and Technologies	5
2.1. Details	5
2.1.1. Stakeholders	5
2.2. Technologies	5
2.2.1. Front-end Technologies	5
2.2.2. Back-end Technologies	6
Chapter 3: Project Planning	7
3.1. Starting the Project (10/4-17/4)	8
3.2. Analyzing the Project (17/4-24/4)	8
3.3. Implementation (24/4-10/5)	8
3.4. Testing (10/5-End)	8
3.5. Other tasks (10/5-End)	8
Chapter 4: Analysis	9
4.1. Functional and Non-functional requirements	9
4.2. Non-functional requirements	9
Chapter 5: System Design	10
5.1. Use Case Diagram	10
5.1.1. Use Case: Customers	10
5.1.2. Use Case: Administrators	11
5.1.3. Use Case: Payment Gateway (e.g., Stripe)	11
5.2. Sequence Diagram	13
Reference	13
5.3: Class Diagram	14
Reference	14
5.4. Database	14
5.4.1. Tables	14
5.4.2. Relationships	15
Chapter 6: Implementation	16
6.1. Setting up Workspace	16
6.2. Setting up Database	17
6.2.1. Set up Database Configuration	17
6.2.2. Install MySQL Driver	18
6.2.3. Connect to MySQL from Node.js	18

6.3. Coding Frontend	19
6.3.1. Create React Components	19
6.3.2. Use React Router for Routing	20
6.3.3. Use Axios for API Calls	21
6.3.4. Style your Components	22
6.4. Coding Backend	22
6.4.1. Create Express Routes	22
6.4.2. Implement Controllers and Business Logic	23
6.4.3. Connect to the Database	24
6.4.4. Handle Authentication and Authorization	25
6.4.5. Error Handling and Middleware	26
6.5. Stripe	26
6.5.1. Sign up for a Stripe Account	26
6.5.2. Install the Stripe Node.js Library	26
6.5.3. Set up Stripe API Keys	26
6.5.4. Implement Stripe Integration	26
6.6. Cloudinary	27
6.6.1. Sign up for a Cloudinary Account	27
6.6.2. Install the Cloudinary Node.js SDK	28
6.6.3. Set up Cloudinary API Credentials	28
6.6.4. Implement Cloudinary Integration	28
6.7. Admin Dashboards	29
6.7.1. Design the Admin Dashboard	29
6.7.2. Implement Admin Routes	29
6.7.3. Build Admin Functionality	29
6.7.4. Apply Security Measures	29
6.8. Project Github	29
Chapter 7: Deployment	30
7.1. Prepare Your Project	30
7.2. Choose a Hosting Provider	30
7.3. Set Up Hosting Environment	30
7.4. Build and Bundle React App	30
7.5. Set Up Deployment Configuration	30
7.6. Deploy Your Backend (Node.js/Express)	31
7.7. Set Up Database	31
7.8. Deploy Your Frontend (React)	31
Chapter 8: Conclusion	31
8.1. Summary of the project	31
8.2. Challenges faced and lessons learned	32
8.3. Possible future improvements or extensions	32
References	33

ABSTRACT

An e-commerce website is a reliable tool that businesses create to provide products and services to customers over the Internet. This type of website typically comes with a user-friendly interface with which users can interact, browse products and services inventory, add items to shopping carts and securely complete transactions.

There are multiple features incorporated into the website, such as search functionality, product categories, customer reviews, ratings, and site recommendations. Many sites also offer personalized experiences through the use of data-driven insights and analytics. In order to determine the success of an e-commerce website, user experience is the primary factor, along with other factors such as pricings, range of products, sales and promotions.

Therefore, this Web Application Development project's goal is to design a web application that can operate as an e-commerce website with all of the aforementioned features, together with functional user authentications and databases.

Chapter 1: Introduction to the Project

1.1. Overview of the Project

In today's digital era, e-commerce has revolutionized the way businesses operate and consumers shop. The online marketplace provides immense opportunities for businesses to reach a global customer base, expand their market presence, and increase sales. This project aims to develop an e-commerce website that caters to the growing demands of the digital marketplace.

1.2. Purpose and Goals

The purpose of this project is to create a comprehensive and user-friendly e-commerce website that allows customers to browse and purchase products with ease. The primary goal is to develop a robust platform that seamlessly integrates essential features such as product catalog, shopping cart, secure payment options, order management, and customer support.

1.2.1. Main Goals of the Project

1. Design and develop an attractive and intuitive user interface that enhances the shopping experience for customers.
2. Implement a reliable and secure payment gateway to facilitate safe transactions.
3. Develop an efficient inventory management system to track product availability and update stock levels in real-time.
4. Incorporate personalized recommendations and search functionalities to assist customers in finding relevant products.

1.2.2. Target Audience

The target audience for this e-commerce website includes both individual consumers and businesses seeking to purchase products online. The website will cater to a diverse range of customers, including fashion enthusiasts, tech-savvy individuals, and those looking for convenient and reliable online shopping experiences.

By successfully achieving these goals, the developed e-commerce website will provide a platform that facilitates seamless transactions, enhances customer satisfaction, and drives business growth in the highly competitive online marketplace.

In the following chapters of this report, we will delve into the detailed process of developing the e-commerce website, including the technologies, methodologies, and challenges encountered throughout the project. We will also provide insights into the final product, highlighting its features and functionalities that make it a compelling and successful e-commerce platform.

Chapter 2: Details and Technologies

2.1. Details

2.1.1. Stakeholders

The development of an e-commerce website involves various stakeholders who play crucial roles in its success. It is important to identify and understand the needs and expectations of these stakeholders to ensure the website meets their requirements. The key stakeholders involved in an e-commerce website project typically include:

1. **Customers:** The end-users of the website who browse products, make purchases, and provide feedback on their shopping experience.
2. **Business Owners:** The individuals or organizations who own and operate the e-commerce business, responsible for defining the business goals, marketing strategies, and overall website direction.
3. **Employees:** The internal team members who manage and maintain the website, including administrators, customer support personnel, and marketing professionals.
4. **Suppliers:** The vendors and suppliers who provide the products or services featured on the website.
5. **Partners:** Collaborative partners, such as payment gateway providers, shipping and logistics companies, and marketing affiliates, who contribute to the smooth operation of the website.
6. **Regulators:** Regulatory authorities and governing bodies responsible for ensuring compliance with legal and industry standards, such as data protection regulations, consumer rights, and taxation requirements.

Understanding the expectations and requirements of these stakeholders is crucial for delivering an e-commerce website that effectively serves their needs and achieves the desired outcomes.

2.2. Technologies

2.2.1. Front-end Technologies

The front-end of an e-commerce website focuses on creating an engaging and user-friendly interface that enhances the customer's shopping experience. The following technologies are commonly used in front-end development:

1. **HTML (Hypertext Markup Language):** Used to structure the content and layout of web pages.
2. **CSS (Cascading Style Sheets):** Used to define the visual appearance, layout, and design of web pages.

3. **JavaScript:** A programming language that enables interactive and dynamic elements on web pages.
4. **ReactJS:** A popular JavaScript library for building responsive and interactive user interfaces.

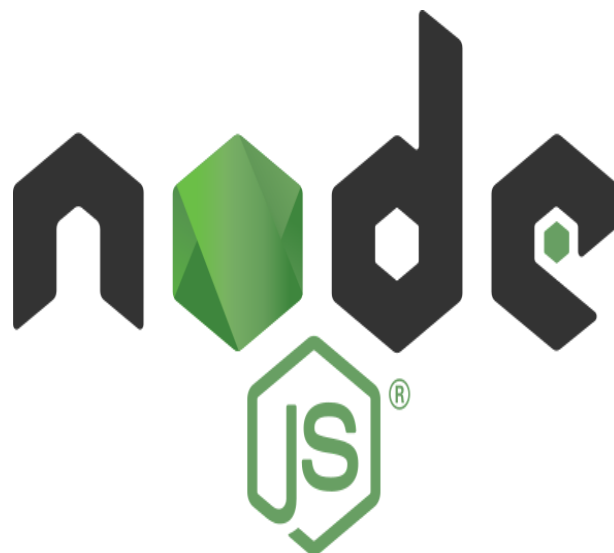


These front-end technologies allow developers to create visually appealing and intuitive interfaces that facilitate seamless navigation and product discovery for customers.

2.2.2. Back-end Technologies

The back-end of an e-commerce website involves managing the business logic, data processing, and integration with various systems. The following technologies are commonly used in back-end development:

1. **Node.js:** A JavaScript runtime environment that allows server-side scripting and event-driven programming, facilitating the development of scalable and high-performance web applications.



2. **MySQL:** A popular relational database management system used for storing and managing product data, customer information, and order details.



3. **Express.js:** A flexible and lightweight web application framework for Node.js, enabling the development of robust APIs and handling server-side functionalities.

#Stripe

These back-end technologies provide the necessary infrastructure to handle tasks such as order processing, inventory management, payment processing, and integration with third-party services.

By leveraging the right combination of front-end and back-end technologies, an e-commerce website can deliver a seamless and efficient shopping experience while ensuring the secure management of data and transactions.

