



MCA Semester – IV

Project – Final Report

Name	Group 13 FSD E-Commerce Web
Project	Shop Hub(E-Commerce)
Group	13
Date of Submission	22/11/2023



A study on “**SHOPHUB**”

A Project submitted to Jain Online (Deemed-to-be University)

In partial fulfilment of the requirements for the award of:

Master of Computer Application

Submitted by:

Under the guidance of:

Mention your Guide's Name

Chandra Bhanu

Jain Online (Deemed-to-be University)

Bangalore

2022-23

DECLARATION

We, the below mentioned students, hereby declare that the Project Report titled “*Shop hub*” has been prepared by us under the guidance of the *Chandra Banu*. We declare that this Project work is towards the partial fulfilment of the University Regulations for the award of the degree of Master of Computer Application by Jain University, Bengaluru. We have undertaken a project for a period of one semester. We further declare that this Project is based on the original study undertaken by us and has not been submitted for the award of any degree/diploma from any other University / Institution.

Place: Hyderabad,

Date:22-11-23

*Names of the Student with
USNs:*

ACKNOWLEDGEMENT

I express my sincere appreciation to the entire project team for their collaborative effort and dedication in bringing our e-commerce website to fruition. I extend my deepest gratitude to every member of our team who played an integral role in the successful completion of our e-commerce website project during our college course.

My fellow developers contributed their expertise in crafting a robust and feature-rich platform, while the design team's creativity brought a visually appealing and user-friendly interface to life. The meticulous work of the testing and quality assurance teams ensured a reliable and secure final product. Our project mentor provided invaluable guidance, ensuring effective coordination and timely progress.

Special thanks to my mentor for their guidance and support throughout this journey. This project has been a tremendous learning experience, and I am proud to have been part of a team that demonstrated exceptional skills, resilience, and a shared commitment to success.

EXECUTIVE SUMMARY

Internet is the rapidest growing media during the past decade. Especially, online shopping is a rapidly growing e-commerce area. Online stores are usually available 24 hours a day, and many consumers have Internet access both at work and at home. A successful web store is not just a good-looking website with dynamic technical features, listed in many search engines. This study aims to establish a preliminary assessment, evaluation and understanding of the characteristics of online shopping. Although the benefits of online shopping are considerable, when the process goes poorly it can create a thorny situation. A few problems that shoppers potentially face include identity theft, faulty products, and the accumulations of spyware, as well as the precautions to be taken are studied in this paper.

Shop hub (e-commerce website), a dynamic and forward-thinking business, is poised to revolutionize the online retail landscape with the launch of its new e-commerce website. This executive summary provides an overview of key elements and strategies that will drive the success of our online platform.

Market Opportunity: The global e-commerce market is experiencing unprecedented growth, with consumers increasingly shifting towards online shopping. Our research indicates a significant opportunity to capture market share by offering a user-friendly and innovative platform that caters to evolving consumer preferences.

TABLE OF CONTENTS

Title	Page Nos.
Executive Summary	iii
Introduction	1- 4
System Architecture	5-6
Technologies Used	7-12
Front-end Development	13-62
Back-end Development	63-101
Database Design	102-104
Authentication and Security	105-106
Project Management	107-110
Results and Evaluation	111-116
Conclusion	117

1. Introduction

1.1 Research Background

Nowadays, the world of e-commerce is developing very strongly. Digitalization helps people to save more costs transported through intermediaries, transaction costs and it significantly helps to save time to invest in other activities. Moreover, e-commerce also allows people to search for many different purposes, provide information according to human needs and preferences. Customers can sit at home and buy everything they want using online sales websites. Open-source technologies are very important in building sales websites efficiently with low costs. Usually open-source technologies are so widely used that there is excellent support available on the Internet. The provided support enables building user-friendly sales websites quickly. Therefore, the topic: “Building an E-commerce Website” was selected for this thesis. An e-commerce website is a simple but powerful enough system which allows rapid construction of sales applications on the Internet.

1.2 E-Commerce

Over the past decade, consumers have continuously changed the way they want to shop, especially in terms of shopping behaviour on e-commerce channels and websites, forcing businesses to find ways to adapt to this trend. With the support of newest technology, it is easier for consumers to find, compare and purchase from online websites, e-commerce markets, mobile applications, and physical stores and social sites, rather than following traditional commercial models. According to research by Get App (Factory 2021), the leading review software in the world, they have given the following Figures about consumer needs and habits of customers:

- Customers like to research and buy products online with 53.1%.
- Customers prefer to research online and buy offline with 28.9%.

- Only 18% of customers say they prefer to going to stores.



Figure 1. E-commerce Research (Factory 2023).

1.2.1 Four Types of E-commerce

B2C: BUSINESS TO CONSUMERS

B2C e-commerce includes transactions made between businesses and consumers. This is one of the most widely used sales models in the e-commerce landscape. For example, when customers buy shoes online from the manufacturer Nike, this is a business-to-consumer transaction.

B2B: BUSINESS TO BUSINESS

B2B e-commerce involves commercial activities carried out between businesses, such as between manufacturers and dealers or retailers. Typically, business-to-business sales will often focus on raw materials, or packaged products.

C2C: CONSUMER TO CONSUMER

One of the earliest established e-commerce business models is the C2C e-commerce business model (The-Future-Of Customer Engagement and Experience 2021; Bloomidea, 2021). This includes customer-to-customer relationships, for example on Shopee, Amazon, and Lazada, Alibaba.

C2B: CONSUMER TO ENTERPRISE

C2B is a business model that reverses the traditional e-commerce model. C2B means the individual consumer produces the product or service, the business will be the one who buys it. (The Future Of Customer Engagement and Experience 2021; Bloomidea, 2021).

1.2.2 Advantages of E-commerce Website

- Increased profits**

E-commerce websites biggest advantage is that E-commerce is not limited in time and space like a traditional form of sales. Via e-commerce websites businesses can actively look for customers to their products and services. In addition, in order to increase profits, businesses must provide products with guaranteed quality, in accordance with the needs of users, reasonable prices with enthusiastic support.

- Cost savings**

Cost savings are the benefits that attract many businesses to invest in owning an e-commerce website. Because businesses do not need to spend a large amount of money to hire and train staff but it still gets high profits if they know how to combine with some online marketing strategies.

- Increased interactivity with customers**

When customers visit an e-commerce website, it is easier for them to update prices, product information and services. Besides, any time the customer needs advice, the customer care team will give advice on time, customer does not have to wait for support. This will show the respect, professionalism and reputation of the company in the heart of customers, helping to increase interaction with customers.

- Improved competitiveness with competitors**

The competition in the technology sales market is getting fiercer. If businesses know how to take advantage of an own e-commerce websites with a unique and easy-to-see interface, it will help to earn money and to attract more customers.

- Brand promotion**

In the digital age, customers can use social media, so it will be very convenient to promote the brand in the international market. Therefore, the design of an e-commerce websites will help businesses easily reach potential customers more quickly and effectively (The Future Of Customer Engagement and Experience 2021).

1.2.3 Drawbacks of E-commerce Website

Sometimes E-commerce cannot create a relationship between brands and buyers like physical stores. This lack of communication can be a disadvantage for many types of services and products, such as interior design or jewellery business. Also, from the security point of view, there have been many attacks of security breach in the world. Customer or credit information and identities can be stolen, . (The-Future-Of Customer Engagement and Experience 2023).

2. System Architecture

4.1 Back-end Side

The Three Tier Architecture model is very popular in Spring Boot. The presentation layer is the communication layer with the end user. All data sent from the server to the client parallelly here only checks the correctness of the data. In REST, three tier has the HTTP Method like GET, POST, PUT, DELETE with the purpose of getting data, adding data, updating data, deleting. The Controller contains all the logic of the program. Moreover, the Data access layer interacts with the database, returning the results to the business logic layer (Controller). Specifically, the project is divided into 3 floors (tier or layer) as shown in Figure 5.

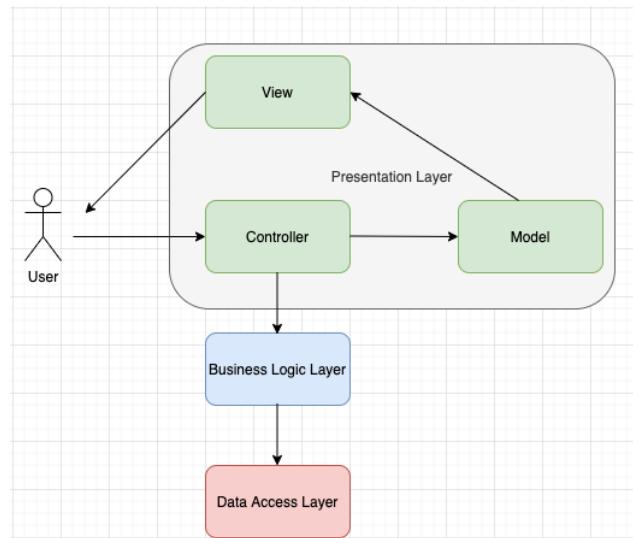


Figure 5. Three Tier Architecture

4.2 Front-end Side

React is a JavaScript library. React has no constraints in the project directory layout and structure. React gives the freedom to refer to different methods and to apply appropriate. Figure 6 illustrates the structure of ReactJS used in the report.

4.3 Data Flow Model of the Web Page

The data flow model of the web site is shown in Figure 11. First the user will go to View to view and interact on the page. When the user starts to load data, for example, by clicking the Reload button, a request from View is sent to the Controller. The Controller receives the request and begins to ask service (in the code is called the method of Service). The Service receives the request from the Controller, for a simple code that can be calculated and returned. But for operations that need to touch the database, the Service must call the Repository to get data in the database. The Repository receives a request from the Service, will manipulate DB. The data retrieved from the database is mapped by the Object Relational Mapping system (like JPA or Hibernate) into objects (in Java). These objects are called Entities. The Service receives Entities returned by the Repository. The transformation here is able to perform calculations, add or remove fields, and finally return Entity via the Model. The Model will be returned to the Controller. The Controller receives the Model and returns to the View. There are two ways, one is to use the template engine to pass Model data into the HTML page, and then return the HTML department (already with the data) to the client.

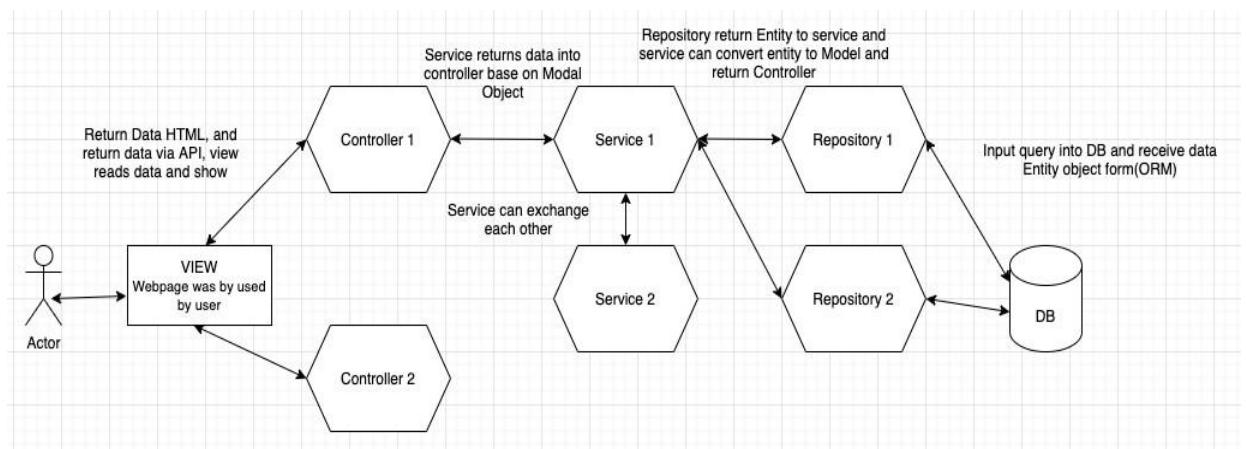


Figure 7. Data flow model.

3. Technologies Used

3.1 Front-end Side:

The front end (also known as the client-side) is all about what a user sees when visit a website, including design and languages like HTML or CSS.

3.1.1 HTML5

HTML (Hypertext Markup Language) is a platform similar to Microsoft Word that helps users to design websites elements, structure pages, categories or design applications. The main function of HTML platform is to create websites layout and format

3.1.2 Creating Responsive User Interface with Bootstrap

Bootstrap is the front-end framework, which is a free collection of tools for creating websites and web applications. Bootstrap includes HTML and CSS based design templates for typography, forms, buttons, and other interface elements, as well as extended JavaScript options (W3Schools 2021).

Bootstrap defines CSS classes available to help websites designers to save a lot of time. Bootstrap libraries have code ready to apply to websites without having to spend too much time writing it ourselves. With Bootstrap, developing websites interfaces to fit multiple devices becomes easier than ever. Bootstrap is the trend of developing websites interface which is very popular today. The websites is self-scalable to be compatible with all devices, from mobile phones to tablets, laptops, desktops. Another aspect is that responsive web design makes the websites a great experience for users across different screen sizes and devices (W3Schools 2021).

Responsive web design is very important since the devices and screen sizes that will be used to access websites cannot be predicted. A page that can perform well regardless of variation will provide a better and more consistent user experience than a page designed for a particular type of device and screen size (W3Schools 2021).

3.1.3 JavaScript/ES6

JavaScript is a programming language with the ability to bring life to websites design. The JavaScript scripting language is based on a built-in development object, or self-defined. JavaScript is the convenient and efficient because it is a dynamic programming language. Besides, JavaScript works based on HTML and CSS to create dynamic content on the website. Moreover, JavaScript is mainly used in client-side programming without installing environment settings whereas with Java or Python environment settings would have to be installed. Therefore, JavaScript programming language is trusted more and more and widely applied on effective websites (Developer Mozilla 2021).

The use of the JavaScript language can be applied to all different browsers, now commonly used as Chrome, or Firefox. Moreover, JavaScript is also an effective programming language, fully supported on mobile device browsers. Therefore, the use is diversified and can well meet many different needs and requirements of users (Developer Mozilla 2021).

JavaScript is simple, easy to learn and use. With the syntax quite similar to English, the use of JavaScript becomes much easier and more accessible. Through the used DOM model, it provides many useful features, which are pre-written to better respond to different needs and requirements of the user. With many useful features that the JavaScript programming language brings, it becomes easier to be able to develop scripts to solve certain requirements or purposes (Developer Mozilla 2021).

3.1.4 ReactJS Framework

ReactJS is a library which contains a lot of open-source JavaScript and it is maintained by Facebook and a community of individual developers and companies. The purpose of ReactJS was to create compelling websites applications with high speed and efficiency with minimal coding. The key purpose of ReactJS is that every website, when using ReactJS, has to run smoothly, fast and is highly scalable and simple to implement.

In general, ReactJS allows functional developers to separate the user interface from a

complex way and turn it into simpler parts which means that rendering data is not only done at a server location but also at the client using ReactJS.

The basic components of React are called components. The React components are easier to write because using JSX, JavaScript's optional syntax extension. JSX allows HTML to be combined with JavaScript. JSX is a blend of JavaScript and HTML. ReactJS clarifies the whole process of writing websites structure. In addition, the extension also makes rendering more options easier. JSX may not be the most popular syntax extension, ReactJS can be effective for developing special components or high-volume applications.

3.2 Back-end side

Backend programming is the handling of all complex business, hidden behind a website, application, system to help the system operate smoothly. The backend usually consists of three parts: the server, the application, and the database. Any product has two places where the code works to make things run smoothly: the client side and the server side.

3.2.1 Spring Framework and Spring Boot

Spring is a framework that was created to help developers build systems and run applications on the JVM conveniently, simply and quickly. Spring Framework is open source developed and used widely in developing websites. Spring framework is a collection of many different small projects, such as Spring MVC (used to build web-based applications), Spring Data and Spring Boot.

Spring Boot is the fastest way to create a standalone REST service. Spring Boot simplifies configuration, in particular, Spring Boot configures all by itself by providing specific behaviors. Spring Boot simplifies deploying, packing application into a jar package and it can be easily integrated into web containers. Earlier the initialization of a Spring project was hard when dependencies needed to be declared in pom.xml file using XML or other complex annotations. With Spring Boot Spring application creation is quick and the configuration is also simpler.

The benefits of Spring Boot are:

- Easy to create Spring-based applications with Java or Groovy.
- Spring Boot does not write a huge of standard Code, Annotations, and XML configuration.
- With Spring Boot it is simple to communicate applications with Spring ecosystems, for example, Spring JDBC, Spring ORM, Spring Data, Spring Security and so forth
- Spring Boot gives Embedded HTTP like Tomcat to create and test webapplications rapidly and without any problem.
- Spring Boot gives a CLI (Command Line Interface) device to create and test Spring Boot (Java or Groovy) applications from order brief effectively and rapidly.
- Spring Boot gives a great deal of modules to create and test Spring Boot applications rapidly utilizing build instruments like Maven.
- Based on Annotations to make beans rather than XML.
- Tomcat can be installed in the JAR fabricate record and can be run any-place java environment (Spring Boot. 2021)

3.2.2 Project Management: Maven

The Spring Boot Maven Plugin provides Spring Boot support in Apache Maven. It allows you to package executable jar or war archives, run Spring Boot applications, generate build information and start your Spring Boot application prior to running integration tests.

3.2.3 MySQL

MySQL is an open-source database management system (referred to as RDBMS) that operates in a client-server model. RDBMS stands for Relational Database Management System. MySQL is combined with Apache or Php MySQL. MySQL manages data through databases. Each database can have many relational tables containing data.

MySQL also has the same access and code as the

SQL language. How MySQL running.

- MySQL creates tables to store data, and also defines the relationships between those tables.
- Client sends SQL requests with a special command on MySQL.
- The application on the server receives and responds to the information and returns the results to the client .



Figure 10. MySQL Flowing.

Benefits of MySQL:

- Ease of Use: MySQL is simple, easy to use. In addition, this software can operate on many operating systems to provide many powerful utility functions.
- High security: MySQL has a lot of security features, including high-level security types.
- Multi-function: MySQL provides many features that any relational database management system should expect.
- Powerful and easily scalable: MySQL is capable of handling large amounts of data. Besides, users can expand it if the need arises.
- Fast: MySQL is faster than other software thanks to built-in

standards.

- Data recovery possible: MySQL allows users to recover data from the effects of crashes.

Drawbacks of MySQL:

- Restricted: MySQL is limited to a few features that applications may need
- Reliability is not too high: Compared with other relational database management systems, the reliability of MySQL is not too high.
- Limited capacity: The more records in MySQL, the more difficult it becomes to retrieve data due to capacity limitation.

4. Front-end Development

4.1 Project Setup:

- Install Visual studio code, Node.js, npm and create a new React application.
- Setup a React Project:
 1. Install Create React App:
 - In Order to install our React App we need to open terminal and run the following command to install Create React App globally:
`npm install -g create-react-app`
 2. Create a New React App:
 - Navigate to the directory where you want to create your React app and run:
`npx create-react-app my-react-app`
 - Replace "my-react-app" with our desired project name.
 3. Navigate to the Project Directory or Open in Visual Studio code:
To run project in cmd we have to change directory: `cd my-react-app`:
 4. Start the Development Server:
 - Run the following command to start the development server in cmd or in the terminal in vs code:
`npm start`
 5. Explore the Project Structure:
Create React App sets up a project with a predefined folder structure.
Key directories include:
 - src:** Contains your React components, styles, and other assets.
 - public:** Static assets and the HTML file where your React app is mounted.
 - node_modules:** Dependencies installed by npm.

6. Understanding the src Directory:

- **index.js**: Entry point for your React app.
- **App.js**: Main component that gets rendered in index.js.
- **index.css**: Global styles for your app.
- **components**: Folder for your React components.
- **App.test.js**: Test file for App component.
- **serviceWorker.js**: Service worker for offline capabilities.

Directory Structure:

Example:

```
/src
  /components
  /styles
  /assets
  /pages
  /utils
  /services
  App.js
  index.js
```

Figure:11

In project:

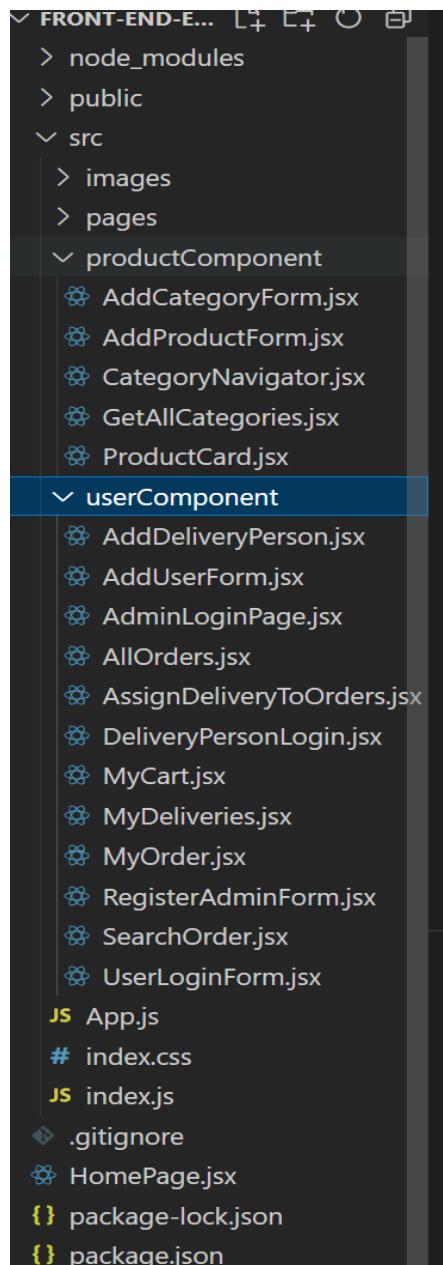


Figure:12

7. Install Additional Packages:

- Based on your project requirements, you may need to install additional packages.
- Here we used Bootstrap for styling, you can install it using:

```
npm i bootstrap@5.3.2
```

4.2 Implementation

Public/index.html:

This file serves as the entry point for the application and is typically located in the public folder. The **index.html** file is a static HTML file that includes a div element with a specific ID, often "root," where the React application will be mounted.

```
public > index.html > html > head
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8" />
5      <link rel="icon" href="%PUBLIC_URL%/Shophub.ico" />
6      <meta name="viewport" content="width=device-width, initial-scale=1" />
7      <meta name="theme-color" content="#000000" />
8      <meta
9        | name="description"
10       | content="Web site created using create-react-app"
11     />
12     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13
14     <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
15
16
17     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css" rel="stylesheet"
18       integrity="sha384-gH2yIjqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8159U5AR6csBvApHHN1/vI1Bx" crossorigin="anonymous">
19
20     <title>Online Shopping</title>
21   </head>
22   <body>
23     <noscript>You need to enable JavaScript to run this app.</noscript>
24     <div id="root"></div>
25
26
27     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/js/bootstrap.bundle.min.js"
28       integrity="sha384-A3rJD856KowSb7dwLzdYEk039Gagi7vIsF0jrRAoQmDKktQBHUuLZ9AsSv4jD4Xa" crossorigin="anonymous"></script>
29   </body>
30 </html>
```

Figure:13

src/pages

- **About Us**

In the website it provides information about the organization, company, or individuals behind the website. This Aboutus page is an opportunity to introduce the audience to the people, mission, values, and history of the entity.

```

src > pages > AboutUs.jsx > [o] AboutUs
  1
  2
  3 const AboutUs = () => {
  4   return (
  5     <div className="text-color ms-5 me-5 mr-5 mt-3">
  6       <b>
  7         Online shopping is a process whereby consumers directly buy goods, services etc. from a seller without an intermediary service over the Internet. Shoppers can visit web stores from the comfort of their house and shop as by sitting in front of the computer. Ecommerce, also known as electronic commerce or internet commerce, refers to the buying and selling of goods or services using the internet, and the transfer of money and data to execute these transactions.
  8       <br />
  9       <br />
 10      In existing system shopping can done in a manual way, the customer has to go for shopping, and then he is having the possibility to choose the products what ever he wants. Selling online also has its advantages when it comes to convincing customers you're the best in the industry. Your website can inform customers about your sales, the quality of your products, and why they should buy from you. You can also show customer reviews, so people know they're buying from a reputable brand. Doing business electronically describes e-commerce. E-commerce (EC), an abbreviation for electronic commerce, is the buying and selling of goods and services, or the transmitting of funds or data, over an electronic network, primarily the internet.
 11      <br />
 12      <br />
 13      The online shopping system is fast gaining media for to sale or purchase items from anywhere and anytime. It is basically based on Internet. It is related with B2C (Business to Customer) model and status of the design and development of e-commerce platform. E-business or Online business trans business transactions that take place online with the help of the internet. The term e-business came into existence in the year 1996. E-business is an abbreviation for electronic business. Therefore, the buyer and the seller do not meet personally. E-commerce is directly link to your business promotions, as it is the age of digital media. Making your business available online is crucial to your business development such as, highly convenience, wide exposure, global customer, easy to run, etc.
 14
 15
 16
 17     </b>
 18   </div>
 19 );
 20 }
 21
 22 export default AboutUs;
 23
 24
 25

```

Figure:14

- **AddCardDetails**

An e-commerce website or any application that involves processing payments.

```

src > pages > AddCardDetails.jsx > [o] AddCardDetails
  1 import { useState } from "react";
  2 import { useLocation } from "react-router-dom";
  3 import { useNavigate } from "react-router-dom";
  4 import { ToastContainer, toast } from "react-toastify";
  5
  6 const AddCardDetails = () => [
  7   const location = useLocation();
  8   const navigate = useNavigate();
  9   const user = JSON.parse(sessionStorage.getItem("active-user"));
 10   const priceToPay = location.state.priceToPay;
 11
 12   const [card, setCard] = useState({
 13     cardName: "",
 14     cardNumber: "",
 15     validThrough: "",
 16     cvv: "",
 17   });
 18
 19   const handleCardInput = (e) => {
 20     setCard({ ...card, [e.target.name]: e.target.value });
 21   };
 22
 23   const payAndOrder = () => {
 24     fetch("http://localhost:8080/api/user/order?userId=" + user.id, {
 25       method: "POST",
 26       headers: {
 27         Accept: "application/json",
 28         "Content-Type": "application/json",
 29       },
 30     }).then((result) => {
 31       console.log("result", result);
 32       result.json().then((res) => {
 33         console.log(res);
 34
 35         });
 36     });
 37   };
 38 }
 39

```

```

const payForOrder = () => {
  payAndOrder();
  toast.success("Products Ordered Sucessfully!!!", {
    position: "top-center",
    autoClose: 1000,
    hideProgressBar: false,
    closeOnClick: true,
    pauseOnHover: true,
    draggable: true,
    progress: undefined,
  });
  navigate("/home");
};

return (
  <div>
    <div class="mt-2 d-flex align-items-center justify-content-center">
      <div class="card form-card border-color" style={{ width: "25rem" }}>
        <div className="card-header bg-color custom-bg-text">
          <h5 className="card-title text-center">Payment Details</h5>
        </div>
        <div class="card-body text-color custom-bg">
          <form onSubmit={payForOrder}>
            <div class="mb-3">
              <label htmlFor="name" className="form-label">
                <b> Name on Card</b>
              </label>
              <input
                type="text"
                className="form-control"
                id="name"
                name="cardName"
                onChange={handleCardInput}
                value={card.cardName}
                required
              />
            </div>

```

```

77          <div class="mb-3">
78            <label htmlFor="cardNumber" className="form-label">
79              <b> Card Number</b>
80            </label>
81            <input
82              type="text"
83              className="form-control"
84              id="cardNumber"
85              name="cardNumber"
86              onChange={handleCardInput}
87              value={card.cardNumber}
88              required
89            />
90          </div>
91
92          <div class="mb-3">
93            <label htmlFor="validThrough" className="form-label">
94              <b>Valid Through</b>
95            </label>
96            <input
97              type="text"
98              className="form-control"
99              id="validThrough"
100             name="validThrough"
101             onChange={handleCardInput}
102             value={card.validThrough}
103             required
104           />
105         </div>
106

```

```

106
107         <div class="mb-3">
108             <label for="cvv" class="form-label">
109                 <b>CVV</b>
110             </label>
111             <input
112                 type="text"
113                 class="form-control"
114                 id="cvv"
115                 name="cvv"
116                 onChange={handleCardInput}
117                 value={card.cvv}
118                 required
119             />
120         </div>
121
122         <input
123             type="submit"
124             class="btn custom-bg-text bg-color"
125             value={"Pay Rs" + priceToPay}
126         />
127
128         <ToastContainer />
129     </form>
130 </div>
131 </div>
132 </div>
133 </div>
134 );
135 ];
136
137 export default AddCardDetails;
138

```

Figure:15

- **AdminHeader**

To Display different for the admin(seller)

```

src > pages > AdminHeader.jsx > ...
1 import { Link, useNavigate } from "react-router-dom";
2 import { ToastContainer, toast } from "react-toastify";
3 import "react-toastify/dist/ReactToastify.css";
4
5 const AdminHeader = () => {
6     let navigate = useNavigate();
7
8     const user = JSON.parse(sessionStorage.getItem("active-admin"));
9     console.log(user);
10
11     const adminLogout = () => {
12         to (property) CommonOptions.hideProgressBar?: boolean | undefined
13             Hide or show the progress bar. Default: false
14             hideProgressBar: false,
15             closeOnClick: true,
16             pauseOnHover: true,
17             draggable: true,
18             progress: undefined,
19         );
20         sessionStorage.removeItem("active-admin");
21         window.location.reload(true);
22     };
23
24
25     return (
26         <ul class="navbar-nav ms-auto mb-2 mb-lg-0 me-5">
27             <li class="nav-item">
28                 <Link to="/addcategory" class="nav-link active" aria-current="page">
29                     <b className="text-color"> Add Category</b>
30                 </Link>
31             </li>
32
33             <li class="nav-item">
34                 <Link to="/addproduct" class="nav-link active" aria-current="page">
35                     <b className="text-color">Add Product</b>
36                 </Link>
37             </li>
38

```

```

38
39         <li class="nav-item">
40             <Link
41                 to="/user/admin/allorder"
42                 class="nav-link active"
43                 aria-current="page"
44             >
45                 <b className="text-color">All Orders</b>
46             </Link>
47         </li>
48
49         <li class="nav-item">
50             <Link
51                 to="/user/admin/assigndelivery"
52                 class="nav-link active"
53                 aria-current="page"
54             >
55                 <b className="text-color">Assign Order Delivery</b>
56             </Link>
57         </li>
58
59         <li class="nav-item">
60             <Link
61                 to=""
62                 class="nav-link active"
63                 aria-current="page"
64                 onClick={adminLogout}
65             >
66                 <b className="text-color">Logout</b>
67             </Link>
68             <ToastContainer />
69         </li>
70     </ul>
71 );
72 }
73
74 export default AdminHeader;
75

```

Figure:16

- **Carousel**

It is used to showcase a set of images or content in a rotating manner. It's commonly employed on websites and applications to display multiple items in a limited space, allowing users to navigate through them manually or automatically. Here's a basic example of how you might implement a simple image carousel in a React application using a popular library called "react-slick."

```

src > pages > Carousel.jsx > ...
1  import carousell1 from "../images/carousel_1.jpeg";
2
3  const Carousel = () => {
4      return (
5          <div
6              id="carouselExampleCaptions"
7              class="carousel slide"
8              data-bs-ride="false"
9          >
10             <div class="carousel-indicators">
11                 <button
12                     type="button"
13                     data-bs-target="#carouselExampleCaptions"
14                     data-bs-slide-to="0"
15                     class="active"
16                     aria-current="true"
17                     aria-label="Slide 1"
18                 ></button>
19                 <button
20                     type="button"
21                     data-bs-target="#carouselExampleCaptions"
22                     data-bs-slide-to="1"
23                     aria-label="Slide 2"
24                 ></button>
25                 <button
26                     type="button"
27                     data-bs-target="#carouselExampleCaptions"
28                     data-bs-slide-to="2"
29                     aria-label="Slide 3"
30                 ></button>
31             </div>
32             <div class="carousel-inner">
33                 <div class="carousel-item active">
34                     <img src={carousell1} class="d-block w-100" alt="..." />
35                 </div>
36                 <div class="carousel-item">
37                     <img src={carousell1} class="d-block w-100" alt="..." />
38                 </div>

```

```

39   |   <div class="carousel-item">
40   |   |   <img src={carousel1} class="d-block w-100" alt="..." />
41   |   </div>
42   <button
43   |   class="carousel-control-prev"
44   |   type="button"
45   |   data-bs-target="#carouselExampleCaptions"
46   |   data-bs-slide="prev"
47   >
48   |   <span class="carousel-control-prev-icon" aria-hidden="true"></span>
49   |   <span class="visually-hidden">Previous</span>
50   </button>
51   <button
52   |   class="carousel-control-next"
53   |   type="button"
54   |   data-bs-target="#carouselExampleCaptions"
55   |   data-bs-slide="next"
56   >
57   |   <span class="carousel-control-next-icon" aria-hidden="true"></span>
58   |   <span class="visually-hidden">Next</span>
59   </button>
60   </div>
61   );
62   };
63   export default Carousel;
64
65
66

```

Figure:17

- **ContactUs**

Creating a "Contact Us" section in a web application involves providing users with a way to get in touch with the organization or individuals behind the website.

```

src > pages > ContactUs.jsx > ContactUs
1
2  const ContactUs = () => {
3    return [
4      <div className="text-color ms-5 me-5 mr-5 mt-3">
5        <b>
6          Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime
7          mollitia, molestiae quas vel sint commodi repudiandae consequuntur
8          voluptatum laborum numquam blanditiis harum quisquam eius sed odit
9          fugiat iusto fuga praesentium optio, eaque rerum! Provident similique
10         accusantium nemo autem. Veritatis obcaecati tenetur iure eius earum ut
11         molestias architecto voluptate aliquam nihil, eveniet aliquid culpa
12         officia aut! Impedit sit sunt quaerat, odit, tenetur error, harum
13         nesciunt ipsum debitis quas aliquid. Reprehenderit, quia. Quo neque
14
15        <br />
16        <br />
17
18      </b>
19    </div>
20  ];
21
22
23  export default ContactUs;
24

```

Figure:18

- **Delivery Person header**

Separate Interface for the delivery person

```

src > pages > DeliveryPersonHeader.jsx > ...
  1  import { Link, useNavigate } from "react-router-dom";
  2  import { ToastContainer, toast } from "react-toastify";
  3  import "react-toastify/dist/ReactToastify.css";
  4
  5  const DeliveryPersonHeader = () => {
  6    let navigate = useNavigate();
  7
  8    const user = JSON.parse(sessionStorage.getItem("active-delivery"));
  9    console.log(user);
 10
 11    const userLogout = () => {
 12      toast.success("logged out!!!", {
 13        position: "top-center",
 14        autoClose: 1000,
 15        hideProgressBar: false,
 16        closeOnClick: true,
 17        pauseOnHover: true,
 18        draggable: true,
 19        progress: undefined,
 20      });
 21      sessionStorage.removeItem("active-delivery");
 22      window.location.reload(true);
 23    };
 24
 25    return (
 26      <ul class="navbar-nav ms-auto mb-2 mb-lg-0 me-5 text-color">
 27        <li class="nav-item">
 28          <Link
 29            to="/user/delivery/myorders"
 30            class="nav-link active"
 31            aria-current="page"
 32            >
 33              <b className="text-color">My Deliveries</b>
 34            </Link>
 35          </li>
 36
 37          <li class="nav-item">
 38            <Link
 39              to="/user/admin/searchOrder"
 40              class="nav-link active"
 41              aria-current="page"
 42              >
 43                <b className="text-color">Update Order Delivery</b>
 44              </Link>
 45            </li>
 46
 47            <li class="nav-item">
 48              <Link
 49                to=""
 50                class="nav-link active"
 51                aria-current="page"
 52                onClick={userLogout}
 53                >
 54                  <b className="text-color">Logout</b>
 55                </Link>
 56                <ToastContainer />
 57              </li>
 58            </ul>
 59        );
 60    };
 61    export default DeliveryPersonHeader;
 62
 63

```

Figure:19

- **Header**

Involves designing the top section of the page that typically contains navigation links, a logo, and possibly other important information.

```

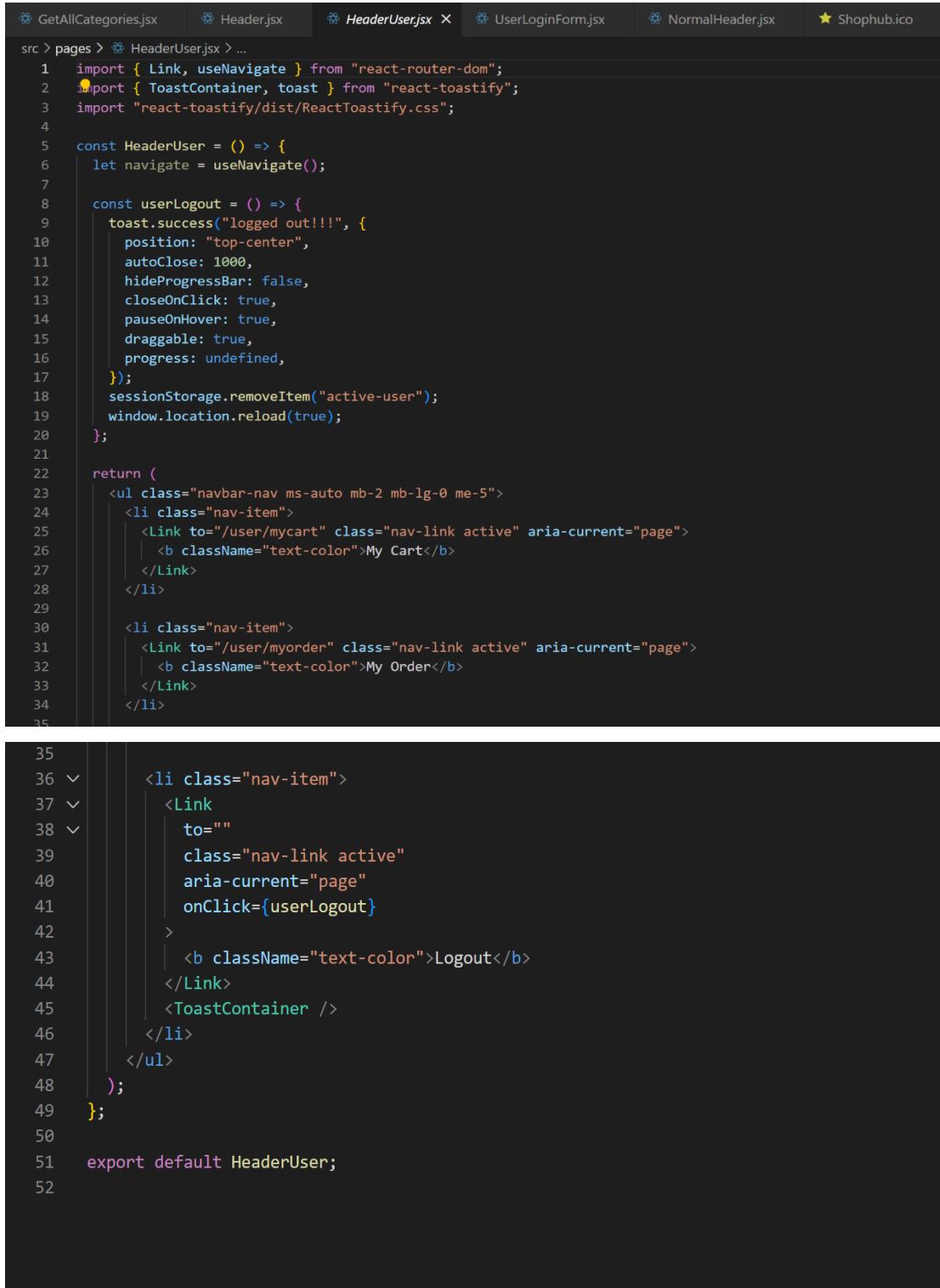
src > pages > Header.jsx > [o] Header
● 1  ✓ import { Link } from "react-router-dom";
  2   import RoleNav from "./RoleNav";
  3   import logo from "../images/e_logo.png";
  4
  5  ✓ const Header = () => {
  6    return (
  7      <div>
  8        <nav class="navbar navbar-expand-lg custom-bg text-color">
  9          <div class="container-fluid text-color">
10            <img
11              src={logo}
12              width="50"
13              height="40"
14              class="d-inline-block align-top"
15              alt=""
16            />
17            <Link to="/" class="navbar-brand">
18              <i>
19                <b class="text-color">Shophub</b>
20                <h6 class="text-color">Explore <span class="text-yellow-300">the every need </span> </h6>
21              </i>
22            </Link>
23
24            <button
25              class="navbar-toggler"
26              type="button"
27              data-bs-toggle="collapse"
28              data-bs-target="#navbarSupportedContent"
29              aria-controls="navbarSupportedContent"
30              aria-expanded="false"
31              aria-label="Toggle navigation"
32            >
33              <span class="navbar-toggler-icon"></span>
34            </button>
35
36            <div class="collapse navbar-collapse" id="navbarSupportedContent">
37              <ul class="navbar-nav me-auto mb-2 mb-lg-0">
38                <li class="nav-item">
39                  <Link to="/about" class="nav-link active" aria-current="page">
40                    <b class="text-color">About Us</b>
41                  </Link>
42                </li>
43
44                <li class="nav-item">
45                  <Link to="/contact" class="nav-link active" aria-current="page">
46                    <b class="text-color">Contact Us</b>
47                  </Link>
48                </li>
49              </ul>
50
51              <RoleNav />
52            </div>
53          </div>
54        </nav>
55      </div>
56    );
57  };
58
59  export default Header;
60