Chapter 3: Project Planning

In this chapter, we will outline the project plan for our upcoming endeavor. A well-structured plan is essential for the successful execution of any project. The following table provides an overview of the tasks, sub-tasks, assignees, deadlines, and reviewers involved. Please note that this table is subject to revisions and updates as the project progresses.

Period	Task	Sub-task	Assignee	Deadline	Reviewer
10/4-17/4	Starting the Project	Setting up workspace	All members	17/4	Duy
17/4-24/4	Analyzing the Project	Requirements	Duy	24/4	All members
		Diagrams	Duy, Phuc		
		Presentation Slides	Duy	14/5	
24/4-10/5	Implementation	MySQL Database	Duy	10/5	
		Application	Duy	10/5	
10/5-End	Testing	Testing	All members	26/5	
		Other tasks	All members	26/5	

3.1. Starting the Project (10/4-17/4)

During this phase, the team will focus on setting up the workspace required for the project. This task will be collectively handled by all team members to ensure a smooth start. The deadline for completing this task is set for 17/4, and Duy will review the progress.

3.2. Analyzing the Project (17/4-24/4)

In this period, the team will analyze the project in detail. Duy will take the lead in identifying and documenting the project's requirements. The deadline for completing the requirements analysis is set for 24/4, and all team members will review and provide their inputs.

Duy, along with Phuc, will work on creating diagrams to visualize the project's structure, workflows, or any other relevant aspects. Additionally, Duy will be responsible for preparing presentation slides to communicate the analyzed project details effectively. The deadline for the diagrams and presentation slides is set for 24/4, with no specific reviewer assigned.

3.3. Implementation (24/4-10/5)

During this phase, the team will focus on implementing the project. Duy will handle the implementation of the MySQL database, ensuring its proper setup and functionality. The deadline for completing the database implementation is set for 10/5, with no specific reviewer assigned.

Furthermore, Duy will also be responsible for developing the application itself. He will work on coding the necessary functionalities and ensuring a seamless user experience. The deadline for completing the application development is set for 10/5, with no specific reviewer assigned.

3.4. Testing (10/5-End)

Once the implementation phase is complete, the team will shift their focus to testing the project. All team members will actively participate in testing the application, ensuring that it meets the specified requirements and functions as intended. The deadline for completing the testing phase is set for 26/5, with no specific reviewer assigned.

3.5. Other tasks (10/5-End)

Throughout the project, there might be additional tasks that arise, which are not explicitly mentioned in the table. These tasks will be collectively handled by all team members to ensure their timely completion. The deadline for completing any other tasks is set for 26/5, with no specific reviewer assigned.

Chapter 4: Analysis

4.1. Functional and Non-functional requirements

As every other project, the requirements analysis is the highest priority in the work chart. Understanding requirements is the key in creating a satisfactory application for clients. Thus, the team has tackled this problem first and created a detailed functional and non-functional requirements table to keep track and modify.

Req.ID	Requirement Name	Detailed Descriptions	Type
001	Register and login an account	Users can login to their account or register an account for the site	Functional Requirement
002	Product Catalog	The website should display all relevant information about the products (price, image, availability,...)	Functional Requirement
003	Shopping Cart	Users can add items into the shopping carts, remove items and view their carts before checking out	Functional Requirement
004	Payment Gateways	Payment gateways are required to facilitate transactions between customers and business	Functional Requirement
005	Order Management	Order Management allows customers to view their orders and businesses to manage their orders	Functional Requirement
006	Item Search Function	Search Function allows users to find items based on their requirements	Functional Requirement
007	Account Recovery	The website should support account recovery for users who have forgotten their credentials	Functional Requirement

4.2. Non-functional requirements

Req.ID	Requirement Name	Detailed Descriptions	Type
008	Performance	The website should have a fast loading times and be able to	Non-Functional Requirement

		handle high volume or traffic without crashing or slowing down	
009	Security	The website should be secure, with measures to protect customer data and prevent unauthorized access	Non-Functional Requirement
010	Usability	The website should be easy to navigate, with a clear and intuitive interface that allows users to quickly find what they are looking for	Non-Functional Requirement
011	Reliability	The website can be available most of the time, except for downtime for maintenance or update	Non-Functional Requirement
012	Scalability	The website should be able to handle growth and expansion, with the ability to add new products, users and features without compromising performance	Non-Functional Requirement

Chapter 5: System Design

System design involves many complicated phases which include various types of diagrams for the visualization of the system. With these diagrams, system design follows a strictly-observed principle. In this project, three types of diagrams will be used for various purposes.

5.1. Use Case Diagram

Use case diagrams provide a visual representation of the interactions between actors and the system, illustrating the functionalities that the system needs to support.

Actors: Customers, Administrators, Payment Gateway (e.g., Stripe).

Use Cases: Customers, Administrators, Payment Gateway (e.g., Stripe).

5.1.1. Use Case: Customers

Customer use case:

- **Login:** Customers can log in to their accounts.
- **Register:** Customers can create new accounts.

- **Edit Profile:** Customers can modify their profile information.
- **Browse Catalog:** Customers can explore the website's product catalog, filtering and searching for clothing items based on various criteria such as category, price, color, etc.
- **Add to Cart:** Customers can add selected products to their shopping cart for future purchase.
- **Checkout:** Customers can initiate the checkout process, providing billing and shipping information, and selecting a payment method.
- **Payment Processing:** The payment gateway handles the payment processing, verifying customer payment details and authorizing transactions.

5.1.2. Use Case: Administrators

Administrator use case:

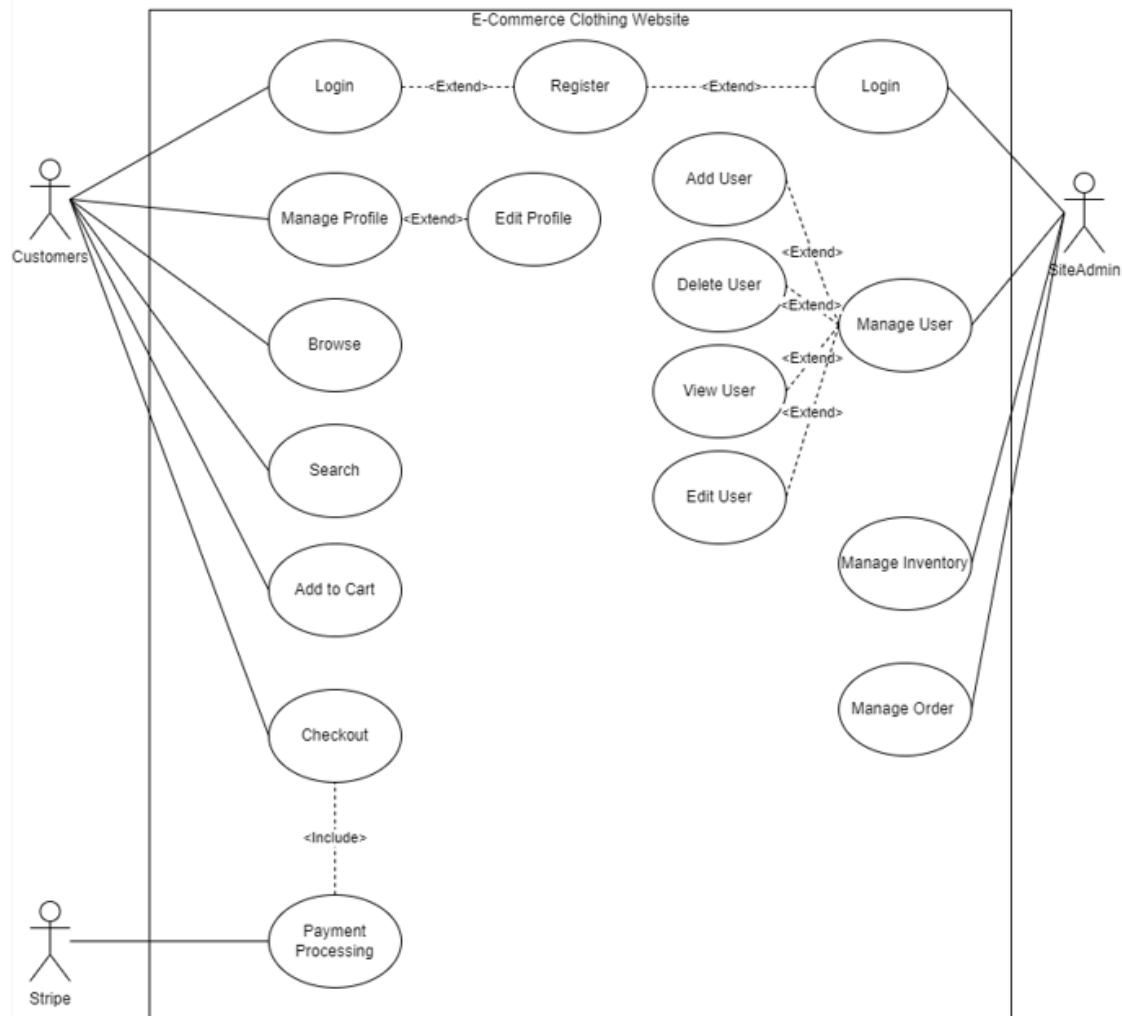
- **Login:** Administrators can log in to their accounts.
- **Manage Inventory:** Administrators can manage the product inventory, including adding new products, updating prices and descriptions, and removing out-of-stock items.
- **Manage Orders:** Administrators can manage customer orders, such as processing refunds, handling returns, and addressing customer service inquiries.
- **Manage Users:** Administrators can manage user accounts, including creating new accounts, updating account information, and resetting passwords.