```

Figure:20

- **HeaderUser**

A user-related section in the header of a web application, commonly referred to as a user menu or user profile section.



```
 1 import { Link, useNavigate } from "react-router-dom";
 2 import { ToastContainer, toast } from "react-toastify";
 3 import "react-toastify/dist/ReactToastify.css";
 4
 5 const HeaderUser = () => {
 6   let navigate = useNavigate();
 7
 8   const userLogout = () => {
 9     toast.success("Logged out!!!", {
10       position: "top-center",
11       autoClose: 1000,
12       hideProgressBar: false,
13       closeOnClick: true,
14       pauseOnHover: true,
15       draggable: true,
16       progress: undefined,
17     });
18     sessionStorage.removeItem("active-user");
19     window.location.reload(true);
20   };
21
22   return (
23     <ul class="navbar-nav ms-auto mb-2 mb-lg-0 me-5">
24       <li class="nav-item">
25         <Link to="/user/mycart" class="nav-link active" aria-current="page">
26           <b className="text-color">My Cart</b>
27         </Link>
28       </li>
29
30       <li class="nav-item">
31         <Link to="/user/myorder" class="nav-link active" aria-current="page">
32           <b className="text-color">My Order</b>
33         </Link>
34       </li>
35
36     <li class="nav-item">
37       <Link
38         to=""
39         class="nav-link active"
40         aria-current="page"
41         onClick={userLogout}
42       >
43         <b className="text-color">Logout</b>
44       </Link>
45       <ToastContainer />
46     </li>
47   </ul>
48 );
49
50
51 export default HeaderUser;
52
```

Figure:21

- **HomePage**

A homepage for an e-commerce website using React involves building reusable components and managing the state of the application and involves showcasing featured products, promotions, and providing easy navigation for users.



```

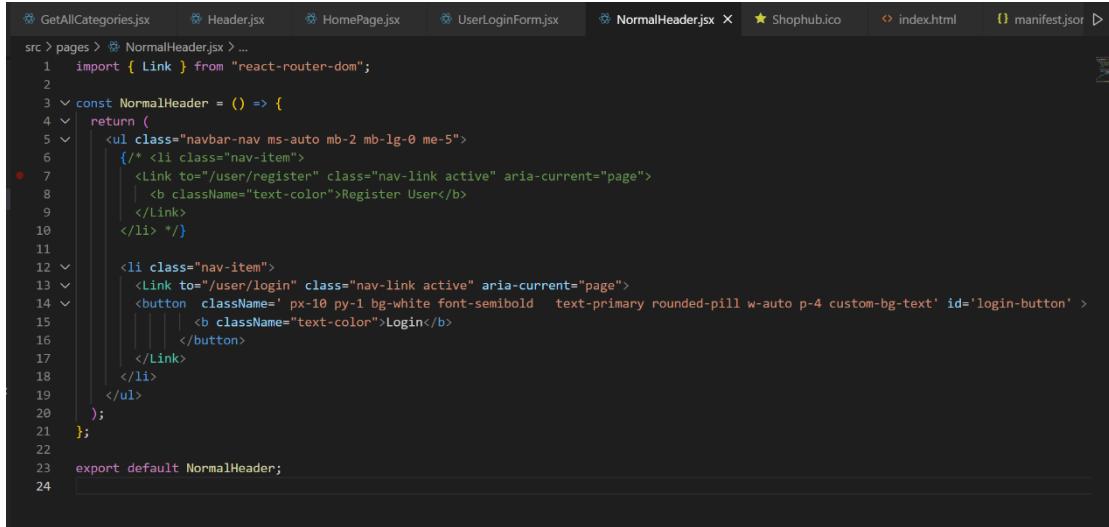
src > pages > HomePage.jsx > ...
1  import Carousel from "./Carousel";
2  import GetAllCategories from "../productComponent/GetAllCategories";
3  import ProductCard from "../productComponent/ProductCard";
4  import axios from "axios";
5  import { useState, useEffect } from "react";
6  import { useParams } from "react-router-dom";
7
8  const HomePage = () => {
9    const [products, setProducts] = useState([]);
10
11   const { categoryId } = useParams();
12
13   useEffect(() => {
14     const getAllProducts = async () => {
15       const allProducts = await retrieveAllProducts();
16       if (allProducts) {
17         setProducts(allProducts);
18       }
19     };
20
21     const getProductsByCategory = async () => {
22       const allProducts = await retrieveProductsByCategory();
23       if (allProducts) {
24         setProducts(allProducts);
25       }
26     };
27
28     if (categoryId == null) {
29       console.log("Category Id is null");
30       getAllProducts();
31     } else {
32       console.log("Category Id is NOT null");
33       getProductsByCategory();
34     }
35   }, [categoryId]);
36
37   const retrieveAllProducts = async () => {
38     const response = await axios.get("http://localhost:8080/api/product/all");
39
40     return response.data;
41   };
42
43   const retrieveProductsByCategory = async () => {
44     const response = await axios.get(
45       "http://localhost:8080/api/product/category?categoryId=" + categoryId
46     );
47
48     return response.data;
49   };
50
51   return (
52     <div className="container-fluid mb-2">
53       <Carousel />
54       <div className="mt-2 mb-5">
55         <div className="row">
56           <div className="col-md-2">
57             <GetAllCategories />
58           </div>
59           <div className="col-md-10">
60             <div className="row row-cols-1 row-cols-md-4 g-4">
61
62               <div>
63                 {products.map((product) => {
64                   return <ProductCard item={product} />;
65                 })}
66               </div>
67             </div>
68           </div>
69         </div>
70       </div>
71     );
72   };
73
74   export default HomePage;
75

```

Figure:22

- **NormalHeader**

the NormalHeader component is a simple header with a container and element displaying the website's name. You can customize it further based on your design and requirements For Example Login button in our website.



```

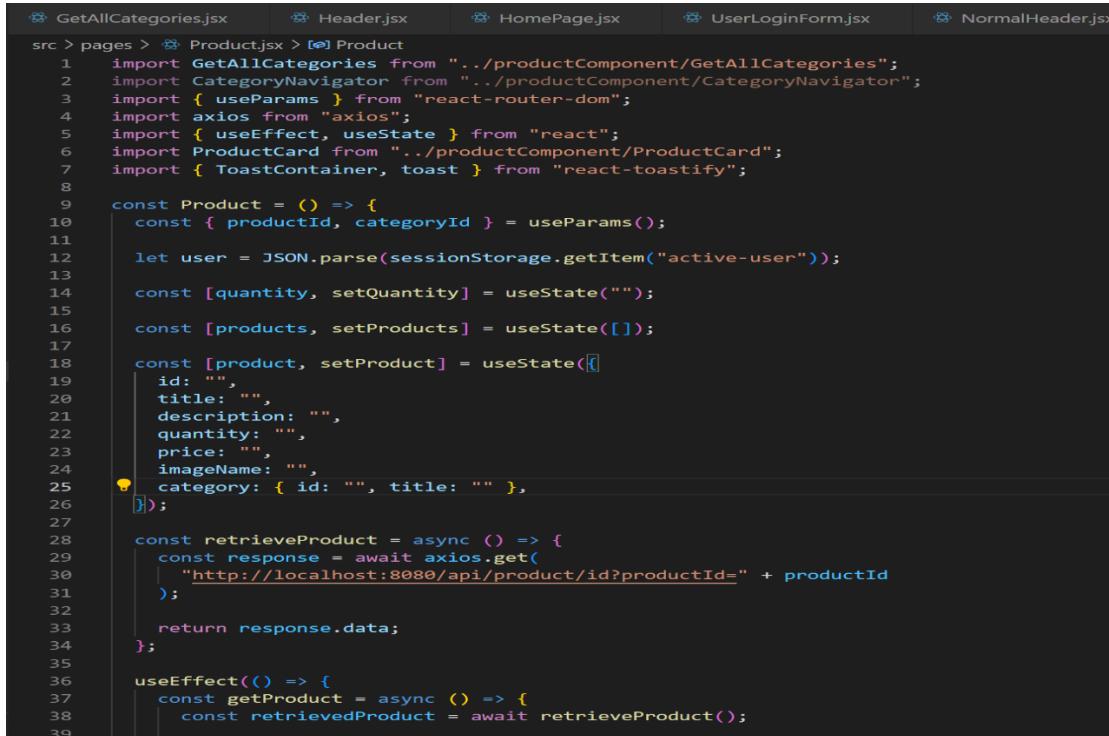
src > pages > NormalHeader.jsx
1 import { Link } from "react-router-dom";
2
3 const NormalHeader = () => {
4   return (
5     <ul class="navbar-nav ms-auto mb-2 mb-lg-0 me-5">
6       /* <li class="nav-item">
7         <Link to="/user/register" class="nav-link active" aria-current="page">
8           <b className="text-color">Register User</b>
9         </Link>
10        </li> */
11
12       <li class="nav-item">
13         <Link to="/user/login" class="nav-link active" aria-current="page">
14           <button className='px-10 py-1 bg-white font-semibold text-primary rounded-pill w-auto p-4 custom-bg-text' id='login-button'>
15             <b className="text-color">Login</b>
16           </button>
17         </Link>
18       </li>
19     </ul>
20   );
21 }
22
23 export default NormalHeader;
24

```

Figure:23

- **Product**

Product has props title, description, quantity, price,imageName and category renders a simple product card with an image, product name, price, and an "Add to Cart" button.



```

src > pages > Product.jsx
1 import GetAllCategories from "../productComponent/GetAllCategories";
2 import CategoryNavigator from "../productComponent/CategoryNavigator";
3 import { useParams } from "react-router-dom";
4 import axios from "axios";
5 import { useEffect, useState } from "react";
6 import ProductCard from "../productComponent/ProductCard";
7 import { ToastContainer, toast } from "react-toastify";
8
9 const Product = () => {
10   const { productId, categoryId } = useParams();
11
12   let user = JSON.parse(sessionStorage.getItem("active-user"));
13
14   const [quantity, setQuantity] = useState("");
15
16   const [products, setProducts] = useState([]);
17
18   const [product, setProduct] = useState([
19     id: "",
20     title: "",
21     description: "",
22     quantity: "",
23     price: "",
24     imageName: "",
25     category: { id: "", title: "" },
26   ]);
27
28   const retrieveProduct = async () => {
29     const response = await axios.get(
30       "http://localhost:8080/api/product/id?productId=" + productId
31     );
32
33     return response.data;
34   };
35
36   useEffect(() => {
37     const getProduct = async () => {
38       const retrievedProduct = await retrieveProduct();
39

```

```

39      setProduct(retrievedProduct);
40    };
41
42    const getProductsByCategory = async () => {
43      const allProducts = await retrieveProductsByCategory();
44      if (allProducts) {
45        setProducts(allProducts);
46      }
47    };
48  };
49
50  getProduct();
51  getProductsByCategory();
52 }, [productId]);
53
54 const retrieveProductsByCategory = async () => {
55   const response = await axios.get(
56     "http://localhost:8080/api/product/category?categoryId=" + categoryId
57   );
58   console.log(response.data);
59   return response.data;
60 };
61
62 const saveProductToCart = (userId) => {
63   fetch("http://localhost:8080/api/user/cart/add", {
64     method: "POST",
65     headers: {
66       Accept: "application/json",
67       "Content-Type": "application/json",
68     },
69     body: JSON.stringify({
70       quantity: quantity,
71       userId: userId,
72       productId: productId,
73     }),
74   }).then((result) => {
75     console.log("result", result);
76   });
77
78   toast.success("Products added to Cart Successfully!!!", {
79     position: "top-center",
80     autoClose: 1000,
81     hideProgressBar: false,
82     closeOnClick: true,
83     pauseOnHover: true,
84     draggable: true,
85     progress: undefined,
86   });
87   result.json().then((res) => {
88     console.log("response", res);
89   });
90 });
91 };
92
93 const addToCart = (e) => {
94   if (user == null) {
95     alert("Please login to buy the products!!!");
96     e.preventDefault();
97   } else {
98     saveProductToCart(user.id);
99     setQuantity("");
100    e.preventDefault();
101  }
102};

```

```

103
104  return (
105    <div className="container-fluid">
106      <div class="row">
107        <div class="col-sm-2 mt-2">
108          <GetAllCategories />
109        </div>
110        <div class="col-sm-3 mt-2 admin">
111          <div class="card form-card border-color custom-bg">
112            <img
113              src={"http://localhost:8080/api/product/" + product.imageName}
114              style={{
115                maxHeight: "500px",
116                maxWidth: "100%",
117                width: "auto",
118              }}
119              class="card-img-top rounded mx-auto d-block m-2"
120              alt="img"
121            />
122          </div>
123        </div>
124        <div class="col-sm-7 mt-2">
125          <div class="card form-card border-color custom-bg">
126            <div class="card-header bg-color">
127              <div className="d-flex justify-content-between">
128                <h1 className="custom-bg-text">{product.title}</h1>
129              </div>
130            </div>
131
132            <div class="card-body text-left text-color">
133              <div class="text-left mt-3">
134                <h3>Description :</h3>
135              </div>
136              <h4 class="card-text">{product.description}</h4>
137            </div>
138
139            <div class="card-footer custom-bg">
140              <div className="text-center text-color">
141                <p>
142                  <span>
143                    <h4>Price : &#8377;{product.price}</h4>
144                  </span>
145                </p>
146              </div>
147              <div className="d-flex justify-content-between">
148                <div>
149                  <form class="row g-3" onSubmit={addToCart}>
150                    <div class="col-auto">
151                      <input
152                        type="number"
153                        class="form-control"
154                        id="addToCart"
155                        placeholder="Enter Quantity..."
156                        onChange={(e) => setQuantity(e.target.value)}
157                        value={quantity}
158                        required
159                      />
160                    </div>
161                    <div class="col-auto">
162                      <input
163                        type="submit"
164                        className="btn bg-color custom-bg-text mb-3"
165                        value="Add to Cart"
166                      />
167                      <ToastContainer />
168                    </div>
169                  </form>
170                </div>
171

```

```

171
172           <p class="ml-2 text-color">
173             |   <b>Stock : {product.quantity}</b>
174           |   </p>
175           |   </div>
176         </div>
177       </div>
178     </div>
179   </div>
180
181   <div className="row mt-2">
182     <div className="col-sm-2"></div>
183
184     <div className="col-sm-10">
185       <h2>Related Products:</h2>
186       <div className="row row-cols-1 row-cols-md-4 g-4">
187
188         {products.map((product) => {
189           |   return <ProductCard item={product} />;
190         })
191       </div>
192     </div>
193   </div>
194 </div>
195 );
196 };
197
198 export default Product;
199

```

Figure:24

- **RoleNav**

This component can conditionally render navigation links based on the user's role. the RoleNav component takes a prop and conditionally renders different navigation links based on the user's role. Adjust the links and logic according to your specific use case.

src/product Component

- **AddCategoryForm**

the AddCategoryForm component is a controlled form component that uses the useState hook to manage the state of the form inputs. The prop is a function that will be called when the form is submitted, allowing the parent component to handle the addition of the category.

```

    GetAllCategories.jsx      Header.jsx       HomePage.jsx      UserLoginForm.jsx      NormalHeader.jsx
src > productComponent > AddCategoryForm.jsx > ...
  1 import { useState } from "react";
  2 
  3 const AddCategoryForm = () => {
  4   const [title, setTitle] = useState("");
  5   const [description, setDescription] = useState("");
  6 
  7   const saveCategory = () => {
  8     let data = { title, description };
  9 
 10   fetch("http://localhost:8080/api/category/add", {
 11     method: "POST",
 12     headers: {
 13       Accept: "application/json",
 14       "Content-Type": "application/json",
 15     },
 16     body: JSON.stringify(data),
 17   }).then((result) => {
 18     console.warn("result", result);
 19     result.json().then((res) => {
 20       console.log("response", res);
 21     });
 22   });
 23 };
 24 
 25 return (
 26   <div>
 27     <div class="mt-2 d-flex align-items-center justify-content-center">
 28       <div
 29         class="card form-card border-color custom-bg"
 30         style={{ width: "25rem" }}
 31       >
 32         <div className="card-header bg-color text-center custom-bg-text">
 33           <h5 class="card-title">Add Category</h5>
 34         </div>
 35         <div class="card-body text-color">
 36           <form>
 37             <div class="mb-3">
 38               <label for="title" class="form-label">
 39                 <b>Category Title</b>
 40               </label>
 41               <input
 42                 type="text"
 43                 class="form-control"
 44                 id="title"
 45                 placeholder="enter title.."
 46                 onChange={(e) => {
 47                   setTitle(e.target.value);
 48                 }}
 49                 value={title}
 50               />
 51             </div>
 52             <div class="mb-3">
 53               <label for="description" class="form-label">
 54                 <b>Category Description</b>
 55               </label>
 56               <textarea
 57                 class="form-control"
 58                 id="description"
 59                 rows="3"
 60                 placeholder="enter description.."
 61                 onChange={(e) => {
 62                   setDescription(e.target.value);
 63                 }}
 64                 value={description}
 65               />
 66             </div>
 67           </form>
 68         </div>
 69       </div>
 70     </div>
 71   </div>
 72 );

```

```

67
68           <button
69             type="submit"
70             onClick={saveCategory}
71             class="btn bg-color custom-bg-text"
72           >
73             Add Category
74           </button>
75         </form>
76       </div>
77     </div>
78   </div>
79 );
80 };
81 };
82
83 export default AddCategoryForm;
84

```

Figure:25

- **AddProductForm**

the AddProductForm component is a controlled form component that uses the useState hook to manage the state of the form inputs. This prop is a function that will be called when the form is submitted, allowing the parent component to handle the addition of the product.

```

src > productComponent > ✎ AddProductForm.jsx > ...
1 import { useState, useEffect } from "react";
2 import axios from "axios";
3
4 const AddProductForm = () => {
5   const [categories, setCategories] = useState([]);
6
7   const retrieveAllCategories = async () => {
8     const response = await axios.get("http://localhost:8080/api/category/all");
9     return response.data;
10   };
11
12   useEffect(() => {
13     const getAllCategories = async () => {
14       const allCategories = await retrieveAllCategories();
15       if (allCategories) {
16         setCategories(allCategories);
17       }
18     };
19
20     getAllCategories();
21   }, []);
22
23   const [selectedPhoto, setSelectedPhoto] = useState(null);
24   const [product, setProduct] = useState({
25     title: "",
26     description: "",
27     price: "",
28     quantity: "",
29     categoryId: ""
30   });
31
32   const handleInput = (e) => {
33     setProduct({ ...product, [e.target.name]: e.target.value });
34   };
35

```

```

35
36  const saveProduct = () => {
37    const formData = new FormData();
38    formData.append("image", selectedPhoto);
39    formData.append("title", product.title);
40    formData.append("description", product.description);
41    formData.append("price", product.price);
42    formData.append("quantity", product.quantity);
43    formData.append("categoryId", product.categoryId);
44
45    axios
46      .post("http://localhost:8080/api/product/add", formData)
47      .then((resp) => {
48        let result = resp.data.data;
49        alert("Product saved successfully");
50      })
51      .catch((error) => {
52        console.log("Error", error);
53        alert("Error saving product");
54      });
55  };
56
57  return (
58    <div>
59      <div class="mt-2 d-flex align-items-center justify-content-center">
60        <div
61          class="card form-card border-color custom-bg"
62          style={{ width: "25rem" }}
63        >
64          <div className="card-header bg-color custom-bg-text text-center">
65            <h5 className="card-title">Add Product</h5>
66          </div>
67          <div className="card-body text-color">
68            <form>
69              <div className="mb-3">
70                <label htmlFor="title" className="form-label">
71                  <b>Product Title</b>
72                </label>
73                <input
74                  type="text"
75                  className="form-control"
76                  id="title"
77                  name="title"
78                  onChange={handleInput}
79                  value={product.title}
80                />
81              </div>
82              <div className="mb-3">
83                <label htmlFor="description" className="form-label">
84                  <b>Product Description</b>
85                </label>
86                <textarea
87                  className="form-control"
88                  id="description"
89                  name="description"
90                  rows="3"
91                  onChange={handleInput}
92                  value={product.description}
93                />
94              </div>
95
96              <div className="mb-3">
97                <label className="form-label">
98                  <b>Category</b>
99                </label>
100
101                <select
102                  name="categoryId"
103                  onChange={handleInput}
104                  className="form-control"
105                >