5.1.3. Use Case: Payment Gateway (e.g., Stripe)

Payment Gateway use case:

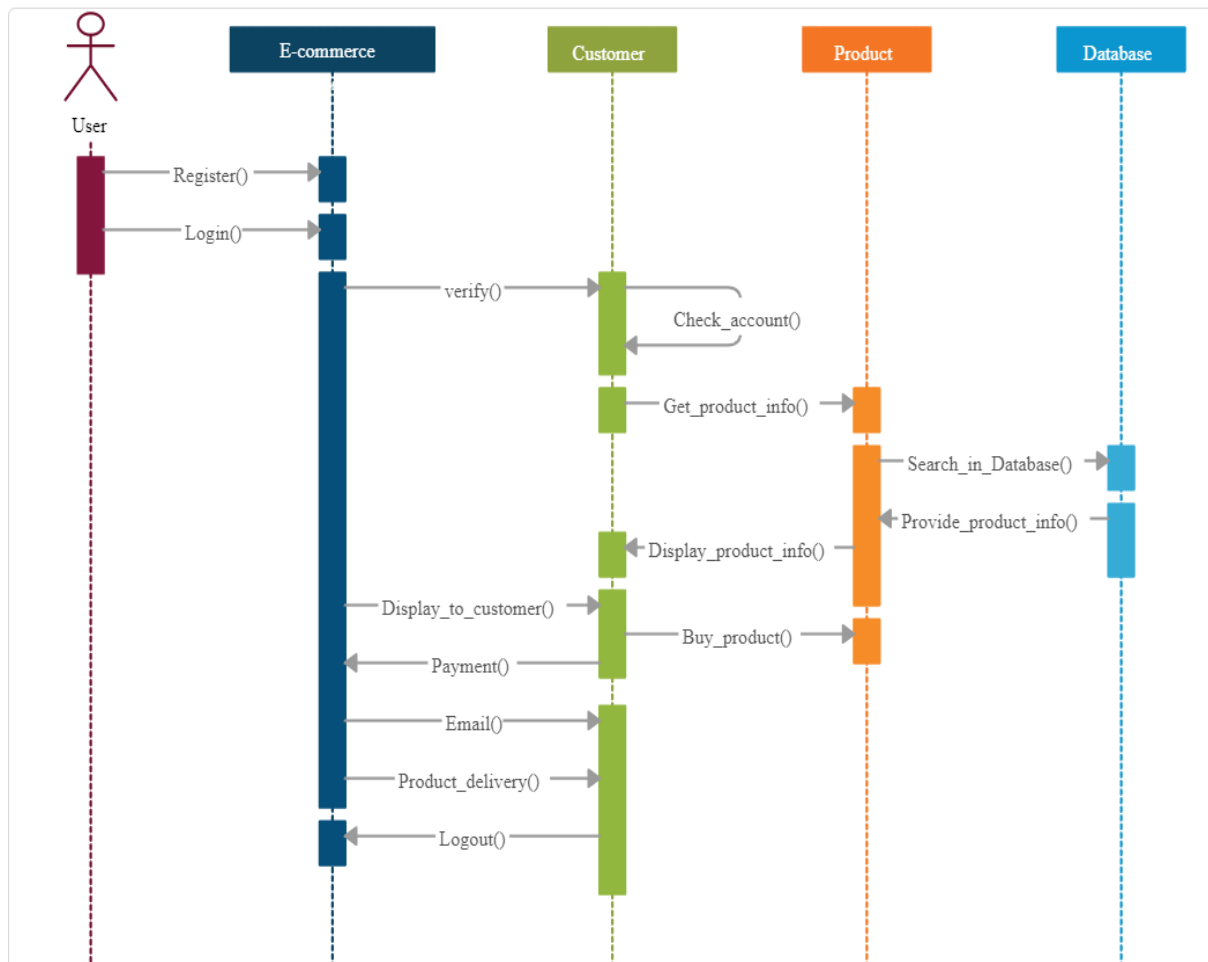
- **Payment Processing:** The payment gateway handles the payment processing, securely managing customer payment information, and facilitating transactions between customers and the business.

Use Case Diagram



Diagrams: [Use Case Diagram](#)

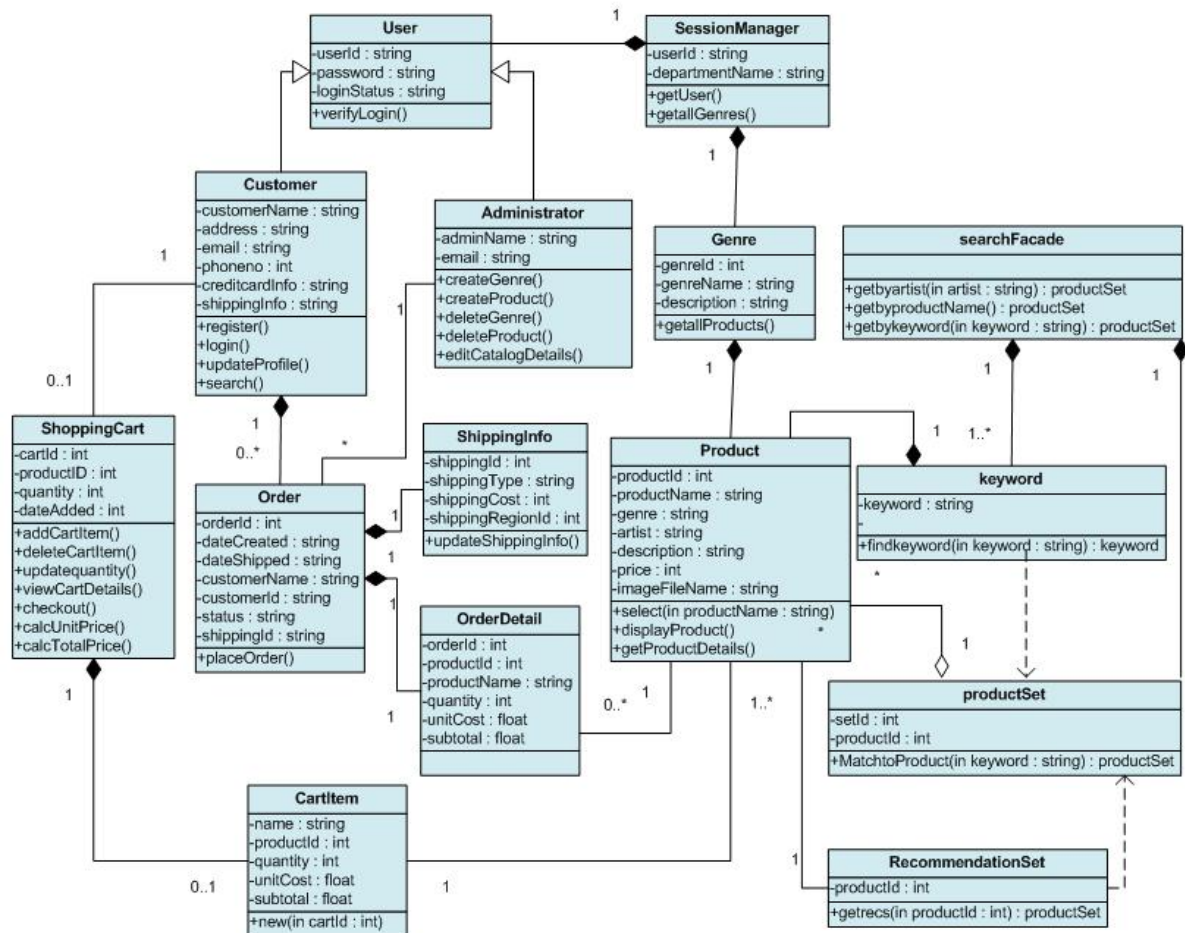
5.2. Sequence Diagram



Reference

<https://svg.template.creately.com/hbscitcx3>

5.3: Class Diagram



Reference

5.4. Database

5.4.1. Tables

The system's database schema includes the following tables/entities:

1. **User:** This table stores user information, including their name, email, and password. Each user can have a cart and proceed with payments.
2. **Product:** This table holds information about the available products for purchase on the website. It includes details such as the product name, description, price, and

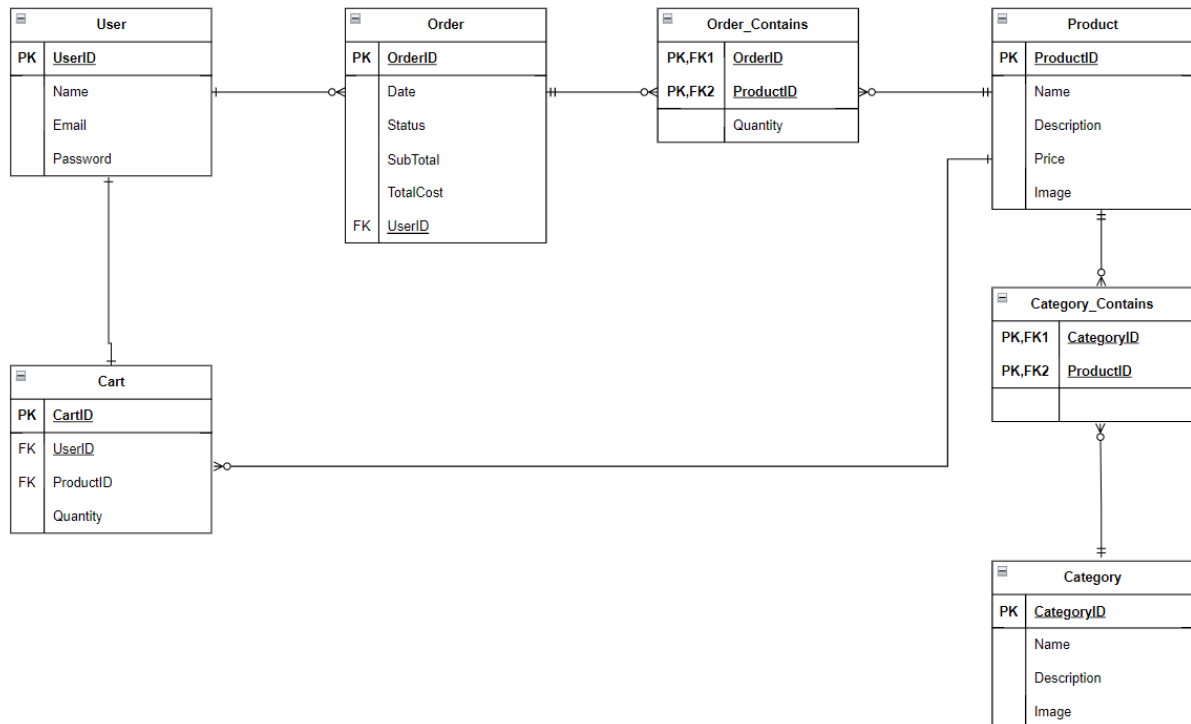
image. Each product can belong to multiple categories, and each category can contain multiple products. Additionally, each product can appear in multiple orders.

3. **Category:** This table stores information about the categories to which products belong, such as men's clothing, women's clothing, and shoes. Each category can contain multiple products, and each product can belong to multiple categories.
4. **Category_Contains:** This table serves as a bridge between the product and category tables. It contains the ProductID and CategoryID columns, enabling efficient query execution and association between products and categories.
5. **Cart:** This table stores information about user carts, including the CartID, UserID, ProductID, and Quantity. It keeps track of the products added to the cart and their respective quantities.
6. **Order:** This table holds information about the payments made for orders. It includes details such as the order amount, date, payment status, UserID, and shipping information.
7. **Order_Products:** This table stores information about the products included in each order and their respective quantities.

5.4.2. Relationships

The tables/entities are interconnected through the following relationships:

1. **User and Cart:** These entities have a one-to-one relationship, indicating that each user has a single cart and can add products to it.
2. **Product and Category:** These entities have a many-to-many relationship, indicating that each category can contain multiple products, and each product can belong to multiple categories. This relationship is facilitated by the Category_Contains table, which holds the ProductID and CategoryID for efficient query execution.
3. **User and Order:** These entities have a one-to-many relationship, indicating that each user can have multiple orders, but each order belongs to only one user.
4. **Product and Order:** These entities have a many-to-many relationship, indicating that each product can appear in multiple orders, and each order can contain multiple products. The Order_Products table, along with the quantity of each product, facilitates this relationship.
5. **Cart and Product:** These entities have a many-to-many relationship, indicating that each product can appear in multiple carts, and each cart can contain multiple products.



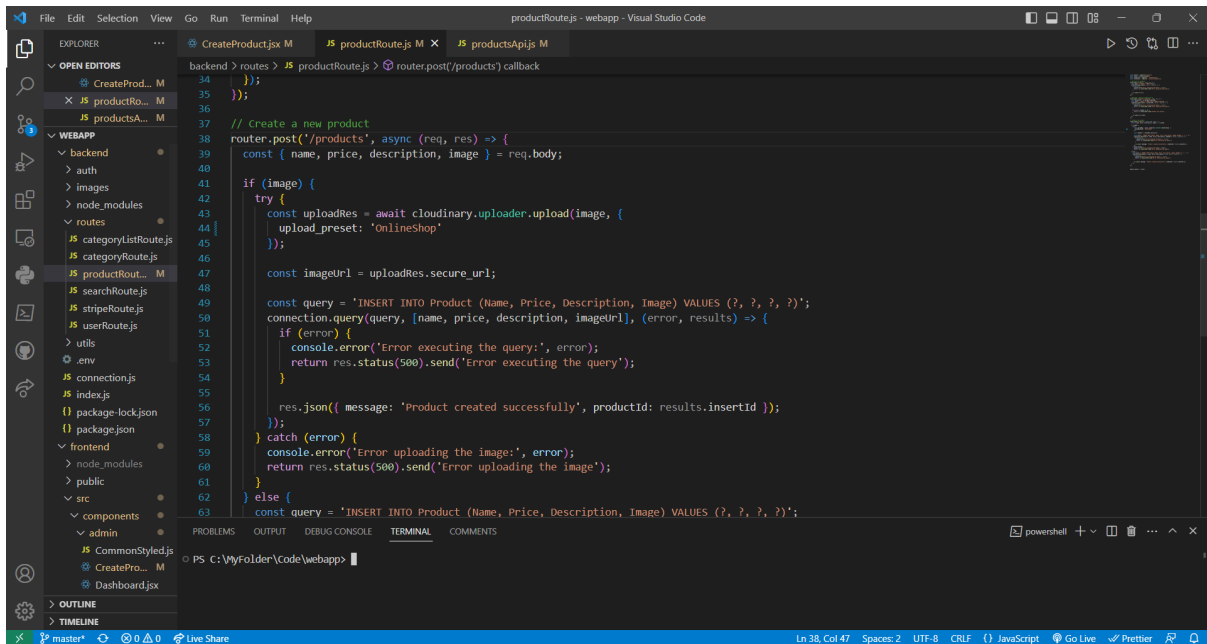
Diagrams: [ERD Diagram](#)

Chapter 6: Implementation

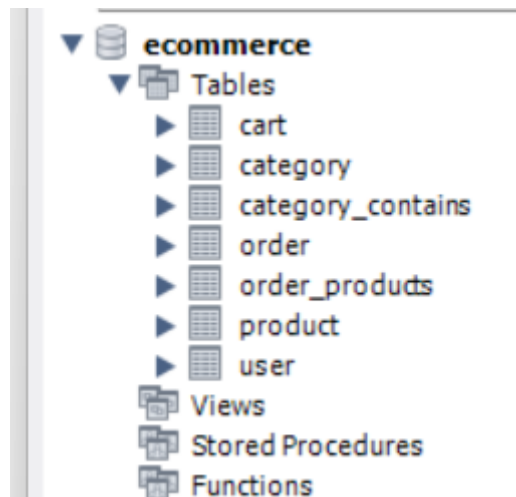
6.1. Setting up Workspace

In this section, we will set up the workspace for React, Node.js, Express, MySQL and Visual Studio Code (VSC).

- For VSC: [Visual Studio Code - Code Editing. Redefined](#)
- For Node.js: [Node.js \(nodejs.org\)](#)
- For React: [React](#)
- For Express: [Express - Node.js web application framework \(expressjs.com\)](#)
- For MySQL: [MySQL](#)



6.2. Setting up Database



6.2.1. Set up Database Configuration

- In the project directory, create a file called `.env` to store environment variables.
- Inside the `.env` file, add the following line to configure your MySQL database connection:

```

DB_CONNECTION=mysql
DB_HOST=your_mysql_host
DB_PORT=your_mysql_port
DB_DATABASE=your_database_name
DB_USERNAME=your_mysql_username
DB_PASSWORD=your_mysql_password

```

- Replace ``your_mysql_host``, ``your_mysql_port``, ``your_database_name``, ``your_mysql_username``, and ``your_mysql_password`` with your actual MySQL connection details.

6.2.2. Install MySQL Driver

- In project directory, run the following command to install the MySQL driver for Node.js:

```
npm install mysql
```

6.2.3. Connect to MySQL from Node.js

- In Node.js code, establish a connection to the MySQL database using the ``mysql`` package and the provided configuration.