```

```

105
106         <option value="">Select Category</option>
107
108     {categories.map((category) => {
109         return (
110             <option value={category.id}> {category.title} </option>
111         );
112     })
113     </select>
114   </div>
115
116   <div class="mb-3 mt-1">
117     <label for="quantity" class="form-label">
118       <b>Product Quantity</b>
119     </label>
120     <input
121       type="number"
122       class="form-control"
123       id="quantity"
124       name="quantity"
125       onChange={handleInput}
126       value={product.quantity}
127     />
128   </div>
129
130   <div class="mb-3">
131     <label for="price" class="form-label">
132       <b>Product Price</b>
133     </label>

```

```

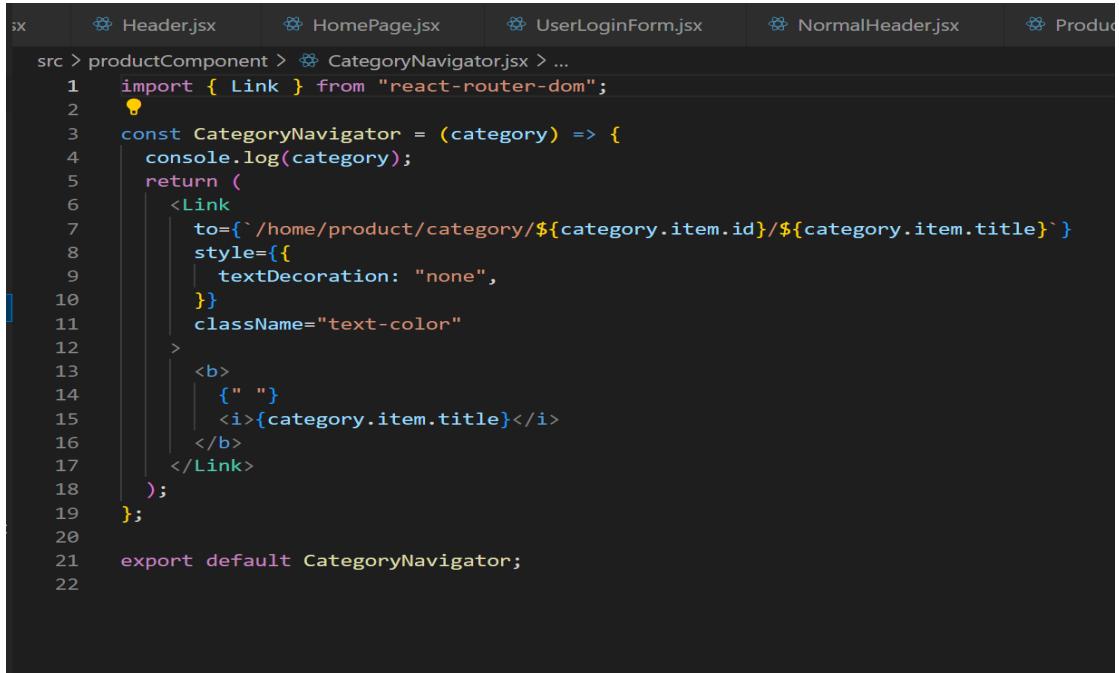
src > productComponent > ✘ AddProductForm.jsx > ...
135   <input
136     type="number"
137     class="form-control"
138     id="price"
139     name="price"
140     onChange={handleInput}
141     value={product.price}
142   />
143   </div>
144
145   <div class="mb-3">
146     <label for="formFile" class="form-label">
147       <b> Select Product Image</b>
148     </label>
149     <input
150       class="form-control"
151       type="file"
152       id="formFile"
153       name="photo"
154       value={product.photo}
155       onChange={(e) => setSelectedPhoto(e.target.files[0])}
156     />
157   </div>
158
159   <button
160     type="submit"
161     class="btn bg-color custom-bg-text"
162     onClick={saveProduct}
163   >
164     Add Product
165   </button>
166   </form>
167 </div>
168 </div>
169 </div>
170   );
171 };
172
173 export default AddProductForm;
174

```

Figure:26

- **Category Navigator**

This component can render a list of categories, and users can navigate between them.



```

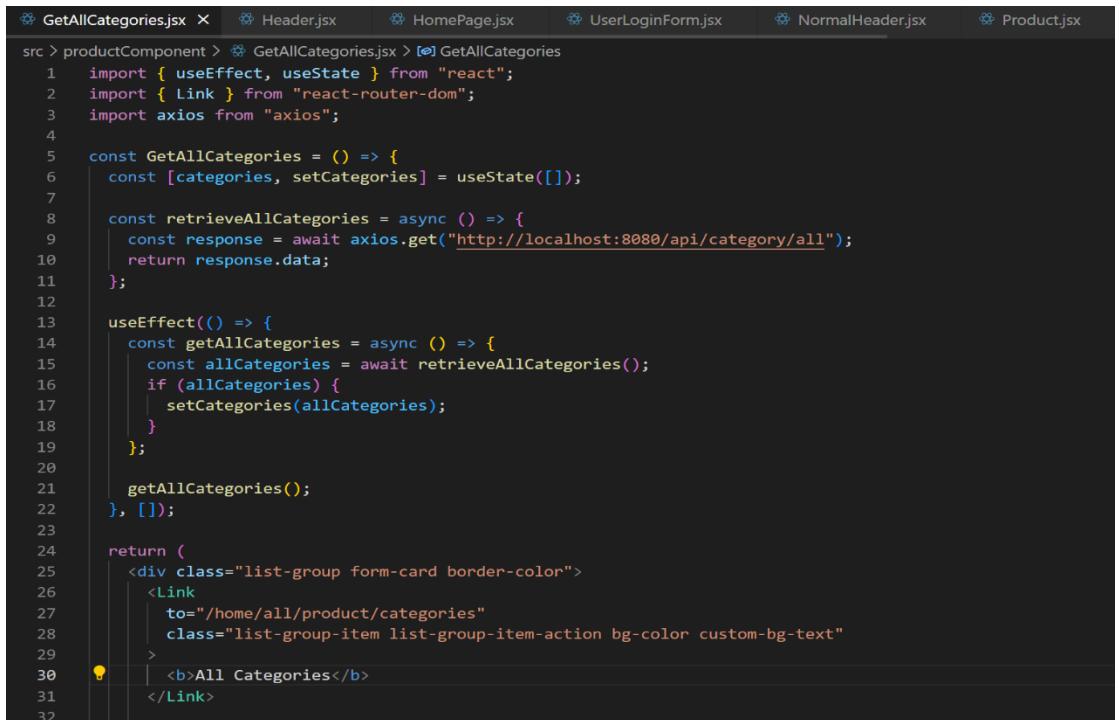
src > productComponent > CategoryNavigator.jsx > ...
1  import { Link } from "react-router-dom";
2
3  const CategoryNavigator = (category) => {
4    console.log(category);
5    return (
6      <Link
7        to={`/home/product/category/${category.item.id}/${category.item.title}`}
8        style={{
9          textDecoration: "none",
10         }}
11        className="text-color"
12      >
13        <b>
14          {" "}
15          <i>{category.item.title}</i>
16        </b>
17      </Link>
18    );
19  };
20
21  export default CategoryNavigator;
22

```

Figure:27

- **GetAllCategories**

To fetch all categories from an API or data source, you typically use asynchronous operations, such as making an HTTP request.



```

src > productComponent > GetAllCategories.jsx > GetAllCategories
1  import { useEffect, useState } from "react";
2  import { Link } from "react-router-dom";
3  import axios from "axios";
4
5  const GetAllCategories = () => {
6    const [categories, setCategories] = useState([]);
7
8    const retrieveAllCategories = async () => {
9      const response = await axios.get("http://localhost:8080/api/category/all");
10     return response.data;
11   };
12
13   useEffect(() => {
14     const getAllCategories = async () => {
15       const allCategories = await retrieveAllCategories();
16       if (allCategories) {
17         setCategories(allCategories);
18       }
19     };
20
21     getAllCategories();
22   }, []);
23
24   return (
25     <div class="list-group form-card border-color">
26       <Link
27         to="/home/all/product/categories"
28         class="list-group-item list-group-item-action bg-color custom-bg-text"
29       >
30         <b>All Categories</b>
31       </Link>
32     </div>

```

```

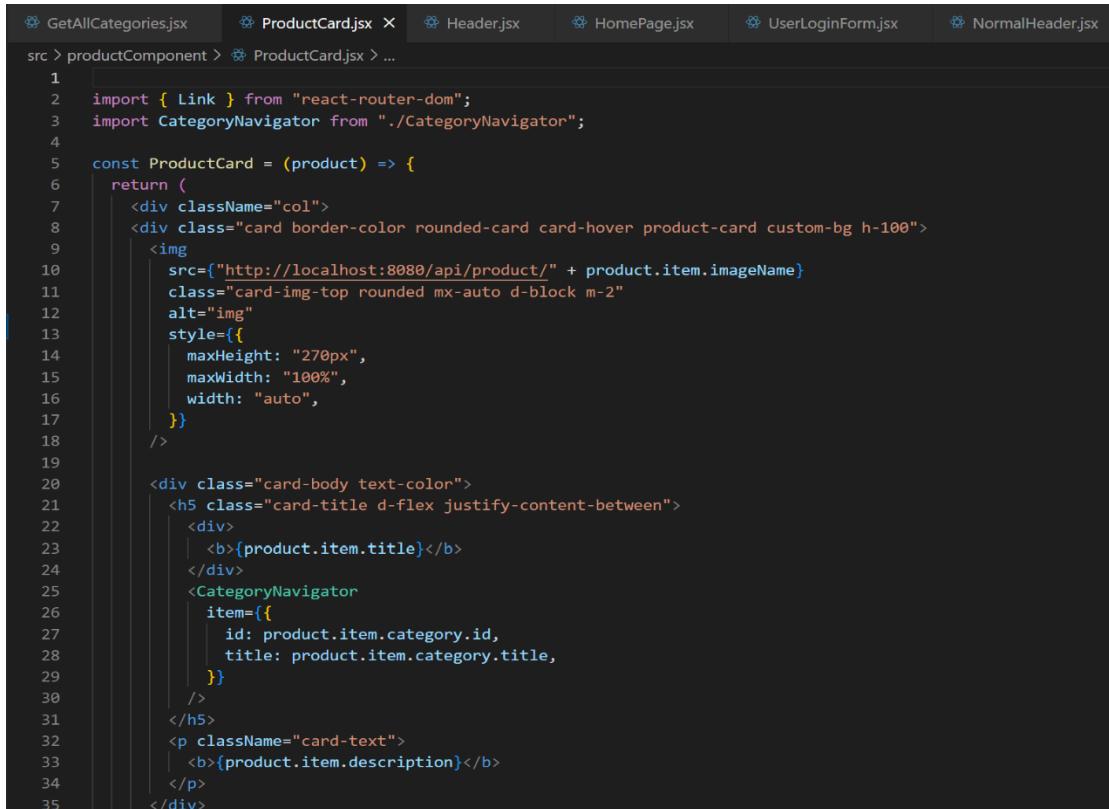
32
33     {categories.map((category) => {
34         return (
35             <Link
36                 to={`/home/product/category/${category.id}/${category.title}`}
37                 class="list-group-item list-group-item-action text-color custom-bg"
38             >
39                 <b>{category.title}</b>
40             </Link>
41         );
42     })
43     </div>
44   );
45 }
46
47 export default GetAllCategories;
48

```

Figure:28

- **Product Card**

Product Card component is a common element in e-commerce websites to display information about a product in a visually appealing way. The ProductCard component takes a product prop, which is an object containing information about a product (e.g., name, price, description, imageUrl). The component renders an image, product details (name, price, description), and an "Add to Cart" button.



```

GetAllCategories.jsx  ProductCard.jsx  Header.jsx  HomePage.jsx  UserLoginForm.jsx  NormalHeader.jsx
src > productComponent > ProductCard.jsx > ...
1
2 import { Link } from "react-router-dom";
3 import CategoryNavigator from "./CategoryNavigator";
4
5 const ProductCard = (product) => {
6   return (
7     <div className="col">
8       <div class="card border-color rounded-card card-hover product-card custom-bg h-100">
9         <img
10           src={"http://localhost:8080/api/product/" + product.item.imageName}
11           class="card-img-top rounded mx-auto d-block m-2"
12           alt="img"
13           style={({
14             maxHeight: "270px",
15             maxWidth: "100%",
16             width: "auto",
17           })}
18       />
19
20       <div className="card-body text-color">
21         <h5 className="card-title d-flex justify-content-between">
22           <div>
23             <b>{product.item.title}</b>
24           </div>
25           <CategoryNavigator
26             item={({
27               id: product.item.category.id,
28               title: product.item.category.title,
29             })}
30           />
31         </h5>
32         <p className="card-text">
33           <b>{product.item.description}</b>
34         </p>
35       </div>

```

```

35      </div>
36      <div class="card-footer">
37          <div className="text-center text-color">
38              <p>
39                  <span>
40                      | <h4>Price : &#8377;{product.item.price}</h4>
41                  </span>
42              </p>
43          </div>
44          <div className="d-flex justify-content-between">
45              <Link
46                  to={`/product/${product.item.id}/category/${product.item.category.id}`}
47                  className="btn bg-color custom-bg-text"
48              >
49                  Add to Cart
50              </Link>
51
52              <p class="text-color">
53                  <b>
54                      | <i>Stock :</i> {product.item.quantity}
55                  </b>
56              </p>
57          </div>
58      </div>
59  </div>
60</div>
61);
62};
63
64 export default ProductCard;
65

```

Figure:29

Src/userComponents

- **AdduseFrom**

The adduserfrom component is a controlled form component using the useState hook to manage the state of the form inputs.

```

src > userComponent > ✎ AddUserForm.jsx > ...
1  import { useState } from "react";
2  import "react-toastify/dist/ReactToastify.css";
3  import { ToastContainer, toast } from "react-toastify";
4
5  const AddUserForm = () => {
6      const [user, setUser] = useState({
7          firstName: "",
8          lastName: "",
9          emailId: "",
10         password: "",
11         phoneNo: "",
12         street: "",
13         city: "",
14         pincode: "",
15         role: ""
16     });
17
18     const handleUserInput = (e) => {
19         setUser({ ...user, [e.target.name]: e.target.value });
20     };
21
22     const saveUser = (event) => {
23         event.preventDefault();
24         fetch("http://localhost:8080/api/user/register", {
25             method: "POST",
26             headers: {
27                 Accept: "application/json",
28                 "Content-Type": "application/json",
29             },
30             body: JSON.stringify(user),
31         }).then((result) => {
32             console.log("*****near toast thing");
33             toast.success("Registered Successfully!!!", {
34                 position: "top-center",
35                 autoClose: 1000,
36                 hideProgressBar: false,
37                 closeOnClick: true,
38                 pauseOnHover: true,
39                 draggable: true,

```

```

40     progress: undefined,
41   });
42   console.warn("result", result);
43   result
44     .json()
45     .then((res) => {
46       console.log("response", res);
47     })
48     .catch((error) => {
49       console.log("*****", error);
50       console.log(error);
51     });
52   });
53 };
54
55 return (
56   <div>
57     <div class="mt-2 d-flex align-items-center justify-content-center ms-2 me-2 mb-2">
58       <div
59         class="card form-card border-color text-color custom-bg"
60         style={{ width: "25rem" }}
61       >
62         <div className="card-header bg-color custom-bg-text text-center">
63           <h5 className="card-title">Add User</h5>
64         </div>
65         <div className="card-body">
66           <form onSubmit={saveUser}>
67             <div className="mb-3 text-color">
68               <label htmlFor="role" className="form-label">
69                 <b>User Role</b>
70               </label>
71               <select
72                 onChange={handleUserInput}
73                 className="form-control"
74                 name="role"
75               >
76                 <option value="0">Select Role</option>
77                 <option value="Admin"> Admin </option>
78                 <option value="Customer"> Customer </option>
79                 <option value="Delivery"> Delivery Person </option>
80               </select>
81             </div>
82             <div className="mb-3 text-color">
83               <label htmlFor="title" className="form-label">
84                 <b> First Name</b>
85               </label>
86               <input
87                 type="text"
88                 className="form-control"
89                 id="firstName"
90                 name="firstName"
91                 onChange={handleUserInput}
92                 value={user.firstName}
93               />
94             </div>
95             <div className="mb-3 text-color">
96               <label htmlFor="description" className="form-label">
97                 <b>Last Name</b>
98               </label>
99               <input
100                 type="text"
101                 className="form-control"
102                 id="lastName"
103                 name="lastName"
104                 onChange={handleUserInput}
105                 value={user.lastName}
106               />
107             </div>
108           </form>
109         </div>

```

```

109
110          <div className="mb-3 text-color">
111              <b>
112                  | <label className="form-label">Email Id</label>
113              </b>
114              <input
115                  type="email"
116                  className="form-control"
117                  id="emailId"
118                  name="emailId"
119                  onChange={handleUserInput}
120                  value={user.emailId}
121              />
122          </div>
123
124          <div className="mb-3 mt-1">
125              <label htmlFor="quantity" className="form-label">
126                  | Password
127              </label>
128              <input
129                  type="password"
130                  className="form-control"
131                  id="password"
132                  name="password"
133                  onChange={handleUserInput}
134                  value={user.password}
135              />
136          </div>
137
138          <div className="mb-3">
139              <label htmlFor="price" className="form-label">
140                  | <b>Mobile No</b>
141              </label>
142              <input
143                  type="number"
144                  className="form-control"
145                  id="phoneNo"
146                  name="phoneNo"
147                      | onChange={handleUserInput}
148                      | value={user.phoneNo}
149                  />
150          </div>
151
152          <div className="mb-3">
153              <label htmlFor="description" className="form-label">
154                  | <b> Street</b>
155              </label>
156              <textarea
157                  className="form-control"
158                  id="street"
159                  name="street"
160                  rows="3"
161                  onChange={handleUserInput}
162                  value={user.street}
163              />
164          </div>
165
166          <div className="mb-3">
167              <label htmlFor="price" className="form-label">
168                  | <b>City</b>
169              </label>
170              <input
171                  type="text"
172                  className="form-control"
173                  id="city"
174                  name="city"
175                  onChange={handleUserInput}
176                  value={user.city}
177              />
178          </div>
179

```

```

179
180          <div class="mb-3">
181              <label for="pincode" class="form-label">
182                  <b>Pincode</b>
183              </label>
184              <input
185                  type="number"
186                  class="form-control"
187                  id="pincode"
188                  name="pincode"
189                  onChange={handleUserInput}
190                  value={user.pincode}
191              />
192          </div>
193
194          <input
195              type="submit"
196              class="btn bg-color custom-bg-text"
197              value="Register User"
198          />
199
200          <ToastContainer />
201      </form>
202  </div>
203 </div>
204 </div>
205 </div>
206 );
207 };
208
209 export default AddUserForm;
210

```

Figure:30

- **AdminLoginPage**

The AdminLoginPage component is a controlled form component using the useState hook to manage the state of the form inputs.

The component takes an onloginprop, which is a function that will be called when the form is submitted. This allows the parent component to handle the login process for administrators.

The form includes input fields for the username and password. It also includes basic validation to check if the fields are not empty.