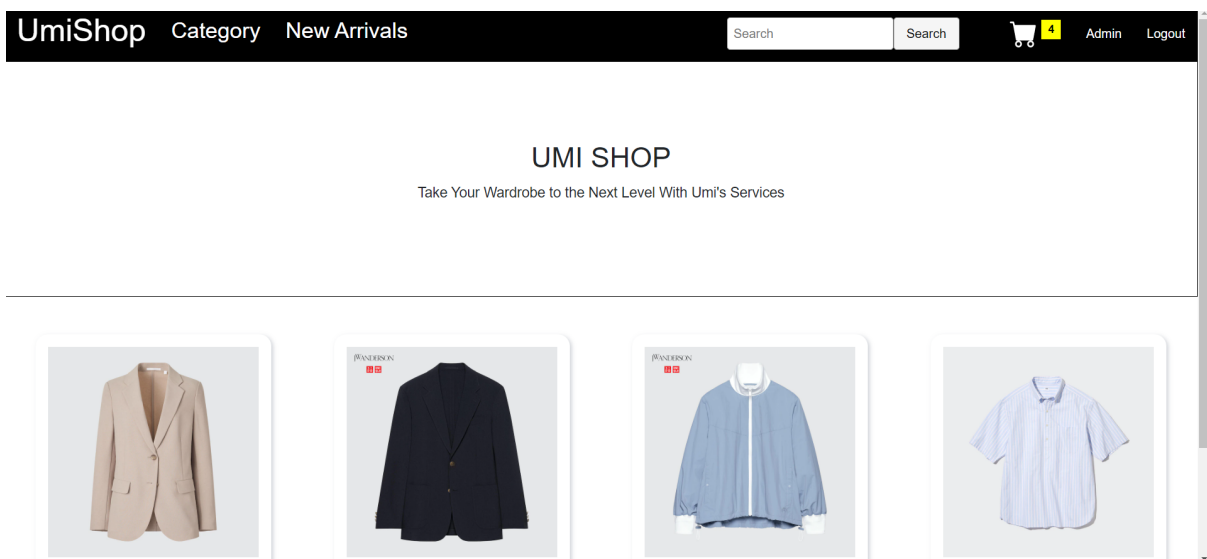
javascript

```
const mysql = require('mysql');
const connection = mysql.createConnection({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  database: process.env.DB_DATABASE,
  user: process.env.DB_USERNAME,
  password: process.env.DB_PASSWORD
});

connection.connect((err) => {
  if (err) {
    console.error('Error connecting to MySQL database:', err);
  } else {
    console.log('Connected to MySQL database');
  }
});
```

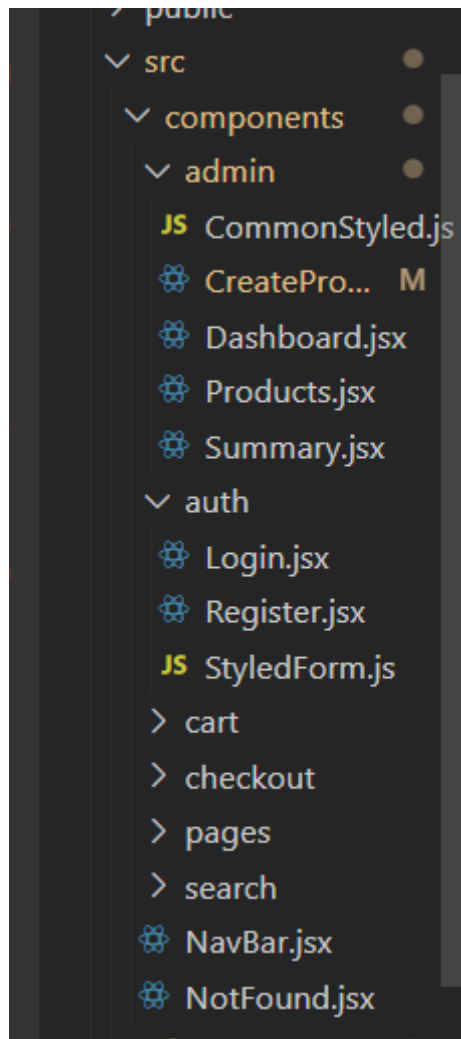
```
backend > JS connection.js > ...
1  const mysql = require('mysql');
2
3  const connection = mysql.createConnection({
4    host: 'localhost',
5    user: 'root',
6    password: 'root123',
7    database: 'ecommerce',
8  });
9
10 connection.connect((err) => {
11   if (err) {
12     console.error('Error connecting to the database:', err);
13     return;
14   }
15   console.log('Connected to the database');
16 });
17
18 module.exports = connection;
19
```

6.3. Coding Frontend



6.3.1. Create React Components

- Inside the `src/components` directory, create individual files for React components.
- Define and implement each component according to application's requirements.



6.3.2. Use React Router for Routing

- Install React Router by running the following command in project directory:
npm install react-router-dom
- Define the application's routes using React Router's `` component and implement the necessary components for each route.

```

# CreateProduct.jsx M    JS productRoute.js M    JS App.js    X    JS productsApi.js M
frontend > src > JS App.js > ...
27   return (
28     <div>
29       <BrowserRouter>
30         <NavBar />
31         <Switch>
32           <Route path="/cart" exact component={Cart} />
33           <Route path="/register" exact component={Register} />
34           <Route path="/login" exact component={Login} />
35           <Route path="/not-found" component={NotFound} />
36           <Route path="/" exact component={HomePage} />
37           <Route path="/product/:ProductID" component={Product} />
38           <Route path="/category" exact component={Category} />
39           <Route path="/search/:searchItem" exact component={Search} />
40           <Route path="/category/:CategoryName" component={ProductCategory} />
41           <Route path="/checkout-success" component={CheckoutSuccess} />
42           <Route path="/admin">
43             <Dashboard>
44               <Route path="/admin/products" component={Products} />
45               <Route path="/admin/products/create-product" component={CreateProduct} />
46               <Route path="/admin/summary" component={Summary} />
47             </Dashboard>
48           </Route>
49           <Route path="*" component={NotFound} />
50         </Switch>
51       </BrowserRouter>
52       <ToastContainer />
53     </div>
54   );
55 }
56

```

6.3.3. Use Axios for API Calls

- Install Axios, a popular HTTP client, by running the following command:
npm install axios
- Use Axios to make API calls to the backend server and handle the responses.

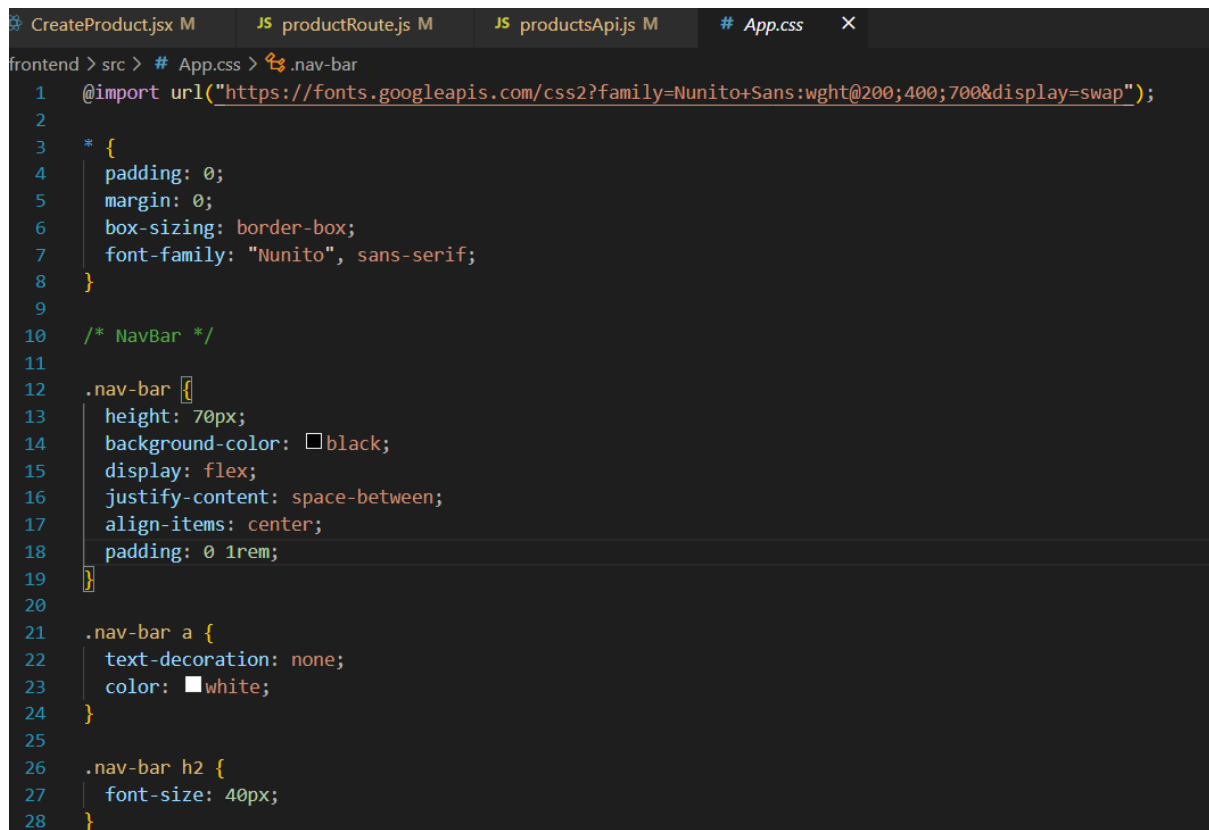
```

# CreateProduct.jsx M    JS productRoute.js M    JS productsApi.js M    JS authenticate.js X
frontend > src > features > JS authenticate.js > ...
1   import axios from "axios";
2
3   const authenticate = async (formData, endpoint) => {
4     try {
5       const response = await axios.post(
6         `http://localhost:5000/${endpoint}`,
7         formData
8       );
9       return response.data;
10    } catch (error) {
11      console.error(error);
12      throw error;
13    }
14  };
15
16  export default authenticate;
17

```

6.3.4. Style your Components

- Utilize CSS frameworks (e.g., Bootstrap, Material-UI) or custom CSS to style components.
- Create CSS files or use CSS-in-JS libraries (e.g., styled-components) to manage component styling.