When the form is submitted, the onlogin function is called with the login credentials, and the form fields are cleared.

```

  ⚡ GetAllCategories.jsx ✘ ⚡ ProductCard.jsx ✘ ⚡ AddUserForm.jsx ✘ ⚡ AdminLoginPage.jsx ✘ ⚡ Header.jsx
src > userComponent > ⚡ AdminLoginPage.jsx > ...
1  import { useState } from "react";
2  import { ToastContainer, toast } from "react-toastify";
3  import "react-toastify/dist/ReactToastify.css";
4  import { useNavigate } from "react-router-dom";
5
6  const AdminLoginPage = () => {
7    const [loginRequest, setLoginRequest] = useState({
8      emailId: "",
9      password: "",
10    });
11
12    const [response, setResponse] = useState();
13
14    const handleUserInput = (e) => {
15      setLoginRequest({ ...loginRequest, [e.target.name]: e.target.value });
16    };
17
18    const loginAction = (e) => {
19      fetch("http://localhost:8080/api/user/admin/login", {
20        method: "POST",
21        headers: {
22          Accept: "application/json",
23          "Content-Type": "application/json",
24        },
25        body: JSON.stringify(loginRequest),
26      }).then((result) => {
27        console.log("result", result);
28        result.json().then((res) => {
29          sessionStorage.setItem("active-admin", JSON.stringify(res));
30          toast.success("logged in successfully!!!", {
31            position: "top-center",
32            autoClose: 1000,
33            hideProgressBar: false,
34            closeOnClick: true,
35            pauseOnHover: true,
36            draggable: true,
37            progress: undefined,
38          });
39        });
40      });
41
42      e.preventDefault();
43    };
44
45    return (
46      <div>
47        <div className="mt-2 d-flex align-items-center justify-content-center">
48          <div className="card form-card border-color" style={{ width: "25rem" }}>
49            <div className="card-header bg-color text-center">
50              <h4 className="card-title">Admin Login</h4>
51            </div>
52            <div className="card-body">
53              <form>
54                <div className="mb-3">
55                  <label htmlFor="emailId" className="form-label custom-bg-text">
56                    Email Id
57                  </label>
58                  <input
59                    type="email"
60                    className="form-control"
61                    id="emailId"
62                    name="emailId"
63                    onChange={handleUserInput}
64                    value={loginRequest.emailId}
65                  />
66                </div>
67                <div className="mb-3">
68                  <label htmlFor="password" className="form-label custom-bg-text">
69                    Password
70                  </label>
71                  <input
72                    type="password"
73                    className="form-control"
74                    id="password"
75                    name="password"
76                    onChange={handleUserInput}
77                    value={loginRequest.password}
78                    autoComplete="on"
79                  />
80                </div>
81              </form>
82            </div>
83          </div>
84        </div>
85      </div>
86    );
87  }
88
```

```

78           autoComplete="on"
79         />
80       </div>
81
82     <button
83       type="submit"
84       className="btn custom-bg text-color"
85       onClick={loginAction}
86     >
87       Login
88     </button>
89     <ToastContainer />
90   </form>
91 </div>
92 </div>
93 </div>
94 </div>
95 );
96 };
97
98 export default AdminLoginPage;
99

```

Figure:31

- **AllOrders**

The AllOrders component uses the useState and useEffect hooks to manage the state of the orders and fetch orders when the component mounts.

```

src > userComponent > ✎ AllOrders.jsx > ...
1 import { useState, useEffect } from "react";
2 import axios from "axios";
3 import React from "react";
4
5 const AllOrders = () => {
6   const [allOrderData, setAllOrderData] = useState([]);
7
8   useEffect(() => {
9     const getAllOrder = async () => {
10       const allOrder = await retrieveAllOrder();
11       if (allOrder) {
12         setAllOrderData(allOrder);
13       }
14     };
15
16     getAllOrder();
17   }, [ ]);
18
19   const retrieveAllOrder = async () => {
20     const response = await axios.get(
21       "http://localhost:8080/api/user/admin/allorder"
22     );
23     console.log(response.data);
24     return response.data;
25   };
26
27   return (
28     <div className="mt-3">
29       <div
30         className="card form-card ms-2 me-2 mb-5 custom-bg border-color "
31         style={{
32           height: "45rem",
33         }}
34       >
35         <div className="card-header custom-bg text-center bg-color">
36           <h2>All Orders</h2>
37         </div>
38         <div
39           className="card-body"

```

```

39   |   className="card-body"
40   |   style={{
41   |     overflowY: "auto",
42   |   }}
43 >   <div className="table-responsive">
44     <table className="table table-hover text-color text-center">
45       <thead className="table-bordered border-color bg-color custom-bg-text">
46         <tr>
47           <th scope="col">Order Id</th>
48           <th scope="col">Product</th>
49           <th scope="col">Name</th>
50           <th scope="col">Description</th>
51           <th scope="col">Quantity</th>
52           <th scope="col">Total Price</th>
53           <th scope="col">Customer Name</th>
54           <th scope="col">Street</th>
55           <th scope="col">City</th>
56           <th scope="col">Pin code</th>
57           <th scope="col">Mobile No.</th>
58           <th scope="col">Order Date</th>
59           <th scope="col">Delivery Date</th>
60           <th scope="col">Delivery Status</th>
61           <th scope="col">Delivery Person</th>
62           <th scope="col">Delivery Mobile No</th>
63         </tr>
64       </thead>
65       <tbody>
66         {allOrderData.map((orderData) => {
67           return (
68             <tr>
69               <td>
70                 |   <b>{orderData.orderId}</b>
71               </td>
72               <td>
73                 <img
74                   src={
75                     "http://localhost:8080/api/product/" +
76                     orderData.productImage
77                   }
78                   orderData.productImage
79                   class="img-fluid"
80                   alt="product_pic"
81                   style={{
82                     |   maxWidth: "90px",
83                   }}
84                   />
85               </td>
86               <td>
87                 |   <b>{orderData.productName}</b>
88               </td>
89               <td>
90                 |   <b>{orderData.productDescription}</b>
91               </td>
92               <td>
93                 |   <b>{orderData.quantity}</b>
94               </td>
95               <td>
96                 |   <b>{orderData.totalPrice}</b>
97               </td>
98               <td>
99                 |   <b>{orderData.userName}</b>
100              </td>
101              <td>
102                |   <b>{orderData.address.street}</b>
103              </td>
104
105              <td>
106                |   <b>{orderData.address.city}</b>
107              </td>
108              <td>
109                |   <b>{orderData.address.pincode}</b>
110              </td>

```

```

111  <td>
112      <b>{orderData.userPhone}</b>
113  </td>
114  <td>
115      <b>{orderData.orderDate}</b>
116  </td>
117  <td>
118      <b>{orderData.deliveryDate}</b>
119  </td>
120  <td>
121      <b>{orderData.deliveryStatus}</b>
122  </td>
123  <td>
124      <b>{orderData.deliveryPersonName}</b>
125  </td>
126  <td>
127      <b>{orderData.deliveryPersonContact}</b>
128  </td>
129  </tr>
130      );
131  );
132  </tbody>
133  </table>
134  </div>
135  </div>
136  </div>
137  );
138  );
139  );
140
141  export default AllOrders;
142

```

Figure:32

- **MyCart**

The MyCart component takes three props: cartItems (an array of items in the cart), on Checkout (a function to handle the checkout process).

The component maps over the cartItems array to display each item in the cart

The total and a "Checkout" button are displayed at the bottom of the cart.

```

src > userComponent > ✎ MyCart.jsx > ...
1  import { useState, useEffect } from "react";
2  import axios from "axios";
3  import React from "react";
4  import { useNavigate } from "react-router-dom";
5
6  const MyCart = () => {
7      let navigate = useNavigate();
8      const user = JSON.parse(sessionStorage.getItem("active-user"));
9      const [totalPrice, setTotalPrice] = useState("");
10     const [myCartData, setMyCartData] = useState([]);
11
12     useEffect(() => {
13         const getMyCart = async () => {
14             const myCart = await retrieveMyCart();
15             if (myCart) {
16                 console.log("cart data is present :)");
17                 console.log(myCart.totalCartPrice);
18                 console.log(myCart.cartData);
19                 setTotalPrice(myCart.totalCartPrice);
20                 setMyCartData(myCart.cartData);
21             }
22         };
23
24         getMyCart();
25     }, []);
26
27     const retrieveMyCart = async () => {
28         const response = await axios.get(
29             "http://localhost:8080/api/user/mycart?userId=" + user.id
30         );
31         console.log(response.data);
32         return response.data;
33     };
34
35     const deleteProductFromCart = (cartId, e) => {
36         const response = axios.get(
37             "http://localhost:8080/api/user/mycart/remove?cartId=" + cartId
38         );

```

```

42
43   const checkout = (e) => {
44     e.preventDefault();
45     console.log("CHECKOUT PAGE REQUEST");
46     navigate("/user/order/payment", { state: { priceToPay: totatPrice } });
47   };
48
49   return (
50     <div className="mt-3">
51       <div
52         className="card form-card ms-2 me-2 mb-5 custom-bg border-color"
53         style={{
54           height: "45rem",
55         }}
56       >
57         <div className="card-header text-center bg-color custom-bg-text">
58           <h2>My Cart</h2>
59         </div>
60         <div
61           className="card-body"
62           style={{
63             overflowY: "auto",
64           }}
65         >
66           <div className="table-responsive">
67             <table className="table table-hover custom-bg-text text-center">
68               <thead className="bg-color table-bordered border-color">
69                 <tr>
70                   <th scope="col">Product</th>
71                   <th scope="col">Name</th>
72                   <th scope="col">Description</th>
73                   <th scope="col">Quantity</th>
74                   <th scope="col">Action</th>
75                 </tr>
76               </thead>
77               <tbody className="text-color">
78                 {myCartData.map((cartData) => {
79                   return (
80                     <tr>
81                       <td>
82                         <img
83                           src={
84                             "http://localhost:8080/api/product/" +
85                             cartData.productImage
86                           }
87                           class="img-fluid"
88                           alt="product_pic"
89                           style={{
90                             maxWidth: "90px",
91                           }}
92                         />
93                       </td>
94                       <td>
95                         <b>{cartData.productName}</b>
96                       </td>
97                       <td>
98                         <b>{cartData.productDescription}</b>
99                       </td>
100                      <td>
101                        <b>{cartData.quantity}</b>
102                      </td>
103                      <td>
104                        <button
105                          className="btn bg-color custom-bg-text btn-sm"
106                          onClick={() => deleteProductFromCart(cartData.cartId)}
107                        >
108                          Delete
109                        </button>
110                      </td>

```

```

111           |     </tr>
112           |   );
113           | });
114           | </tbody>
115           | </table>
116       </div>
117   </div>
118   <div className="card-footer custom-bg">
119     <div className="float-right">
120       <div
121         |   className="text-color me-2"
122         |   style={{
123         |     textAlign: "right",
124         |   }}
125       >
126         |   <h5>Total Price: &#8377; {totatPrice}/-</h5>
127     </div>
128
129     <div className="float-end me-2">
130       <button
131         |   type="submit"
132         |   className="btn bg-color custom-bg-text mb-3"
133         |   onClick={checkout}
134       >
135         |   Checkout
136       </button>
137     </div>
138   </div>
139 </div>
140 </div>
141 >;
142 );
143 >;
144 >
145 export default MyCart;
146

```

Figure:33

- **MyOrders**

The MyOrders component takes a userId prop, representing the ID of the user whose orders are being displayed.

The useEffect hook is used to trigger the fetchUserOrders function when the component mounts or when the userId prop changes.

```

src > userComponent > > MyOrder.jsx > ...
1 import { useState, useEffect } from "react";
2 import axios from "axios";
3 import React from "react";
4
5 const MyOrder = () => {
6   const user = JSON.parse(sessionStorage.getItem("active-user"));
7   const [myOrderData, setMyOrderData] = useState([]);
8
9   useEffect(() => {
10     const getMyOrder = async () => {
11       const myOrder = await retrieveMyOrder();
12       if (myOrder) {
13         console.log("my order data is present :)");
14
15         setMyOrderData(myOrder);
16       }
17     };
18
19     getMyOrder();
20   }, []);
21
22   const retrieveMyOrder = async () => {
23     const response = await axios.get(
24       "http://localhost:8080/api/user/myorder?userId=" + user.id
25     );
26     console.log(response.data);
27     return response.data;
28   };
29
30   return (
31     <div className="mt-3">
32       <div
33         |   className="card form-card ms-2 me-2 mb-5 custom-bg"
34         |   style={{
35         |     height: "45rem",
36         |   }}
37       >
38         <div className="card-header text-center bg-color custom-bg-text">
39           <h2>My Orders</h2>

```

```

39 |           <h2>My Orders</h2>
40 |       </div>
41 |       <div
42 |           className="card-body"
43 |           style={{
44 |               overflowY: "auto",
45 |           }}
46 |       >
47 |           <div className="table-responsive">
48 |               <table className="table table-hover custom-bg-text text-center">
49 |                   <thead className="bg-color table-bordered border-color">
50 |                       <tr>
51 |                           <th scope="col">Order Id</th>
52 |                           <th scope="col">Product</th>
53 |                           <th scope="col">Name</th>
54 |                           <th scope="col">Description</th>
55 |                           <th scope="col">Quantity</th>
56 |                           <th scope="col">Total Price</th>
57 |                           <th scope="col">Order Date</th>
58 |                           <th scope="col">Delivery Date</th>
59 |                           <th scope="col">Delivery Status</th>
60 |                           <th scope="col">Delivery Person</th>
61 |                           <th scope="col">Delivery Mobile No</th>
62 |                       </tr>
63 |                   </thead>
64 |                   <tbody className="text-color">
65 |                       {myOrderData.map((orderData) => {
66 |                           return (
67 |                               <tr>
68 |                                   <td>
69 |                                       <b>{orderData.orderId}</b>
70 |                                   </td>
71 |                                   <td>
72 |                                       <img
73 |                                           src={
74 |                                               "http://localhost:8080/api/product/" +
75 |                                               orderData.productImage
76 |                                           }
77 |                                           class="img-fluid"
78 |                                           alt="product_pic"
79 |                                           style={{
80 |                                               maxWidth: "90px",
81 |                                           }}
82 |                                       />
83 |                                   </td>
84 |                                   <td>
85 |                                       <b>{orderData.productName}</b>
86 |                                   </td>
87 |                                   <td>
88 |                                       <b>{orderData.productDescription}</b>
89 |                                   </td>
90 |                                   <td>
91 |                                       <b>{orderData.quantity}</b>
92 |                                   </td>
93 |                                   <td>
94 |                                       <b>{orderData.totalPrice}</b>
95 |                                   </td>
96 |                                   <td>
97 |                                       <b>{orderData.orderDate}</b>
98 |                                   </td>
99 |                                   <td>
100 |                                       <b>{orderData.deliveryDate}</b>
101 |                                   </td>
102 |                                   <td>
103 |                                       <b>{orderData.deliveryStatus}</b>
104 |                                   </td>
105 |                                   <td>
106 |                                       <b>{orderData.deliveryPersonName}</b>
107 |                                   </td>
108 |                                   <td>
109 |                                       <b>{orderData.deliveryPersonContact}</b>
110 |                                   </td>
111 |                               </tr>
112 |                           );
113 |                       );
114 |                   </tbody>
115 |               </table>
116 |           </div>
117 |       </div>
118 |   </div>
119 | 
```

```

112
113     },
114     ),
115   ),
116   ),
117   ),
118   ),
119   );
120 );
121 };
122
123 export default MyOrder;
124

```

Figure:34

- **RegisterAdminForm**

Creating a RegisterAdminForm component involves building a form that allows administrators to register in your system.

The RegisterAdminForm component is a controlled form component using the useState hook to manage the state of the form inputs.

The component takes an onRegisterAdmin prop, which is a function that will be called when the form is submitted. This allows the parent component to handle the registration of administrators.

The form includes input fields for the username, email, and password. It also includes basic validation to check if the fields are not empty.

```

src > userComponent > RegisterAdminForm.jsx > ...
1 import { useState } from "react";
2
3 const RegisterAdminForm = () => {
4   const [user, setUser] = useState({
5     firstName: "",
6     lastName: "",
7     emailId: "",
8     password: "",
9     phoneNo: "",
10    street: "",
11    city: "",
12    pincode: ""
13 });
14
15 const handleUserInput = (e) => {
16   setUser({ ...user, [e.target.name]: e.target.value });
17 };
18
19 const sav Follow link (ctrl + click)
20 fetch("http://localhost:8080/api/user/admin/register", {
21   method: "POST",
22   headers: {
23     Accept: "application/json",
24     "Content-Type": "application/json",
25   },
26   body: JSON.stringify(user),
27 }).then((result) => {
28   console.warn("result", result);
29   result.json().then((res) => {
30     console.log("response", res);
31   });
32 });
33
34

```

```

35     return (
36       <div>
37         <div class="mt-2 d-flex align-items-center justify-content-center ms-2 me-2 mb-2">
38           <div class="card form-card border-color" style={{ width: "25rem" }}>
39             <div className="card-header bg-color">
40               <h5 className="card-title">Add Admin</h5>
41             </div>
42             <div className="card-body">
43               <form>
44                 <div className="mb-3">
45                   <label htmlFor="title" className="form-label">
46                     First Name
47                   </label>
48                   <input
49                     type="text"
50                     className="form-control"
51                     id="firstName"
52                     name="firstName"
53                     onChange={handleUserInput}
54                     value={user.firstName}
55                   />
56                 </div>
57                 <div className="mb-3">
58                   <label htmlFor="description" className="form-label">
59                     Last Name
60                   </label>
61                   <input
62                     type="text"
63                     className="form-control"
64                     id="lastName"
65                     name="lastName"
66                     onChange={handleUserInput}
67                     value={user.lastName}
68                   />
69                 </div>
70
71                 <div className="mb-3">
72                   <label className="form-label">Email Id</label>
73                   <input
74                     (property) class: string
75                     className="form-control"
76                     id="emailId"
77                     name="emailId"
78                     onChange={handleUserInput}
79                     value={user.emailId}
80                   />
81                 </div>
82
83                 <div className="mb-3 mt-1">
84                   <label htmlFor="quantity" className="form-label">
85                     Password
86                   </label>
87                   <input
88                     type="password"
89                     className="form-control"
90                     id="password"
91                     name="password"
92                     onChange={handleUserInput}
93                     value={user.password}
94                   />
95                 </div>
96
97                 <div className="mb-3">
98                   <label htmlFor="price" className="form-label">
99                     Mobile No
100                  </label>
101                  <input
102                     type="number"
103                     className="form-control"
104                     id="phoneNo"
105                     name="phoneNo"
106                     onChange={handleUserInput}
107                     value={user.phoneNo}
108                   />
109                 </div>

```

```

110
111      <div class="mb-3">
112          <label for="description" class="form-label">
113              Street
114          </label>
115          <textarea
116              class="form-control"
117              id="street"
118              name="street"
119              rows="3"
120              onChange={handleUserInput}
121              value={user.street}
122          />
123      </div>
124
125      <div class="mb-3">
126          <label for="price" class="form-label">
127              City
128          </label>
129          <input
130              type="text"
131              class="form-control"
132              id="city"
133              name="city"
134              onChange={handleUserInput}
135              value={user.city}
136          />
137      </div>
138
139      <div class="mb-3">
140          <label for="pincode" class="form-label">
141              Pincode
142          </label>
143          <input
144              type="number"
145              class="form-control"
146              id="pincode"
147              name="pincode"
148              onChange={handleUserInput}
149              value={user.pincode}
150          />
151      </div>
152
153      <button
154          type="submit"
155          class="btn custom-bg text-color"
156          onClick={saveUser}
157      >
158          Register
159      </button>
160  </form>
161 </div>
162 </div>
163 </div>
164 </div>
165 );
166 };
167
168 export default RegisterAdminForm;
169

```

Figure:35

- **SearchOrder**

Creating a SearchOrder component involves building a component that allows users to search for specific orders.