A screenshot of a code editor with a dark theme. The editor shows a CSS file named 'App.css' with the following content:

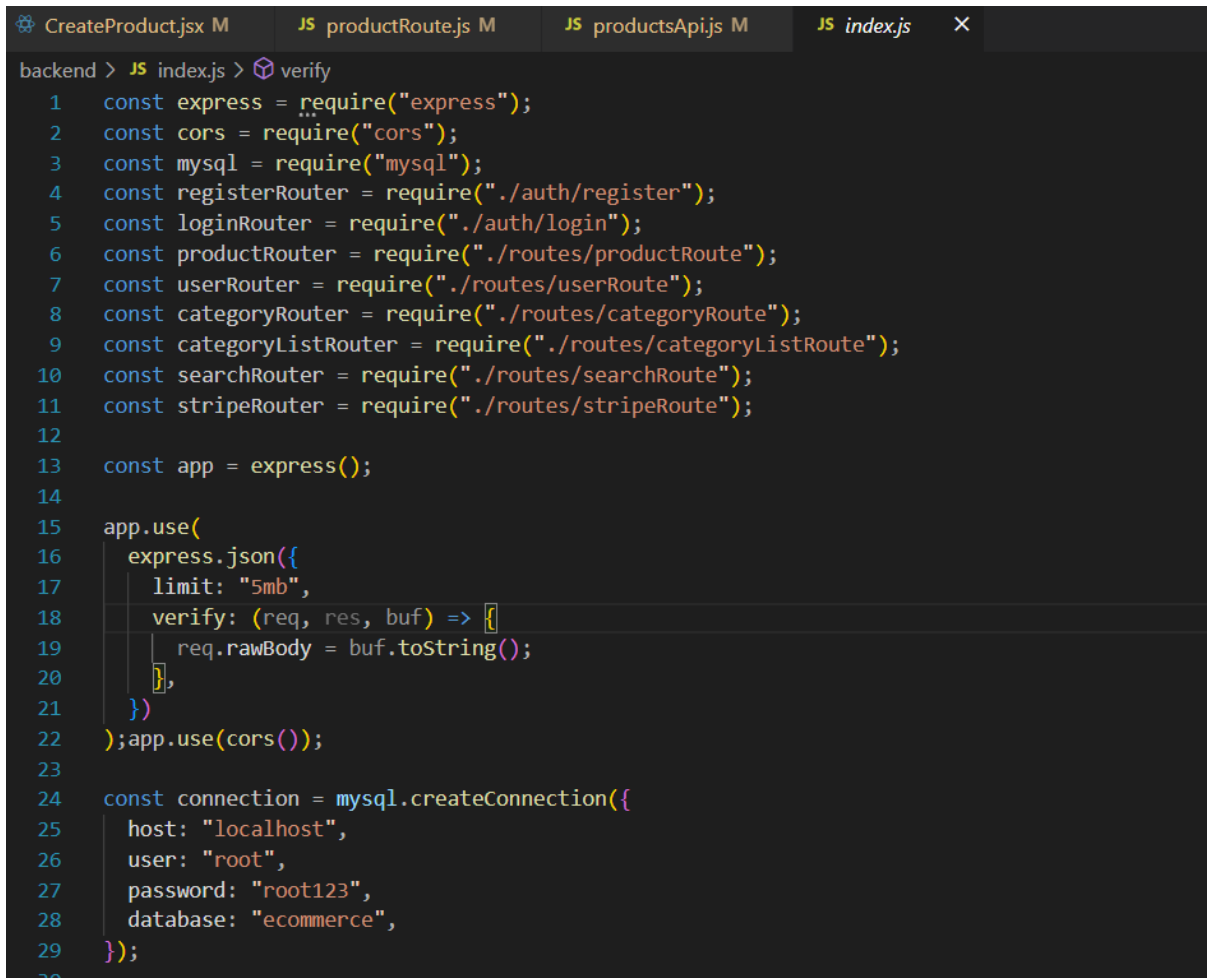
```
1  @import url("https://fonts.googleapis.com/css2?family=Nunito+Sans:wght@200;400;700&display=swap");
2
3  * {
4    padding: 0;
5    margin: 0;
6    box-sizing: border-box;
7    font-family: "Nunito", sans-serif;
8  }
9
10 /* NavBar */
11
12 .nav-bar {
13   height: 70px;
14   background-color: black;
15   display: flex;
16   justify-content: space-between;
17   align-items: center;
18   padding: 0 1rem;
19 }
20
21 .nav-bar a {
22   text-decoration: none;
23   color: white;
24 }
25
26 .nav-bar h2 {
27   font-size: 40px;
28 }
```

The editor's tab bar at the top shows several files: 'CreateProduct.jsx M', 'JS productRoute.js M', 'JS productsApi.js M', and '# App.css X'. The command line at the bottom of the editor shows the path 'frontend > src > # App.css > .nav-bar'.

6.4. Coding Backend

6.4.1. Create Express Routes

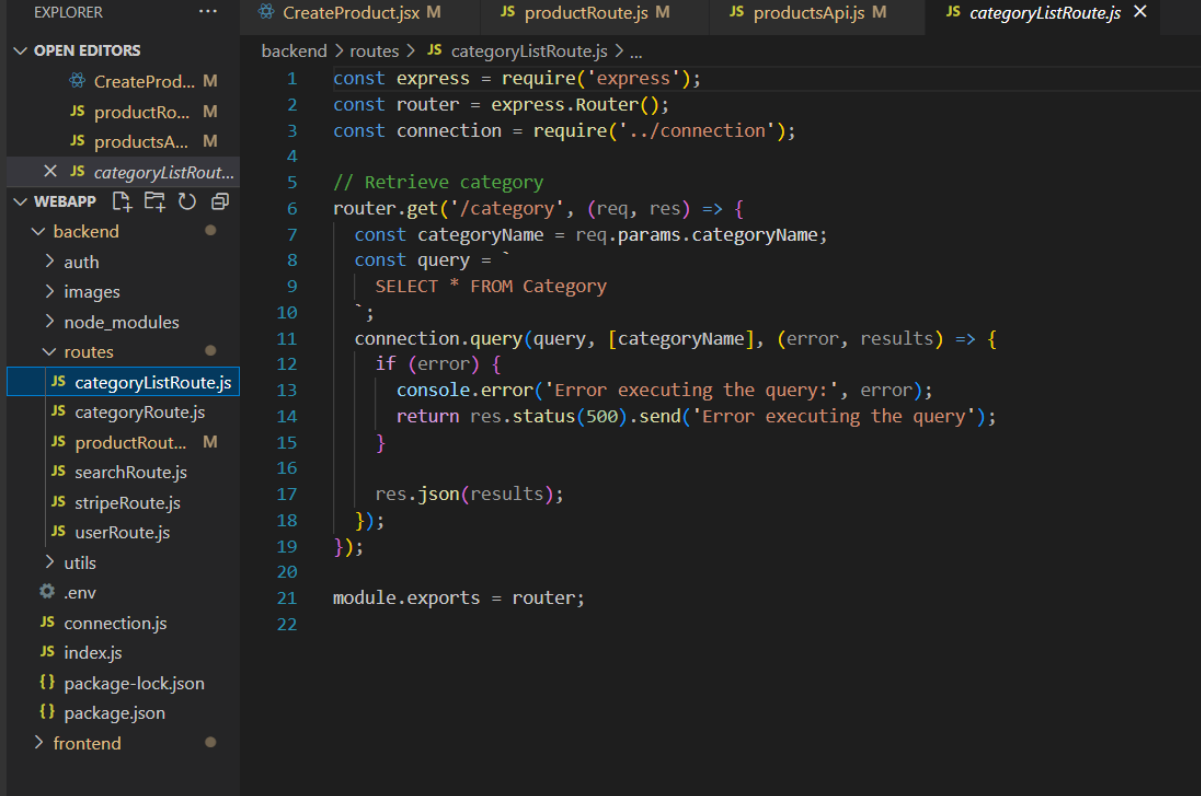
- Inside the 'index.js' file, define and implement Express routes to handle various API requests.
- Use the 'express.Router()' function to create modular route handlers for different parts of application.



```
backend > JS index.js > verify
1  const express = require("express");
2  const cors = require("cors");
3  const mysql = require("mysql");
4  const registerRouter = require("./auth/register");
5  const loginRouter = require("./auth/login");
6  const productRouter = require("./routes/productRoute");
7  const userRouter = require("./routes/userRoute");
8  const categoryRouter = require("./routes/categoryRoute");
9  const categoryListRouter = require("./routes/categoryListRoute");
10 const searchRouter = require("./routes/searchRoute");
11 const stripeRouter = require("./routes/stripeRoute");
12
13 const app = express();
14
15 app.use(
16   express.json({
17     limit: "5mb",
18     verify: (req, res, buf) => {
19       req.rawBody = buf.toString();
20     },
21   })
22 ); app.use(cors());
23
24 const connection = mysql.createConnection({
25   host: "localhost",
26   user: "root",
27   password: "root123",
28   database: "ecommerce",
29 });
30
```

6.4.2. Implement Controllers and Business Logic

- Create separate controller files to handle the logic for each route.
- Implement the necessary functions to process incoming requests, interact with the database, and send appropriate responses.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under 'WEBAPP'. The 'routes' directory is expanded, and 'categoryListRoute.js' is selected. The main editor area shows the code for 'categoryListRoute.js' with the following content:

```
backend > routes > JS categoryListRoute.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const connection = require('../connection');
4
5  // Retrieve category
6  router.get('/category', (req, res) => {
7      const categoryName = req.params.categoryName;
8      const query = `
9          SELECT * FROM Category
10      `;
11      connection.query(query, [categoryName], (error, results) => {
12          if (error) {
13              console.error('Error executing the query:', error);
14              return res.status(500).send('Error executing the query');
15          }
16          res.json(results);
17      });
18  });
19
20
21  module.exports = router;
22
```

6.4.3. Connect to the Database

- Use the MySQL driver (as described in section 6.2) to connect to the MySQL database from backend code.
- Implement database queries and operations within controllers to perform CRUD (Create, Read, Update, Delete) operations.

```

1  const mysql = require('mysql');
2
3  const connection = mysql.createConnection({
4    host: 'localhost',
5    user: 'root',
6    password: 'root123',
7    database: 'ecommerce',
8  });
9
10 connection.connect((err) => {
11   if (err) {
12     console.error('Error connecting to the database:', err);
13     return;
14   }
15   console.log('Connected to the database');
16 });
17
18 module.exports = connection;
19

```

6.4.4. Handle Authentication and Authorization

- Implement user authentication and authorization mechanisms.
- Secure routes based on user roles and permissions.

```

1  const express = require("express");
2  const bcrypt = require("bcrypt");
3  const connection = require('../connection');
4  const router = express.Router();
5
6  router.post("/login", (req, res) => {
7    const { Username, Password } = req.body;
8
9    // Check if the provided username and password match a user in the database
10   const query = "SELECT * FROM User WHERE Username = ?";
11   connection.query(query, [Username], (error, results) => {
12     if (error) {
13       console.error(error);
14       return res.status(500).json({ message: "Failed to login" });
15     }
16
17     if (results.length === 0) {
18       return res.status(401).json({ message: "Invalid username or password" });
19     }
20
21     const user = results[0];
22
23     // Compare the provided password with the stored hashed password
24     bcrypt.compare(Password, user.Password, (error, isMatch) => {
25       if (error) {
26         console.error(error);
27         return res.status(500).json({ message: "Failed to login" });
28       }
29
30       if (!isMatch) {

```

6.4.5. Error Handling and Middleware

- Implement error handling middleware to catch and handle errors that occur during API requests.

6.5. Stripe

- For Stripe: [Stripe | Payment Processing Platform for the Internet](#)

The screenshot displays a Stripe checkout interface. On the left, a cart summary shows two items: 'shirt 1' (Qty 3, Brown) for ₫1,350,000 and 'test' (Qty 1) for ₫100,000. The subtotal is ₫1,450,000, shipping is free, and the total due is ₫1,450,000. On the right, the 'Pay with card' section includes a 'Shipping information' form with fields for Email, Name, Country (Vietnam), Address line 1, Address line 2, City, Province, and Postal code (091 234 56 78). Below this, the 'Shipping method' section offers 'Free shipping' (5-7 business days, Free) and 'Next day air' (1 business day, ₫50,000).

6.5.1. Sign up for a Stripe Account

- Visit the Stripe website at <https://stripe.com> and sign up for an account.
- Follow the verification process and complete the necessary steps to activate the account.

6.5.2. Install the Stripe Node.js Library

- In project directory, run the following command to install the Stripe library:
`npm install stripe`

6.5.3. Set up Stripe API Keys

- Retrieve Stripe API keys from Stripe account dashboard.
- Store API keys and access them in backend code.

6.5.4. Implement Stripe Integration

- Use the Stripe Node.js library to handle payment processing within the application.

- Follow the Stripe API documentation to implement functionalities such as creating charges, managing customers, and handling webhooks.

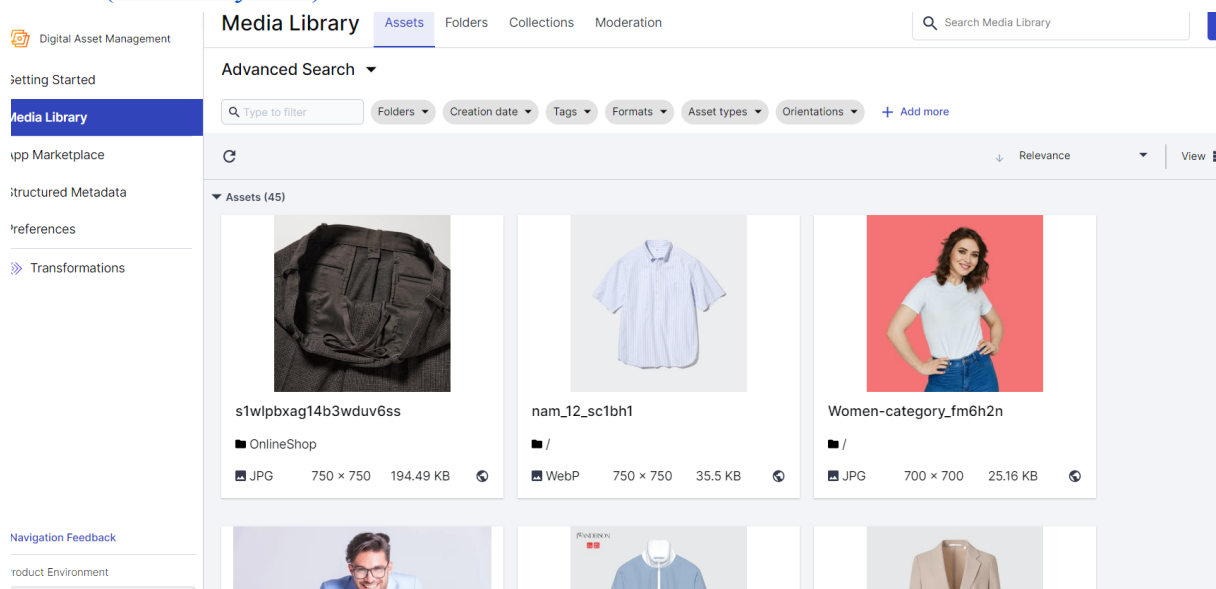
```

1 const express = require("express");
2 const router = express.Router();
3 const connection = require("../connection");
4 const util = require("util");
5 const queryPromise = util.promisify(connection.query).bind(connection);
6 const stripe = require("stripe")(
7   "sk_test_51NAV7gIu8TWD2EDaCDcPlcoP03rOug1aVumex3l7oqaHvKYNBHD85Stq9Z10fuz05w1x4hu2YbYtt0rs10DhKsU00z6lwn8S4"
8 );
9 const bodyParser = require("body-parser");
10 router.use(bodyParser.raw({ type: "application/json" }));
11
12 router.post("/create-checkout-session", async (req, res) => {
13   const customer = await stripe.customers.create({
14     metadata: {
15       userID: req.body.userID,
16       cart: JSON.stringify(req.body.cartItems),
17     },
18   });
19
20   const line_items = req.body.cartItems.map((item) => {
21     return {
22       price_data: {
23         currency: "vnd",
24         product_data: {
25           name: item.Name,
26           images: [item.Image],
27           description: item.Description,
28           metadata: {
29             id: item.ProductID,
30           },
31         },
32       },
33     };
34   });
35 }

```

6.6. Cloudinary

- For Cloudinary: [Image and Video Upload, Storage, Optimization and CDN \(cloudinary.com\)](https://cloudinary.com)



6.6.1. Sign up for a Cloudinary Account

- Visit the Cloudinary website at <https://cloudinary.com> and sign up for an account.
- Complete the registration process and create a new cloudinary project.

6.6.2. Install the Cloudinary Node.js SDK

- In project directory, run the following command to install the Cloudinary SDK:
npm install cloudinary

6.6.3. Set up Cloudinary API Credentials

- Retrieve Cloudinary API credentials from Cloudinary account dashboard.
- Store API credentials securely, and access them in backend code.