The SearchOrder component is a controlled form component using the useState hook to manage the state of the form inputs.

```

src > userComponent > SearchOrder.jsx > ...
1  import { useState, useEffect } from "react";
2  import axios from "axios";
3  import "react-datepicker/dist/react-datepicker.css";
4
5  const SearchOrder = () => {
6    const [allOrderData, setAllOrderData] = useState([]);
7    const [orderId, setOrderId] = useState("");
8
9    const deliveryStatus = ["Delivered", "On the Way", "Processing"];
10   const deliveryTime = ["Morning", "Afternoon", "Evening", "Night"];
11
12   const [orderDeliveryStatus, setOrderDeliveryStatus] = useState({
13     orderId: "",
14     deliveryStatus: "",
15     deliveryTime: "",
16     deliveryDate: ""
17   });
18
19   const handleOrderDelivery = (e) => {
20     setOrderDeliveryStatus({
21       ...orderDeliveryStatus,
22       [e.target.name]: e.target.value,
23     });
24   };
25
26   const getAllOrder = async () => {
27     const allOrder = await retrieveAllOrder();
28     if (allOrder) {
29       setAllOrderData(allOrder);
30     }
31   };
32
33   const retrieveAllOrder = async () => {
34     const response = await axios.get(
35       "http://localhost:8080/api/user/admin/showorder?orderId=" + orderId
36     );
37     console.log(response.data);
38     return response.data;
39   };
40
41   const searchOrderBy = (e) => {
42     getAllOrder();
43     setOrderId("");
44     e.preventDefault();
45   };
46
47   const updateDeliveryStatus = (e) => {
48     console.log("CLICKED DELIVERY STATUS UPDATED");
49     fetch("http://localhost:8080/api/user/admin/order/deliveryStatus/update", {
50       method: "POST",
51       headers: {
52         Accept: "application/json",
53         "Content-Type": "application/json",
54       },
55       body: JSON.stringify(orderDeliveryStatus),
56     }).then((result) => {
57       console.log("result", result);
58       result.json().then((res) => {
59         console.log("response", res);
60         setAllOrderData({
61           orderId: "",
62           deliveryStatus: "",
63           deliveryTime: "",
64           deliveryDate: ""
65         });
66
67         setAllOrderData(res);
68       });
69     });
70

```

```

70
71     e.preventDefault();
72 };
73
74 return (
75   <div>
76     <d (property) React.HTMLAttributes<HTMLDivElement>.className?: string | undefined
77       className="card form-card mt-1 ms-2 me-2 mb-2 custom-bg"
78       style={{
79         height: "35rem",
80       }}
81   >
82     <div className="card-header text-center bg-color custom-bg-text">
83       <h4>Search Customer Orders</h4>
84     </div>
85     <div
86       className="card-body"
87       style={{
88         overflowY: "auto",
89       }}
90   >
91     <form class="row g-3">
92       <div class="col-auto">
93         <input
94           type="text"
95           class="form-control"
96           id="inputPassword2"
97           placeholder="Enter Order Id..."
98           onChange={(e) => setOrderId(e.target.value)}
99           value={orderId}
100          />
101        </div>
102       <div class="col-auto">
103         <button
104           type="submit"
105           class="btn bg-color custom-bg-text mb-3"
106           onClick={searchOrderBy}
107         >
108           Search
109         </button>
110       </div>
111     </form>
112     <div className="table-responsive">
113       <table className="table table-hover custom-bg-text text-center">
114         <thead className="bg-color table-bordered border-color">
115           <tr>
116             <th scope="col">Order Id</th>
117             <th scope="col">Product</th>
118             <th scope="col">Name</th>
119             <th scope="col">Description</th>
120             <th scope="col">Quantity</th>
121             <th scope="col">Total Price</th>
122             <th scope="col">Customer Name</th>
123             <th scope="col">Street</th>
124             <th scope="col">City</th>
125             <th scope="col">Pin code</th>
126             <th scope="col">Mobile No.</th>
127             <th scope="col">Order Date</th>
128             <th scope="col">Delivery Date</th>
129             <th scope="col">Delivery Status</th>
130             <th scope="col">Delivery Person</th>
131             <th scope="col">Delivery Mobile No</th>
132           </tr>
133         </thead>

```

```

134     <tbody className="text-color">
135       {allOrderData.map((orderData) => {
136         return (
137           <tr>
138             <td>
139               <b>{orderData.orderId}</b>
140             </td>
141             <td>
142               <img
143                 src={
144                   "http://localhost:8080/api/product/" +
145                   orderData.productImage
146                 }
147                 class="img-fluid"
148                 alt="product_pic"
149                 style={{
150                   maxWidth: "90px",
151                 }}
152               />
153             </td>
154             <td>
155               <b>{orderData.productName}</b>
156             </td>
157             <td>
158               <b>{orderData.productDescription}</b>
159             </td>
160             <td>
161               <b>{orderData.quantity}</b>
162             </td>
163             <td>
164               <b>{orderData.totalPrice}</b>
165             </td>
166             <td>
167               <b>{orderData.userName}</b>
168             </td>
169             <td>
170               <b>{orderData.address.street}</b>
171             </td>
172             <td>
173               <b>{orderData.address.city}</b>
174             </td>
175             <td>
176               <b>{orderData.address.pincode}</b>
177             </td>
178             <td>
179               <b>{orderData.userPhone}</b>
180             </td>
181             <td>
182               <b>{orderData.orderDate}</b>
183             </td>
184             <td>
185               <b>{orderData.deliveryDate}</b>
186             </td>
187             <td>
188               <b>{orderData.deliveryStatus}</b>
189             </td>
190             <td>
191               <b>{orderData.deliveryPersonName}</b>
192             </td>
193             <td>
194               <b>{orderData.deliveryPersonContact}</b>
195             </td>
196           </tr>
197         );
198       );
199     );
200   );
201 </tbody>
202 </table>
203 </div>
204 </div>
205

```

```

205
206    <div>
207      <div className="card form-card ms-2 me-2 mb-2 custom-bg">
208        <div className="card-header text-center bg-color custom-bg-text">
209          <h4>Update Delivery Status</h4>
210        </div>
211        <div className="card-body">
212          <form class="row g-3">
213            <div class="col-auto">
214              <label className="text-color">
215                <b>Order Id</b>
216              </label>
217              <input
218                type="text"
219                class="form-control"
220                id="inputPassword2"
221                placeholder="Enter Order Id..."
222                name="orderId"
223                onChange={handleOrderDelivery: (e: any) => void}
224                value={handleOrderDelivery.orderId}
225              />
226            </div>
227            <div class="col-auto">
228              <label className="text-color">
229                <b>Select Delivery Date</b>
230              </label>
231              <input
232                type="date"
233                class="form-control"
234                id="inputPassword2"
235                name="deliveryDate"
236                placeholder="dd-mm-yyyy"
237                min="1997-01-01"
238                max="2030-12-31"
239                value={handleOrderDelivery.deliveryDate}
240                onChange={handleOrderDelivery}
241              />
242            </div>

```



```

242          </div>
243          <div class="col-auto">
244            <label className="text-color">
245              <b>Delivery Time</b>
246            </label>
247
248            <select
249              name="deliveryTime"
250              value={handleOrderDelivery.deliveryTime}
251              onChange={handleOrderDelivery}
252              className="form-control"
253            >
254              <option value="">Select Delivery Time</option>
255
256              {deliveryTime.map((time) => {
257                return <option value={time}> {time} </option>;
258              })}
259            </select>
260          </div>
261          <div class="col-auto">
262            <label className="text-color">
263              <b>Delivery Status</b>
264            </label>
265            <select
266              name="deliveryStatus"
267              value={handleOrderDelivery.deliveryStatus}
268              onChange={handleOrderDelivery}
269              className="form-control"
270            >
271              <option value="">Select Delivery Status</option>
272
273              {deliveryStatus.map((status) => {
274                return <option value={status}> {status} </option>;
275              })}
276            </select>
277          </div>

```

```

277         </div>
278     <div class="col-auto">
279         <button
280             type="submit"
281             class="btn bg-color custom-bg-text mt-4"
282             onClick={updateDeliveryStatus}
283         >
284             Update Delivery Status
285         </button>
286     </div>
287     </form>
288   </div>
289   </div>
290 </div>
291 );
292 };
293
294
295 export default SearchOrder;
296

```

Figure:36

- **UserLoginForm**

Creating a UserLoginForm component involves building a form that allows users to log in.

The UserLoginForm component is a controlled form component using the useState hook to manage the state of the form inputs.

```

src > userComponent > ✎ UserLoginForm.jsx > ↗ UserLoginForm
  1 import { useState } from "react";
  2 import { ToastContainer, toast } from "react-toastify";
  3 import "react-toastify/dist/ReactToastify.css";
  4 import { useNavigate } from "react-router-dom";
  5 import { Link } from "react-router-dom";
  6
  7 const UserLoginForm = () => {
  8   let navigate = useNavigate();
  9
 10   const [loginRequest, setLoginRequest] = useState({
 11     emailId: "",
 12     password: "",
 13     role: "",
 14   });
 15
 16   const handleUserInput = (e) => {
 17     setLoginRequest({ ...loginRequest, [e.target.name]: e.target.value });
 18   };
 19
 20   const loginAction = (e) => {
 21     fetch("http://localhost:8080/api/user/login", {
 22       method: "POST",
 23       headers: {
 24         Accept: "application/json",
 25         "Content-Type": "application/json",
 26       },
 27       body: JSON.stringify(loginRequest),
 28     }).then((result) => {
 29       console.log("result", result);
 30       result.json().then((res) => {
 31         console.log(res);
 32       });
 33     });
 34   };

```

```

32
33     if (res.role === "Admin") {
34         console.log("Working fine:");
35         sessionStorage.setItem("active-admin", JSON.stringify(res));
36     } else if (res.role === "Customer") {
37         sessionStorage.setItem("active-user", JSON.stringify(res));
38     } else if (res.role === "Delivery") {
39         sessionStorage.setItem("active-delivery", JSON.stringify(res));
40     }
41
42     toast.success("logged in successfully!!!", {
43         position: "top-center",
44         autoClose: 1000,
45         hideProgressBar: false,
46         closeOnClick: true,
47         pauseOnHover: true,
48         draggable: true,
49         progress: undefined,
50     });
51
52     navigate("/home");
53     window.location.reload(true);
54 });
55 });
56 e.preventDefault();
57 };
58
59 return (
60     <div>
61         <div className="mt-2 d-flex align-items-center justify-content-center">
62             <div
63                 className="card form-card border-color custom-bg"
64                 style={{ width: "25rem" }}
65             >
66                 <div className="card-header bg-color text-center custom-bg-text">
67                     <h4 className="card-title">User Login</h4>
68                 </div>
69                 <div className="card-body">
70                     <form>
71                         <div className="mb-3 text-color">
72                             <label htmlFor="role" className="form-label">
73                                 <b>User Role</b>
74                             </label>
75                             <select
76                                 onChange={handleUserInput}
77                                 className="form-control"
78                                 name="role"
79                             >
80                                 <option value="0">Select Role</option>
81                                 <option value="Admin"> Admin </option>
82                                 <option value="Customer"> Customer </option>
83                                 <option value="Delivery"> Delivery Person </option>
84                             </select>
85                         </div>
86                     </form>

```

```

87 |         <div className="mb-3 text-color">
88 |             <label for="emailId" className="form-label">
89 |                 <b>Email Id</b>
90 |             </label>
91 |             <input
92 |                 type="email"
93 |                 className="form-control"
94 |                 id="emailId"
95 |                 name="emailId"
96 |                 onChange={handleUserInput}
97 |                 value={loginRequest.emailId}
98 |             />
99 |         </div>
100 |         <div className="mb-3 text-color">
101 |             <label for="password" className="form-label">
102 |                 <b>Password</b>
103 |             </label>
104 |             <input
105 |                 type="password"
106 |                 className="form-control"
107 |                 id="password"
108 |                 name="password"
109 |                 onChange={handleUserInput}
110 |                 value={loginRequest.password}
111 |                 autoComplete="on"
112 |             />
113 |         </div>
114 |         <button
115 |             type="submit"
116 |             className="btn bg-color custom-bg-text w-50 p-2.5 rounded-pill"
117 |             onClick={loginAction}
118 |         >
119 |             Login
120 |         </button>
121 |         <Link to="/user/register" className="nav-link active m-1" aria-current="page"> New User?
122 |             <b className="text-color">Register Here</b>
123 |         </Link>
124 |         </div>
125 |         <ToastContainer />
126 |     </form>
127 |     </div>
128 | </div>
129 | </div>
130 | );
131 | };
132 | export default UserLoginForm;
133 |

```

Figure:37

Src/HomePage

Creating a HomePage component involves building the main landing page of your application.

The HomePage component renders a welcome message and a brief description of the e-commerce platform.

You can customize the content and layout of the HomePage component based on your application's design and requirements.

```

    ⚡ HomePage.jsx > ...
1   import Carousel from "./Carousel";
2   import GetAllCategories from "../productComponent/GetAllCategories";
3   import ProductCard from "../productComponent/ProductCard";
4   import axios from "axios";
5   import { useState, useEffect } from "react";
6   import { useParams } from "react-router-dom";
7
8   const HomePage = () => {
9     const [products, setProducts] = useState([]);
10
11   const { categoryId } = useParams();
12
13   useEffect(() => {
14     const getAllProducts = async () => {
15       const allProducts = await retrieveAllProducts();
16       if (allProducts) {
17         setProducts(allProducts);
18       }
19     };
20
21     const getProductsByCategory = async () => {
22       const allProducts = await retrieveProductsByCategory();
23       if (allProducts) {
24         setProducts(allProducts);
25       }
26     };
27
28     if (categoryId == null) {
29       console.log("Category Id is null");
30       getAllProducts();
31     } else {
32       console.log("Category Id is NOT null");
33       getProductsByCategory();
34     }
35   }, [categoryId]);
36
37
38   const retrieveAllProducts = async () => {
39     const response = await axios.get("http://localhost:8080/api/product/all");
40
41     return response.data;
42   };
43
44   const retrieveProductsByCategory = async () => {
45     const response = await axios.get(
46       "http://localhost:8080/api/product/category?categoryId=" + categoryId
47     );
48
49     return response.data;
50   };
51
52   return (
53     <div className="container-fluid mb-2">
54       <Carousel />
55       <div className="mt-2 mb-5">
56         <div className="row">
57           <div className="col-md-2">
58             <GetAllCategories />
59           </div>
60           <div className="col-md-10">
61             <div className="row row-cols-1 row-cols-md-4 g-4">
62               <div>
63                 {products.map((product) => {
64                   return <ProductCard item={product} />;
65                 })}
66               </div>
67             </div>
68           </div>
69         </div>
70       </div>
71     );
72   };
73
74   export default HomePage;
75

```

```

121         <Link to="/user/register" className="nav-link active m-1" aria-current="page"> New User?
122             <b className="text-color">Register Here</b>
123         </Link>
124         <ToastContainer />
125     </form>
126     </div>
127     </div>
128   </div>
129 );
130 );
131 };
132
133 export default UserLoginForm;
134

```

Figure:38

Src/App.js

The App component includes several components, such as HomePage, UserLoginContainer, AdminLoginPage, MyOrders, AdminRegistration, and OrderSearchContainer and etc...,

Each component is imported and rendered within the App component.

You may need to pass appropriate props to the components or handle state management based on your application's requirements.

Ensure that you have proper styling for the components and adjust the layout as needed.

```

src > JS App.js > ...
1 import Header from "./pages/Header";
2 import AddProductForm from "./productComponent/AddProductForm";
3 import { Routes, Route } from "react-router-dom";
4 import ContactUs from "./pages/ContactUs";
5 import AboutUs from "./pages/AboutUs";
6 import AddCategoryForm from "./productComponent/AddCategoryForm";
7 import HomePage from "./pages/HomePage";
8 import Product from "./pages/Product";
9 import AddUserForm from "./userComponent/AddUserForm";
10 import UserLoginForm from "./userComponent/UserLoginForm";
11 import MyCart from "./userComponent/MyCart";
12 import AddCardDetails from "./pages/AddCardDetails";
13 import MyOrder from "./userComponent/MyOrder";
14 import AllOrders from "./userComponent/AllOrders";
15 import SearchOrder from "./userComponent/SearchOrder";
16 import RegisterAdminForm from "./userComponent/RegisterAdminForm";
17 import AdminLoginPage from "./userComponent/AdminLoginPage";
18 import AddDeliveryPerson from "./userComponent/AddDeliveryPerson";
19 import DeliveryPersonLogin from "./userComponent/DeliveryPersonLogin";
20 import AssignDeliveryToOrders from "./userComponent/AssignDeliveryToOrders";
21 import MyDeliveries from "./userComponent/MyDeliveries";
22
23 function App() {
24   return (
25     <div>
26       <Header />
27       <Routes>
28         <Route path="/" element={<HomePage />} />
29         <Route path="/home" element={<HomePage />} />
30         <Route path="/home/all/product/categories" element={<HomePage />} />
31         <Route path="contact" element={<ContactUs />} />
32         <Route path="about" element={<AboutUs />} />
33         <Route path="addproduct" element={<AddProductForm />} />
34         <Route path="addcategory" element={<AddCategoryForm />} />
35         <Route path="/product" element={<Product />} />
36         <Route path="/user/register" element={<AddUserForm />} />
37         <Route path="/user/login" element={<UserLoginForm />} />
38         <Route path="/user/admin/register" element={<RegisterAdminForm />} />
39         <Route path="/user/admin/login" element={<AdminLoginPage />} />

```

```

39   <Route path="/user/admin/login" element={<AdminLoginPage />} />
40   <Route
41     path="/user/deliveryperson/register"
42     element={<AddDeliveryPerson />}
43   />
44   <Route
45     path="/user/deliveryperson/login"
46     element={<DeliveryPersonLogin />}
47   />
48   <Route
49     path="/home/product/category/:categoryId/:categoryName"
50     element={<HomePage />}
51   />
52   <Route
53     path="/product/:productId/category/:categoryId"
54     element={<Product />}
55   />
56   <Route path="/user/mycart" element={<MyCart />} />
57   <Route path="/user/order/payment" element={<AddCardDetails />} />
58   <Route path="/user/myorder" element={<MyOrder />} />
59   <Route path="/user/admin/allorder" element={<AllOrders />} />
60   <Route path="/user/admin/searchOrder" element={<SearchOrder />} />
61   <Route
62     path="/user/admin/assigndelivery"
63     element={<AssignDeliveryToOrders />}
64   />
65   <Route path="/user/delivery/myorders" element={<MyDeliveries />} />
66 </Routes>
67 </div>
68 );
69 }
70 export default App;
71
72

```

Figure:39

src/index.css

The index.css file is typically used to apply global styles to your React application. It is the main CSS file that gets loaded by default when your application starts.

```

src > # index.css > body
1  body {
2    Sets the background color of an element.
3    (Edge 12, Firefox 1, Safari 1, Chrome 1, IE 4, Opera 3)
4    Syntax: <color>
5    MDN Reference
6    background-color: #FDF1E7
7  }
8
9
10
11 code {
12   font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
13   | monospace;
14 }
15
16 .custom-bg {
17   background-color: #FDF1E7;
18 }
19
20 .text-color {
21   color: #25493F;
22 }
23
24 .border-color {
25   border-color: #25493F;
26 }
27
28 .custom-bg-text {
29   color: #FDF1E7;
30 }
31
32 .bg-color {
33   background-color: #25493F;
34 }
35

```

```

35   .rounded-card {
36     padding: 1.5em .5em .5em;
37     border-radius: 2em;
38     box-shadow: 0 5px 10px □ rgba(0,0,0,.2);
39   }
40 }
41
42 .form-card {
43   padding: .2em .2em .2em;
44   border-radius: 1em;
45   box-shadow: 0 5px 10px □ rgba(0,0,0,.2);
46 }
47
48 .card {
49   transition: transform 0.2s ease;
50 }
51
52 .card:hover .product-card {
53   transform: scale(1.02);
54 }

```

Figure:40

Src/index.js

The index.js file is the entry point of a React application. It is the first file that gets executed when your application runs.

```

src > JS index.js > ...
1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import App from "./App";
4 import { BrowserRouter } from "react-router-dom";
5 import "./index.css";
6
7 const root = ReactDOM.createRoot(document.getElementById("root"));
8 root.render(
9   <BrowserRouter>
10   |   <App />
11   </BrowserRouter>
12 );
13

```

Figure:41

4.3 User Interface Design

4.3.1 Layout and Structure

Grid Layout:

Used a responsive grid layout for product displays using BootStrap.

Navigation:

Implemented a clear navigation menu with Bootstrap components.

4.3.2 Visual Elements

Typography:

Used readable fonts and maintained consistency.

Buttons:

Used Bootstrap buttons and make primary actions visually prominent.

Images and Multimedia:

Optimize images for fast loading and use Bootstrap classes for responsive images.

4.3.3 Responsive Design

Bootstrap Grid System:

Used Bootstrap's grid system for responsive layouts.

4.3.4 Color Scheme

Brand Colors:

Used a consistent color scheme aligning with the brand and title of website identity.

Contrast:

Ensure sufficient contrast for readability and accessibility.

4.4 User Experience Considerations

4.4.1 User Flow

Homepage:

This website is designed for an intuitive user flow from the homepage to product discovery.

4.4.2 Product Pages:

Implemented clear pathways from product pages to the shopping cart in our website.

4.4.3 Categories

Implement a prominent Categories list which added by the admin.

4.4.4 Mobile-Friendly Interactions

Responsive Buttons:

Design responsive buttons and links are appropriately sized for mobile users using bootstrap.

Form Validation:

Implemented inline validation for form inputs.

4.5 Features and functionalities

4.5.1 Homepage:

Interface: Website Name, Login Button, Display productise.,

Product Categories: Organize products into categories for easy navigation.

4.5.2 Product Listing:

Grid View: Display products in a grid layout with images, titles, and prices.

4.5.3. Product Details:

High-Quality Images: Showcase detailed images of the product.

Product Description: Provide information about the product's features and specifications.

Add to Cart: Include a button to add the product to the shopping cart.

4.5.4. Shopping Cart:

Cart Summary: Show a summary of items in the cart.

Proceed to Checkout: Direct users to the checkout process.

4.5.5 Responsive Design:

Mobile Optimization: Ensured the website is fully responsive for various devices.

Touch-Friendly Interactions: Implement touch-friendly elements for mobile users.

4.5.6 Accessibility:

Semantic HTML: Used semantic HTML elements for better accessibility.

Alt Text: Include descriptive alt text for images.

5. Back-end Development

5.1 Project Structure:

- Created a well-organized project structure to maintain clarity and separation of concerns.

Example:

```
/src
  /main
    /java
      /com.example.ecommerce
        /config
        /controller
        /model
        /repository
        /service
    /resources
      application.properties
```

Figure:42

In Project:

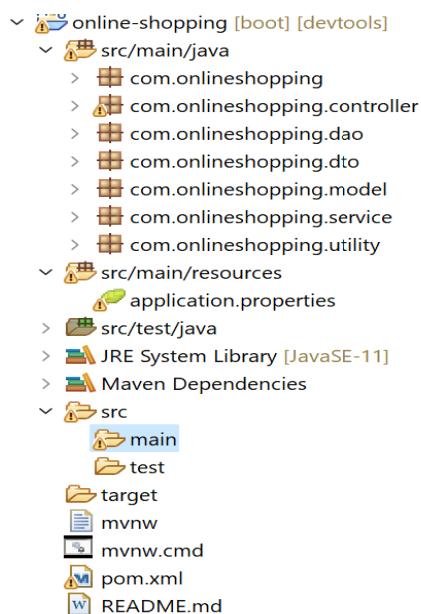


Figure:43

- **Config:** Configuration classes for Spring Boot.

- **Controller:** Handles incoming HTTP requests.
- **Model:** Entity classes representing database tables.
- **Service:** Business logic and service classes.

Configure MySQL Database

MySQL database connection:

```

spring.datasource.url=jdbc:mysql://localhost:3306/your_database_name
spring.datasource.username=your_mysql_username
spring.datasource.password=your_mysql_password

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

```

Figure:44

Replace your_database_name, your_mysql_username, and your_mysql_password with your actual MySQL database information.

5.2 Implementation

5.2.1 Spring Boot Application:

Created a main class to bootstrap the Spring Boot application.

```

package com.onlineshopping;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class OnlineShoppingApplication {

    public static void main(String[] args) {
        SpringApplication.run(OnlineShoppingApplication.class, args);
    }

}

```

Figure:45

5.2.2 Controllers

Cart Controller:

A Cart Controller in a Spring Boot application. This controller will handle operations related to a shopping cart.

```

1 package com.onlineshopping.controller;
2
3 import java.util.ArrayList;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30 @RestController
31 @RequestMapping("api/user/")
32 @CrossOrigin(origins = "http://localhost:3000")
33 public class CartController {
34
35     @Autowired
36     private CartDao cartDao;
37
38     @Autowired
39     private UserDao userDao;
40
41     @Autowired
42     private ProductDao productDao;
43
44     ObjectMapper objectMapper = new ObjectMapper();
45
46     @PostMapping("cart/add")
47     public ResponseEntity add(@RequestBody AddToCartRequest addToCartRequest) {
48
49         System.out.println("request came for ADD PRODUCT TO CART");
50         System.out.println(addToCartRequest);
51         Optional<User> optionalUser = userDao.findById(addToCartRequest.getUserId());
52         User user = null;
53         if(optionalUser.isPresent()) {
54             user = optionalUser.get();
55         }
56
57         Optional<Product> optionalProduct = productDao.findById(addToCartRequest.getProductId());
58         Product product = null;
59         if(optionalProduct.isPresent()) {
60             product = optionalProduct.get();
61         }
62
63         Cart cart = new Cart();
64         cart.setProduct(product);
65         cart.setQuantity(addToCartRequest.getQuantity());
66         cart.setUser(user);
67
68         cartDao.save(cart);
69
70         return new ResponseEntity(HttpStatus.OK);
71     }
72
73
74     @GetMapping("mycart")
75     public ResponseEntity getMyCart(@RequestParam("userId") int userId) throws JsonProcessingException {
76
77         System.out.println("request came for MY CART for USER ID : "+userId);
78
79         List<CartDataResponse> cartDatas = new ArrayList<>();
80
81         List<Cart> userCarts = cartDao.findByUserId(userId);
82
83         double totalCartPrice = 0;
84
85         for (Cart cart : userCarts) {
86             CartDataResponse cartData = new CartDataResponse();
87             cartData.setCartId(cart.getId());
88             cartData.setProductDescription(cart.getProduct().getDescription());
89             cartData.setProductName(cart.getProduct().getTitle());
90             cartData.setProductImage(cart.getProduct().getImageName());
91             cartData.setQuantity(cart.getQuantity());
92             cartData.setProductId(cart.getProduct().getId());
93
94             cartDatas.add(cartData);
95
96             double productPrice = Double.parseDouble(cart.getProduct().getPrice().toString());
97
98             totalCartPrice = totalCartPrice + (cart.getQuantity() * productPrice);
99
100        }
101

```

```
101
102     CartResponse cartResponse = new CartResponse();
103     cartResponse.setTotalCartPrice(String.valueOf(totalCartPrice));
104     cartResponse.setCartData(cartDatas);
105
106     String json = objectMapper.writeValueAsString(cartResponse);
107
108     System.out.println(json);
109
110     return new ResponseEntity(cartResponse, HttpStatus.OK);
111 }
112
113
114 @GetMapping("mycart/remove")
115 public ResponseEntity removeCartItem(@RequestParam("cartId") int cartId) throws JsonProcessingException {
116
117     System.out.println("request came for DELETE CART ITEM WHOSE ID IS : "+cartId);
118
119     Optional<Cart> optionalCart = this.cartDao.findById(cartId);
120     Cart cart = new Cart();
121
122     if(optionalCart.isPresent()) {
123         cart = optionalCart.get();
124     }
125
126     this.cartDao.delete(cart);
127
128     return new ResponseEntity("SUCCESS", HttpStatus.OK);
129 }
130
131
132 }
133
```

Figure:46

Category Controller:

A Category Controller in a Spring Boot application. This controller will handle operations related to product categories.

```

1 package com.onlineshopping.controller;
2
3④ import java.util.List;[]
4
5
6 @RestController
7 @RequestMapping("api/category")
8 @CrossOrigin(origins = "http://localhost:3000")
9 public class CategoryController {
10
11    @Autowired
12    private CategoryDao categoryDao;
13
14    @GetMapping("all")
15    public ResponseEntity<List<Category>> getAllCategories() {
16
17        System.out.println("request came for getting all categories");
18
19        List<Category> categories = this.categoryDao.findAll();
20
21        ResponseEntity<List<Category>> response = new ResponseEntity<>(categories, HttpStatus.OK);
22
23        System.out.println("response sent");
24
25        return response;
26    }
27
28
29    @PostMapping("add")
30    public ResponseEntity add(@RequestBody Category category) {
31
32        System.out.println("request came for add category");
33
34        Category c = categoryDao.save(category);
35
36        if(c != null) {
37            System.out.println("response sent");
38            return new ResponseEntity( c ,HttpStatus.OK);
39        }
40
41        else {
42            System.out.println("response sent");
43            return new ResponseEntity("Failed to add category!",HttpStatus.INTERNAL_SERVER_ERROR);
44        }
45
46    }
47
48}

```

Figure:47

Order Controller:

An Order Controller in a Spring Boot application. This controller will handle operations related to orders.

```

1 package com.onlineshopping.controller;
2
3④ import java.time.LocalDateTime;□
4
5④ @RestController
6④ @RequestMapping("api/user/")
7④ @CrossOrigin(origins = "*", allowedHeaders = "*")
8 public class OrderController {
9
10    @Autowired
11    private OrderDao orderDao;
12
13    @Autowired
14    private CartDao cartDao;
15
16    @Autowired
17    private UserDao userDao;
18
19    @Autowired
20    private ProductDao productDao;
21
22    private ObjectMapper objectMapper = new ObjectMapper();
23
24④ @PostMapping("order")
25    public ResponseEntity customerOrder(@RequestParam("userId") int userId) throws JsonProcessingException {
26
27        System.out.println("request came for ORDER FOR CUSTOMER ID : " + userId);
28
29        String orderId = Helper.getAlphaNumericOrderId();
30
31        List<Cart> userCarts = cartDao.findByUser_id(userId);
32
33        LocalDateTime currentDateTime = LocalDateTime.now();
34        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
35        String formatDate = currentDateTime.format(formatter);
36
37        for (Cart cart : userCarts) {
38
39            Orders order = new Orders();
40            order.setOrderId(orderId);
41            order.setUser(cart.getUser());
42            order.setProduct(cart.getProduct());
43            order.setQuantity(cart.getQuantity());
44            order.setOrderDate(formatDate);
45            order.setDeliveryDate(DeliveryStatus.PENDING.value());
46
47            order.setDeliveryStatus(DeliveryStatus.PENDING.value());
48            order.setDeliveryTime(DeliveryTime.DEFAULT.value());
49            order.setDeliveryAssigned(IsDeliveryAssigned.NO.value());
50
51            orderDao.save(order);
52            cartDao.delete(cart);
53        }
54
55        System.out.println("response sent!!!");
56
57        return new ResponseEntity("ORDER SUCCESS", HttpStatus.OK);
58    }
59
60    @GetMapping("myorder")
61    public ResponseEntity getMyOrder(@RequestParam("userId") int userId) throws JsonProcessingException {
62
63        System.out.println("request came for MY ORDER for USER ID : " + userId);
64
65        List<Orders> userOrder = orderDao.findByUser_id(userId);
66
67        OrderDataResponse orderResponse = new OrderDataResponse();
68
69        List<MyOrderResponse> orderDatas = new ArrayList<>();
70
71        for (Orders order : userOrder) {
72            MyOrderResponse orderData = new MyOrderResponse();
73            orderData.setOrderId(order.getOrderId());
74            orderData.setProductDescription(order.getProduct().getDescription());
75            orderData.setProductName(order.getProduct().getTitle());
76            orderData.setProductImage(order.getProduct().getImageName());
77            orderData.setQuantity(order.getQuantity());
78            orderData.setOrderDate(order.getOrderDate());
79            orderData.setProductId(order.getProduct().getId());
80            orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
81            orderData.setDeliveryStatus(order.getDeliveryStatus());
82            orderData.setTotalPrice(
83                String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toString())));
84            if (order.getDeliveryPersonId() == 0) {
85                orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
86                orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
87            }
88        }
89
90    }
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118

```

```

118
119
120
121     User deliveryPerson = null;
122
123     Optional<User> optionalDeliveryPerson = this.userDao.findById(order.getDeliveryPersonId());
124
125     deliveryPerson = optionalDeliveryPerson.get();
126
127     orderData.setDeliveryPersonContact(deliveryPerson.getPhoneNo());
128     orderData.setDeliveryPersonName(deliveryPerson.getFirstName());
129 }
130 orderDatas.add(orderData);
131
132 String json = objectMapper.writeValueAsString(orderDatas);
133
134 System.out.println(json);
135
136 return new ResponseEntity(orderDatas, HttpStatus.OK);
137
138 }
139
140
141 @GetMapping("admin/allorder")
142 public ResponseEntity getAllOrder() throws JsonProcessingException {
143
144     System.out.println("request came for FETCH ALL ORDERS");
145
146     List<Orders> userOrder = orderDao.findAll();
147
148     OrderDataResponse orderResponse = new OrderDataResponse();
149
150     List<MyOrderResponse> orderDatas = new ArrayList<>();
151
152     for (Orders order : userOrder) {
153         MyOrderResponse orderData = new MyOrderResponse();
154         orderData.setOrderId(order.getOrderId());
155         orderData.setProductDescription(order.getProduct().getDescription());
156         orderData.setProductName(order.getProduct().getTitle());
157         orderData.setProductImage(order.getProduct().getImageName());
158         orderData.setQuantity(order.getQuantity());
159         orderData.setOrderDate(order.getDate());
160         orderData.setProductId(order.getProduct().getId());
161         orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
162
163         orderData.setDeliveryStatus(order.getDeliveryStatus());
164         orderData.setTotalPrice(
165             String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice())));
166         orderData.setUserId(order.getUser().getId());
167         orderData.setUserName(order.getUser().getFirstName() + " " + order.getUser().getLastName());
168         orderData.setUserPhone(order.getUser().getPhoneNo());
169         orderData.setAddress(order.getUser().getAddress());
170         if (order.getDeliveryPersonId() == 0) {
171             orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
172             orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
173         }
174     else {
175         User deliveryPerson = null;
176
177         Optional<User> optionalDeliveryPerson = this.userDao.findById(order.getDeliveryPersonId());
178
179         deliveryPerson = optionalDeliveryPerson.get();
180
181         orderData.setDeliveryPersonContact(deliveryPerson.getPhoneNo());
182         orderData.setDeliveryPersonName(deliveryPerson.getFirstName());
183     }
184     orderDatas.add(orderData);
185
186 }
187
188 String json = objectMapper.writeValueAsString(orderDatas);
189
190 System.out.println(json);
191
192 System.out.println("response sent !!!");
193
194 return new ResponseEntity(orderDatas, HttpStatus.OK);
195
196 }
197
198 @GetMapping("admin/showorder")
199 public ResponseEntity getOrdersByOrderId(@RequestParam("orderId") String orderId) throws JsonProcessingException {
200
201     System.out.println("request came for FETCH ORDERS BY ORDER ID : " + orderId);
202
203     List<Orders> userOrder = orderDao.findByOrderId(orderId);
204
205     List<MyOrderResponse> orderDatas = new ArrayList<>();

```

```

206
207     for (Orders order : userOrder) {
208         MyOrderResponse orderData = new MyOrderResponse();
209         orderData.setOrderId(order.getOrderId());
210         orderData.setProductDescription(order.getProduct().getDescription());
211         orderData.setProductName(order.getProduct().getTitle());
212         orderData.setProductImage(order.getProduct().getImageName());
213         orderData.setQuantity(order.getQuantity());
214         orderData.setOrderDate(order.getOrderDate());
215         orderData.setProductId(order.getProduct().getId());
216         orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
217         orderData.setDeliveryStatus(order.getDeliveryStatus());
218         orderData.setTotalPrice(
219             String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toString())));
220         orderData.setUserId(order.getUser().getId());
221         orderData.setUserName(order.getUser().getFirstName() + " " + order.getUser().getLastName());
222         orderData.setUserPhone(order.getUser().getPhoneNo());
223         orderData.setAddress(order.getUser().getAddress());
224         if (order.getDeliveryPersonId() == 0) {
225             orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
226             orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
227         }
228     }
229     else {
230         User deliveryPerson = null;
231         Optional<User> optionalDeliveryPerson = this.userDao.findById(order.getDeliveryPersonId());
232         deliveryPerson = optionalDeliveryPerson.get();
233         orderData.setDeliveryPersonContact(deliveryPerson.getPhoneNo());
234         orderData.setDeliveryPersonName(deliveryPerson.getFirstName());
235     }
236     orderDatas.add(orderData);
237 }
238
239
240     }
241
242
243     String json = objectMapper.writeValueAsString(orderDatas);
244
245     System.out.println(json);
246
247     System.out.println("response sent !!!");
248
249     return new ResponseEntity(orderDatas, HttpStatus.OK);
250
251 }
252
253 @PostMapping("admin/order/deliveryStatus/update")
254 public ResponseEntity updateOrderDeliveryStatus(@RequestBody UpdateDeliveryStatusRequest deliveryRequest)
255     throws JsonProcessingException {
256
257     System.out.println("response came for UPDATE DELIVERY STATUS");
258
259     System.out.println(deliveryRequest);
260
261     List<Orders> orders = orderDao.find by OrderId(deliveryRequest.getOrderId());
262
263     for (Orders order : orders) {
264         order.setDeliveryDate(deliveryRequest.getDeliveryDate());
265         order.setDeliveryStatus(deliveryRequest.getDeliveryStatus());
266         order.setDeliveryTime(deliveryRequest.getDeliveryTime());
267         orderDao.save(order);
268     }
269
270     List<Orders> userOrder = orderDao.find by OrderId(deliveryRequest.getOrderId());
271
272     List<MyOrderResponse> orderDatas = new ArrayList<>();
273
274     for (Orders order : userOrder) {
275         MyOrderResponse orderData = new MyOrderResponse();
276         orderData.setOrderId(order.getOrderId());
277         orderData.setProductDescription(order.getProduct().getDescription());
278         orderData.setProductName(order.getProduct().getTitle());
279         orderData.setProductImage(order.getProduct().getImageName());
280         orderData.setQuantity(order.getQuantity());
281         orderData.setOrderDate(order.getOrderDate());
282         orderData.setProductId(order.getProduct().getId());
283         orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
284         orderData.setDeliveryStatus(order.getDeliveryStatus());
285         orderData.setTotalPrice(
286             String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toString())));
287         orderData.setUserId(order.getUser().getId());
288         orderData.setUserName(order.getUser().getFirstName() + " " + order.getUser().getLastName());
289         orderData.setUserPhone(order.getUser().getPhoneNo());
290         orderData.setAddress(order.getUser().getAddress());
291         if (order.getDeliveryPersonId() == 0) {
292             orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
293             orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
294         }
295     }

```

```

295
296     else {
297         User deliveryPerson = null;
298
299         Optional<User> optionalDeliveryPerson = this.userDao.findById(order.getDeliveryPersonId());
300
301         deliveryPerson = optionalDeliveryPerson.get();
302
303         orderData.setDeliveryPersonContact(deliveryPerson.getPhoneNo());
304         orderData.setDeliveryPersonName(deliveryPerson.getFirstName());
305     }
306     orderDatas.add(orderData);
307
308 }
309
310 String orderJson = objectMapper.writeValueAsString(orderDatas);
311
312 System.out.println(orderJson);
313
314 System.out.println("response sent !!!");
315
316 return new ResponseEntity(orderDatas, HttpStatus.OK);
}
317
318
319 @PostMapping("admin/order/assignDelivery")
320 public ResponseEntity assignDeliveryPersonForOrder(@RequestBody UpdateDeliveryStatusRequest deliveryRequest)
321     throws JsonProcessingException {
322
323     System.out.println("response came for ASSIGN DELIVERY PERSON FOR ORDERS");
324
325     System.out.println(deliveryRequest);
326
327     List<Orders> orders = orderDao.findByOrderId(deliveryRequest.getOrderId());
328
329     User deliveryPerson = null;
330
331     Optional<User> optionalDeliveryPerson = this.userDao.findById(deliveryRequest.getDeliveryId());
332
333     if (optionalDeliveryPerson.isPresent()) {
334         deliveryPerson = optionalDeliveryPerson.get();
335     }
336
337     for (Orders order : orders) {
338         order.setDeliveryAssigned(IsDeliveryAssigned.YES.value());
339         order.setDeliveryPersonId(deliveryRequest.getDeliveryId());
340         orderDao.save(order);
341     }
342
343     List<Orders> userOrder = orderDao.findByOrderId(deliveryRequest.getOrderId());
344
345     List<MyOrderResponse> orderDatas = new ArrayList<>();
346
347     for (Orders order : userOrder) {
348         MyOrderResponse orderData = new MyOrderResponse();
349         orderData.setOrderId(order.getOrderId());
350         orderData.setProductDescription(order.getProduct().getDescription());
351         orderData.setProductName(order.getProduct().getTitle());
352         orderData.setProductImage(order.getProduct().getImageName());
353         orderData.setQuantity(order.getQuantity());
354         orderData.setOrderDate(order.getOrderDate());
355         orderData.setProductId(order.getProduct().getId());
356         orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
357         orderData.setDeliveryStatus(order.getDeliveryStatus());
358         orderData.setTotalPrice(
359             String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toString())));
360         orderData.setUserId(order.getUser().getId());
361         orderData.setUserName(order.getUser().getFirstName() + " " + order.getUser().getLastName());
362         orderData.setUserPhone(order.getUser().getPhoneNo());
363         orderData.setAddress(order.getUser().getAddress());
364
365         if (order.getDeliveryPersonId() == 0) {
366             orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
367             orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
368         }
369
370     else {
371         User dPerson = null;
372
373         Optional<User> optionalPerson = this.userDao.findById(order.getDeliveryPersonId());
374
375         dPerson = optionalPerson.get();
376
377         orderData.setDeliveryPersonContact(dPerson.getPhoneNo());
378         orderData.setDeliveryPersonName(dPerson.getFirstName());
379     }
380