6.6.4. Implement Cloudinary Integration

- Use the Cloudinary Node.js SDK to upload, manage, and serve media assets within the application.
- Follow the Cloudinary API documentation to implement functionalities such as uploading images, generating transformations, and handling media assets.

```
// Create a new product
router.post('/products', async (req, res) => {
  const { name, price, description, image } = req.body;

  if (image) {
    try {
      const uploadRes = await cloudinary.uploader.upload(image, {
        upload_preset: 'OnlineShop'
      });

      const imageUrl = uploadRes.secure_url;

      const query = 'INSERT INTO Product (Name, Price, Description, Image) VALUES (?, ?, ?, ?)';
      connection.query(query, [name, price, description, imageUrl], (error, results) => {
        if (error) {
          console.error('Error executing the query:', error);
          return res.status(500).send('Error executing the query');
        }

        res.json({ message: 'Product created successfully', productId: results.insertId });
      });
    } catch (error) {
      console.error('Error uploading the image:', error);
      return res.status(500).send('Error uploading the image');
    }
  } else {
    const query = 'INSERT INTO Product (Name, Price, Description) VALUES (?, ?, ?)';
    connection.query(query, [name, price, description], (error, results) => {
      if (error) {
        console.error('Error executing the query:', error);
        return res.status(500).send('Error executing the query');
      }

      res.json({ message: 'Product created successfully', productId: results.insertId });
    });
  }
});
```

6.7. Admin Dashboards

The screenshot displays the UmiShop Admin Dashboard. The top navigation bar includes the 'UmiShop' logo, 'Category', 'New Arrivals', and search bars. A shopping cart icon with a '4' badge and 'Admin'/'Logout' links are on the right. The left sidebar shows 'QUICK LINKS' with 'Summary' and 'Products' (highlighted). The main content area is titled 'Products' and features a 'Create' button. The 'Create a Product' form includes a file upload field (labeled 'Choose File' and 'No file chosen'), input fields for 'Name', 'Price', and 'Short Description', and a 'Submit' button. An image preview box on the right contains the text 'Image Preview will appear here!'.

6.7.1. Design the Admin Dashboard

- Plan the layout and features of the admin dashboard.
- Consider using UI libraries or frameworks (e.g., React Admin, Material-UI) to expedite the development process.

6.7.2. Implement Admin Routes

- Create separate routes and components specifically for the admin dashboard.
- Secure these routes to ensure only authorized users can access the admin features.

6.7.3. Build Admin Functionality

- Implement CRUD operations for managing various data entities within your application.
- Create forms, tables, and other UI components to facilitate data management.

6.7.4. Apply Security Measures

- Implement authentication and authorization mechanisms to ensure only authorized administrators can access and modify data.
- Set up role-based access control (RBAC) to manage user permissions and restrict access to sensitive functionalities.

6.8. Project Github

Github: [DuyVu285/webapp \(github.com\)](https://github.com/DuyVu285/webapp)

Chapter 7: Deployment

Deploying a MERN (MySQL, Express, React, Node.js) project involves several steps to ensure that your application is ready to be hosted and accessible to users. Here is a general guide on how to deploy a MERN project:

7.1. Prepare Your Project

- Ensure that the project is fully functional and tested in a local development environment.
- Set up a version control system (such as Git) to manage your project's codebase.
- Update any sensitive information (e.g., database credentials, API keys) to use environment variables instead of hardcoding them in the code.

7.2. Choose a Hosting Provider

- Research and select a hosting provider that supports the technologies used in the project. Some popular options for hosting MERN projects include Heroku, AWS, Azure, and DigitalOcean.
- Consider factors like pricing, scalability, ease of deployment, and support for the project's specific requirements.

7.3. Set Up Hosting Environment

- Create an account with the chosen hosting provider and set up a new server or environment to host the project.
- Configure any necessary server settings, such as installing Node.js and setting up a database (if required).

7.4. Build and Bundle React App

- In project's root directory, run the command to build React app:
`npm run build`
- This command will create a production-ready build of your React app, optimized for performance.

7.5. Set Up Deployment Configuration

- Configure your hosting provider's deployment settings to specify how your project will be deployed.
- This may involve defining the project's root directory, specifying the entry point for your Node.js server, and setting environment variables.

7.6. Deploy Your Backend (Node.js/Express)

- Upload your project's server-side code to the hosting environment.
- Install the project's dependencies by running the command:
`npm install`
- Start your Node.js server using a process manager like PM2, which will ensure that your server stays up and running even if it crashes or encounters errors.

7.7. Set Up Database

- If your project uses a database (such as MySQL), set up the database on your hosting provider or configure a separate database service.
- Update your project's configuration files to connect to the remote database instead of the local one.

7.8. Deploy Your Frontend (React)

- Upload the built React app (created in step 4) to your hosting environment.
- Configure the hosting environment to serve the React app as a static website.

Chapter 8: Conclusion

8.1. Summary of the project

Throughout this project, we embarked on the development of an e-commerce website using the MERN stack, which consists of MySQL, Express, React, and Node.js. Our objective was to create a robust and user-friendly platform that facilitates online transactions and provides an engaging shopping experience for customers.

Starting with the backend, we utilized Node.js and Express to build a scalable and efficient server. By integrating MySQL as our database management system, we were able to store and retrieve data related to products, user information, and transactions. This allowed us to create a solid foundation for the website's functionality.

On the frontend, we employed React, a popular JavaScript library, to design the user interface and handle the dynamic rendering of components. This enabled us to build a responsive and interactive website, providing seamless navigation and an immersive shopping experience for our users.

Key features of the e-commerce website included product listings, shopping cart functionality, secure payment processing, user authentication and authorization. These features were implemented using various technologies from the MERN stack, allowing us to create a comprehensive and fully functional online shopping platform.

8.2. Challenges faced and lessons learned

Throughout the development process, we encountered several challenges and gained valuable insights that contributed to our growth as developers. Some of the challenges we faced included:

1. **Learning and Utilizing the MERN Stack:** As our team was relatively new to the MERN stack, we had to invest time in understanding and effectively utilizing each technology. This involved learning the nuances of Node.js, Express, React, and MySQL, and integrating them seamlessly to create a cohesive web application.
2. **Learning New Technologies:** In addition to the MERN stack, we had to learn and incorporate other technologies and libraries into our project. This included using payment gateways, implementing secure authentication and authorization mechanisms, and integrating third-party APIs for enhanced functionality.
3. **Developing an E-commerce Website:** Building a robust and user-friendly e-commerce website involves tackling various complexities, such as handling product catalogs, managing inventory, implementing shopping cart functionality, and securely processing payments. We had to ensure the website was responsive, intuitive, and optimized for performance.

From these challenges, we learned important lessons that will guide us in future projects:

1. **Continuous Learning:** Technology evolves rapidly, and it is crucial to stay updated with the latest advancements. By maintaining a mindset of continuous learning, we can adapt to new technologies and leverage them effectively in our projects.
2. **Effective Collaboration:** Building a complex web application requires collaboration and effective communication within the development team. Regular meetings, clear task allocation, and maintaining documentation helped us stay organized and meet project goals efficiently.
3. **User-Centric Approach:** Throughout the development process, we realized the importance of focusing on the needs and preferences of our users. Regular user feedback and usability testing helped us improve the website's design and functionality, ensuring a positive user experience.

8.3. Possible future improvements or extensions

While we have successfully developed a functional e-commerce website using the MERN stack, there are several potential areas for future improvements and extensions. Some of the possibilities include:

1. **User Profile Editing:** Implementing a user profile editing feature would allow users to update their personal information, manage saved addresses, and modify their preferences.
2. **Improved Login and Registration Process:** Enhancing the login and registration process with additional security measures, such as two-factor authentication or social media integration, can provide users with a more secure and convenient experience.
3. **Search Optimization:** Enhancing the search functionality to provide more accurate and relevant results can improve the overall user experience. This could involve

implementing advanced search algorithms, autocomplete suggestions, and filters for refined search results.

4. **Footer:** Designing and incorporating a footer section on the website can provide important links, contact information, and additional navigation options, ensuring easy access to essential pages.
5. **Branding:** Developing a distinct brand identity for the e-commerce website, including a logo, color scheme, and visual elements, can help establish a strong brand presence and enhance user recognition.

By addressing these potential areas for improvement, the e-commerce website can be further enhanced to meet evolving user expectations and industry standards.

In conclusion, the development of the MERN-based e-commerce website was a challenging yet rewarding experience. We successfully utilized the MERN stack, learned new technologies, and developed a fully functional e-commerce platform. The project taught us valuable lessons in utilizing the MERN stack effectively, learning new technologies, and building an e-commerce website. Moving forward, the identified future improvements and extensions provide a roadmap for enhancing the website's functionality, user experience, and overall success.

References

- For VSC: [Visual Studio Code - Code Editing. Redefined](#)
- For Node.js: [Node.js \(nodejs.org\)](#)
- For React: [React](#)
- For Express: [Express - Node.js web application framework \(expressjs.com\)](#)
- For MySQL: [MySQL](#)
- For Stripe: [Stripe | Payment Processing Platform for the Internet](#)
- For Cloudinary: [Image and Video Upload, Storage, Optimization and CDN \(cloudinary.com\)](#)
- Diagrams: [Use Case Diagram](#)
- Github: [DuyVu285/webapp \(github.com\)](#)