```

```

380         orderDatas.add(orderData);
381     }
382   }
383
384   String orderJson = objectMapper.writeValueAsString(orderDatas);
385
386   System.out.println(orderJson);
387
388   System.out.println("response sent !!!");
389
390   return new ResponseEntity(orderDatas, HttpStatus.OK);
391 }
392
393
394 @GetMapping("delivery/myorder")
395 public ResponseEntity getMyDeliveryOrders(@RequestParam("deliveryPersonId") int deliveryPersonId)
396   throws JsonProcessingException {
397
398   System.out.println("request came for MY DELIVERY ORDER for USER ID : " + deliveryPersonId);
399
400   User person = null;
401
402   Optional<User> oD = this.userDao.findById(deliveryPersonId);
403
404   if (oD.isPresent()) {
405     person = oD.get();
406   }
407
408   List<Orders> userOrder = orderDao.findByDeliveryPersonId(deliveryPersonId);
409
410   List<MyOrderResponse> orderDatas = new ArrayList<>();
411
412   for (Orders order : userOrder) {
413     MyOrderResponse orderData = new MyOrderResponse();
414     orderData.setOrderId(order.getOrderId());
415     orderData.setProductDescription(order.getProduct().getDescription());
416     orderData.setProductName(order.getProduct().getTitle());
417     orderData.setProductImage(order.getProduct().getImageName());
418     orderData.setQuantity(order.getQuantity());
419     orderData.setOrderDate(order.getOrderDate());
420     orderData.setProductId(order.getProduct().getId());
421     orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
422     orderData.setDeliveryStatus(order.getDeliveryStatus());
423     orderData.setTotalPrice(
424       String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toString())));
425     orderData.setUserId(order.getUser().getId());
426     orderData.setUserName(order.getUser().getFirstName() + " " + order.getUser().getLastName());
427     orderData.setUserPhone(order.getUser().getPhoneNo());
428     orderData.setAddress(order.getUser().getAddress());
429
430     if (order.getDeliveryPersonId() == 0) {
431       orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
432       orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
433     }
434
435     else {
436       orderData.setDeliveryPersonContact(person.getPhoneNo());
437       orderData.setDeliveryPersonName(person.getFirstName());
438     }
439
440     orderDatas.add(orderData);
441   }
442
443
444   String json = objectMapper.writeValueAsString(orderDatas);
445
446   System.out.println(json);
447
448   return new ResponseEntity(orderDatas, HttpStatus.OK);
449 }
450
451
452 }

```

Figure:48

Product Controller

A Product Controller in a Spring Boot application. This controller will handle operations related to products.

```

1 package com.onlineshopping.controller;
2
3④ import java.io.IOException;[]
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30 @RestController
31 @RequestMapping("api/product")
32 @CrossOrigin(origins = "http://localhost:3000")
33 public class ProductController {
34
35④     @Autowired
36     private ProductService productService;
37
38④     @Autowired
39     private ProductDao productDao;
40
41④     @Autowired
42     private CategoryDao categoryDao;
43
44④     @Autowired
45     private StorageService storageService;
46
47④     @Autowired
48     private UserDao userDao;
49
50④     @PostMapping("add")
51     public ResponseEntity<?> addProduct(ProductAddRequest productDto) {
52         System.out.println("recieved request for ADD PRODUCT");
53         System.out.println(productDto);
54         Product product=ProductAddRequest.toEntity(productDto);
55
56         Optional<Category> optional = categoryDao.findById(productDto.getCategoryId());
57         Category category = null;
58         if(optional.isPresent()) {
59             category = optional.get();
60         }
61
62         product.setCategory(category);
63         productService.addProduct(product, productDto.getImage());
64
65         System.out.println("response sent!!!!");
66         return ResponseEntity.ok(product);
67     }
68
69
70④     @GetMapping("all")
71
72④     @GetMapping("all")
73     public ResponseEntity<?> getAllProducts() {
74
75         System.out.println("request came for getting all products");
76
77         List<Product> products = new ArrayList<Product>();
78
79         products = productDao.findAll();
80
81         System.out.println("response sent!!!!");
82
83         return ResponseEntity.ok(products);
84     }
85④     @GetMapping("id")
86     public ResponseEntity<?> getProductById(@RequestParam("productId") int productId) {
87
88         System.out.println("request came for getting Product by Product Id");
89
90         Product product = new Product();
91
92         Optional<Product> optional = productDao.findById(productId);
93
94         if(optional.isPresent()) {
95             product = optional.get();
96         }
97         System.out.println("response sent!!!!");
98
99         return ResponseEntity.ok(product);
100    }
101
102
103④     @GetMapping("category")
104     public ResponseEntity<?> getProductsByCategories(@RequestParam("categoryId") int categoryId) {
105
106         System.out.println("request came for getting all products by category");
107
108         List<Product> products = new ArrayList<Product>();
109
110         products = productDao.findByCategoryId(categoryId);
111
112         System.out.println("response sent!!!!");
113

```

```

    }
    return ResponseEntity.ok(products);
}

}

@GetMapping(value="/{productImageName}", produces = "image/*")
public void fetchProductImage(@PathVariable("productImageName") String productImageName, HttpServletResponse resp) {
    System.out.println("request came for fetching product pic");
    System.out.println("Loading file: " + productImageName);
    Resource resource = storageService.load(productImageName);
    if(resource != null) {
        try(InputStream in = resource.getInputStream()) {
            ServletOutputStream out = resp.getOutputStream();
            FileCopyUtils.copy(in, out);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    System.out.println("response sent!");
}
}
}
}

```

Figure:49

User Controller:

A User Controller in a Spring Boot application. This controller will handle operations related to user management.

```

1 package com.onlineshopping.controller;
2
3 import java.util.List;
4
5 @RestController
6 @RequestMapping("api/user")
7 @CrossOrigin(origins = "http://localhost:3000")
8 public class UserController {
9
10    @Autowired
11    private UserDao userDao;
12
13    @Autowired
14    private AddressDao addressDao;
15
16    @PostMapping("register")
17    public ResponseEntity<?> registerUser(@RequestBody AddUserRequest userRequest) {
18        System.out.println("recieved request for REGISTER USER");
19        System.out.println(userRequest);
20
21        Address address = new Address();
22        address.setCity(userRequest.getCity());
23        address.setPincode(userRequest.getPincode());
24        address.setStreet(userRequest.getStreet());
25
26        Address addAddress = addressDao.save(address);
27
28        User user = new User();
29        user.setAddress(addAddress);
30        user.setEmailId(userRequest.getEmailId());
31        user.setFirstName(userRequest.getFirstName());
32        user.setLastName(userRequest.getLastName());
33        user.setPhoneNo(userRequest.getPhoneNo());
34        user.setPassword(userRequest.getPassword());
35        user.setRole(userRequest.getRole());
36        User addUser = userDao.save(user);
37
38        System.out.println("response sent!!!!");
39        return ResponseEntity.ok(addUser);
40    }
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

```

```

56     @PostMapping("login")
57     public ResponseEntity<?> loginUser(@RequestBody UserLoginRequest loginRequest) {
58         System.out.println("recieved request for LOGIN USER");
59         System.out.println(loginRequest);
60
61         User user = new User();
62         user = userDao.findByEmailIdAndPasswordAndRole(loginRequest.getEmailId(), loginRequest.getPassword(), loginRequest.getRole());
63
64         System.out.println("response sent!!!");
65         return ResponseEntity.ok(user);
66     }
67
68
69     @GetMapping("deliveryperson/all")
70     public ResponseEntity<?> getAllDeliveryPersons() {
71         System.out.println("recieved request for getting ALL Delivery Persons!!!");
72
73         List<User> deliveryPersons = this.userDao.findByRole("Delivery");
74
75         System.out.println("response sent!!!");
76         return ResponseEntity.ok(deliveryPersons);
77     }
78
79 }
80

```

Figure:50

5.2.3 Data Access Object (Dao):

A DAO typically provides an abstract interface to some type of database or other persistence mechanism.

AddressDao:

```

1 package com.onlineshopping.dao;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface AddressDao extends JpaRepository<Address, Integer> {
7
8 }
9
10
11

```

Figure:51

Cartdao:

```

1 package com.onlineshopping.dao;
2
3 import java.util.List;
4
5 @Repository
6 public interface CartDao extends JpaRepository<Cart, Integer> {
7
8     List<Cart> findByUser_id(int userId);
9
10 }
11
12
13
14
15

```

Figure:52

Categorydao:

```

1 package com.onlineshopping.dao;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6 public interface CategoryDao extends JpaRepository<Category, Integer> {
7
8
9 }
10
11
12
13
14
15

```

Figure:53

Orderdao:

```
1 package com.onlineshopping.dao;
2
3+ import java.util.List;[]
4
5 @Repository
6 public interface OrderDao extends JpaRepository<Orders, Integer> {
7
8     List<Orders> findByUser_id(int userId);
9     List<Orders> findByOrderId(String orderId);
10    List<Orders> findByUser_idAndProduct_id(int userId, int productId);
11    List<Orders> findByUser(User user);
12    List<Orders> findByDeliveryPersonId(int deliveryPersonId);
13
14 }
15
16
17
18 }
```

Figure:54

Productdao:

```
1 package com.onlineshopping.dao;
2
3+ import java.util.List;[]
4
5 @Repository
6 public interface ProductDao extends JpaRepository<Product, Integer> {
7
8     List<Product> findByCategoryId(int category);
9
10 }
11
12
13
14
15
16
```

Figure:55

Userdao:

```
1 package com.onlineshopping.dao;
2
3+ import java.util.List;[]
4
5 @Repository
6 public interface UserDao extends JpaRepository<User, Integer> {
7
8     User findByEmailIdAndPasswordAndRole(String emailId, String password, String role);
9     List<User> findByRole(String role);
10
11 }
12
13
14
15
16
```

Figure:56

5.2.4 Data Transfer Object(dto)

A DTO (Data Transfer Object) is a plain JavaScript object that is used to transfer data between different parts of your application. DTOs are often used to represent the shape of data that is exchanged between a frontend component and a backend API.

AddToCartRequest:

```
1 package com.onlineshopping.dto;
2
3 public class AddToCartRequest {
4
5     private int productId;
6
7     private int quantity;
8
9     private int userId;
10
11    public int getProductId() {
12        return productId;
13    }
14
15    public void setProductId(int productId) {
16        this.productId = productId;
17    }
18
19    public int getQuantity() {
20        return quantity;
21    }
22
23    public void setQuantity(int quantity) {
24        this.quantity = quantity;
25    }
26
27    public int getUserId() {
28        return userId;
29    }
30
31    public void setUserId(int userId) {
32        this.userId = userId;
33    }
34
35    @Override
36    public String toString() {
37        return "AddToCartRequest [productId=" + productId + ", quantity=" + quantity + ", userid=" + userId + "]";
38    }
39
40
41
42}
43
```

Figure:57

AddUserRequest:

```
1 package com.onlineshopping.dto;
2
3 public class AddUserRequest {
4
5     private String firstName;
6
7     private String lastName;
8
9     private String emailId;
10
11    private String password;
12
13    private String phoneNo;
14
15    private String street;
16
17    private String city;
18
19    private int pincode;
20
21    private String role;
22
23    public String getFirstName() {
24        return firstName;
25    }
26
27    public void setFirstName(String firstName) {
28        this.firstName = firstName;
29    }
30
31    public String getLastname() {
32        return lastName;
33    }
34
35    public void setLastName(String lastName) {
36        this.lastName = lastName;
37    }
38}
```

```

38
39@     public String getEmailId() {
40         return emailId;
41     }
42
43@     public void setEmailId(String emailId) {
44         this.emailId = emailId;
45     }
46
47@     public String getPassword() {
48         return password;
49     }
50
51@     public void setPassword(String password) {
52         this.password = password;
53     }
54
55@     public String getPhoneNo() {
56         return phoneNo;
57     }
58
59@     public void setPhoneNo(String phoneNo) {
60         this.phoneNo = phoneNo;
61     }
62
63@     public String getStreet() {
64         return street;
65     }
66
67@     public void setStreet(String street) {
68         this.street = street;
69     }
70
71@     public String getCity() {
72         return city;
73     }
74
75@     public void setCity(String city) {
76         this.city = city;
77     }
78
79@     public int getPincode() {
80         return pincode;
81     }
82
83@     public void setPincode(int pincode) {
84         this.pincode = pincode;
85     }
86
87@     public String getRole() {
88         return role;
89     }
90
91@     public void setRole(String role) {
92         this.role = role;
93     }
94
95@     @Override
96     public String toString() {
97         return "AddUserRequest [firstName=" + firstName + ", lastName=" + lastName + ", emailId=" + emailId
98             + ", password=" + password + ", phoneNo=" + phoneNo + ", street=" + street + ", city=" + city
99             + ", pincode=" + pincode + "]";
100    }
101
102}
103

```

Figure:58

CartDataResponse:

```

1 package com.onlineshopping.dto;
2
3 public class CartDataResponse {
4
5     private int cartId;
6
7     private int productId;
8
9     private String productName;
10
11    private String productDescription;
12
13    private String productImage;
14
15    private int quantity;
16
17    public int getCartId() {
18        return cartId;
19    }
20
21    public void setCartId(int cartId) {
22        this.cartId = cartId;
23    }
24
25    public int getProductId() {
26        return productId;
27    }
28
29    public void setProductId(int productId) {
30        this.productId = productId;
31    }
32
33    public String getProductName() {
34        return productName;
35    }
36
37    public void setProductName(String productName) {
38        this.productName = productName;
39    }
40
41
42    public String getProductDescription() {
43        return productDescription;
44    }
45
46    public void setProductDescription(String productDescription) {
47        this.productDescription = productDescription;
48    }
49
50
51    public int getQuantity() {
52        return quantity;
53    }
54
55    public void setQuantity(int quantity) {
56        this.quantity = quantity;
57    }
58
59    public String getProductImage() {
60        return productImage;
61    }
62
63    public void setProductImage(String productImage) {
64        this.productImage = productImage;
65    }
66
67    @Override
68    public String toString() {
69        return "CartDataResponse [cartId=" + cartId + ", productId=" + productId + ", productName=" + productName
70                + ", productDescription=" + productDescription + ", productImage=" + productImage + ", quantity="
71                + quantity + "]";
72    }
73}

```

Figure:59

Cart Response:

```

1 package com.onlineshopping.dto;
2
3 import java.util.List;
4
5 public class CartResponse {
6
7     private String totalCartPrice;
8
9     private List<CartDataResponse> cartData;
10
11    public String getTotalCartPrice() {
12        return totalCartPrice;
13    }
14
15    public void setTotalCartPrice(String totalCartPrice) {
16        this.totalCartPrice = totalCartPrice;
17    }
18
19    public List<CartDataResponse> getCartData() {
20        return cartData;
21    }
22
23    public void setCartData(List<CartDataResponse> cartData) {
24        this.cartData = cartData;
25    }
26
27    @Override
28    public String toString() {
29        return "CartResponse [totalCartPrice=" + totalCartPrice + ", cartData=" + cartData + "]";
30    }
31
32}
33

```

Figure:60

MyOrderResponse:

```

1 package com.onlineshopping.dto;
2
3 import com.onlineshopping.model.Address;
4
5 public class MyOrderResponse {
6
7     private String orderId;
8
9     private int productId;
10
11    private int userId;
12
13    private String userName; // first name + last name
14
15    private Address address;
16
17    private String userPhone;
18
19    private String productName;
20
21    private String productDescription;
22
23    private String productImage;
24
25    private int quantity;
26
27    private String totalPrice;
28
29    private String orderDate;
30
31    private String deliveryDate;
32
33    private String deliveryStatus;
34
35    private String deliveryPersonName;
36
37    private String deliveryPersonContact;
38
39    public String getOrderId() {
40        return orderId;
41    }
42

```

```

42
43@    public void setOrderId(String orderId) {
44        this.orderId = orderId;
45    }
46
47@    public int getProductId() {
48        return productId;
49    }
50
51@    public void setProductId(int productId) {
52        this.productId = productId;
53    }
54
55@    public String getProductName() {
56        return productName;
57    }
58
59@    public void setProductName(String productName) {
60        this.productName = productName;
61    }
62
63@    public String getProductDescription() {
64        return productDescription;
65    }
66
67@    public void setProductDescription(String productDescription) {
68        this.productDescription = productDescription;
69    }
70
71@    public String getProductImage() {
72        return productImage;
73    }
74
75@    public void setProductImage(String productImage) {
76        this.productImage = productImage;
77    }
78
79@    public int getQuantity() {
80        return quantity;
81    }
82
83@    public void setQuantity(int quantity) {
84        this.quantity = quantity;
85    }
86
87@    public String getOrderDate() {
88        return orderDate;
89    }
90
91@    public void setOrderDate(String orderDate) {
92        this.orderDate = orderDate;
93    }
94
95@    public String getDeliveryDate() {
96        return deliveryDate;
97    }
98
99@    public void setDeliveryDate(String deliveryDate) {
100       this.deliveryDate = deliveryDate;
101    }
102
103@    public String getDeliveryStatus() {
104        return deliveryStatus;
105    }
106
107@    public void setDeliveryStatus(String deliveryStatus) {
108        this.deliveryStatus = deliveryStatus;
109    }
110
111@    public String getTotalPrice() {
112        return totalPrice;
113    }
114
115@    public void setTotalPrice(String totalPrice) {
116        this.totalPrice = totalPrice;
117    }
118
119@    public int getUserId() {
120        return userId;
121    }
122

```

```

122
123@ public void setUserId(int userId) {
124     this.userId = userId;
125 }
126@ public String getUserName() {
127     return userName;
128 }
129@ public void setUserName(String userName) {
130     this.userName = userName;
131 }
132
133@ public Address getAddress() {
134     return address;
135 }
136
137@ public void setAddress(Address address) {
138     this.address = address;
139 }
140
141@ public String getUserPhone() {
142     return userPhone;
143 }
144
145@ public void setUserPhone(String userPhone) {
146     this.userPhone = userPhone;
147 }
148
149@ public String getDeliveryPersonName() {
150     return deliveryPersonName;
151 }
152
153@ public void setDeliveryPersonName(String deliveryPersonName) {
154     this.deliveryPersonName = deliveryPersonName;
155 }
156@ public String getDeliveryPersonContact() {
157     return deliveryPersonContact;
158 }
159@ public void setDeliveryPersonContact(String deliveryPersonContact) {
160     this.deliveryPersonContact = deliveryPersonContact;
161 }
162
163 }
164

```

Figure:61

OrderDataResponse:

```

1 package com.onlineshopping.dto;
2
3 import java.util.List;
4
5 public class OrderDataResponse {
6
7     private List<MyOrderResponse> orderResponse;
8
9     private String totalCartPrice;
10
11@ public List<MyOrderResponse> getOrderResponse() {
12     return orderResponse;
13 }
14
15@ public void setOrderResponse(List<MyOrderResponse> orderResponse) {
16     this.orderResponse = orderResponse;
17 }
18
19@ public String getTotalCartPrice() {
20     return totalCartPrice;
21 }
22
23@ public void setTotalCartPrice(String totalCartPrice) {
24     this.totalCartPrice = totalCartPrice;
25 }
26
27@ Override
28 public String toString() {
29     return "OrderDataResponse [orderResponse=" + orderResponse + ", totalCartPrice=" + totalCartPrice + "]";
30 }
31
32 }
33

```

Figure:62

ProductAddRequest:

```
1 package com.onlineshopping.dto;
2
3+ import java.math.BigDecimal;
4
5 public class ProductAddRequest {
6
7     private int id;
8     private String title;
9     private String description;
10    private int quantity;
11    private BigDecimal price;
12    private int categoryId;
13    private MultipartFile image;
14
15    public int getId() {
16        return id;
17    }
18    public void setId(int id) {
19        this.id = id;
20    }
21    public String getTitle() {
22        return title;
23    }
24    public void setTitle(String title) {
25        this.title = title;
26    }
27    public String getDescription() {
28        return description;
29    }
30    public void setDescription(String description) {
31        this.description = description;
32    }
33    public void setQuantity(int quantity) {
34        this.quantity = quantity;
35    }
36    public int getQuantity() {
37        return quantity;
38    }
39    public void setPrice(BigDecimal price) {
40        this.price = price;
41    }
42    public BigDecimal getPrice() {
43        return price;
44    }
45    public void setImage(MultipartFile image) {
46        this.image = image;
47    }
48    public MultipartFile getImage() {
49        return image;
50    }
51    public void setCategoryId(int categoryId) {
52        this.categoryId = categoryId;
53    }
54    public int getCategoryId() {
55        return categoryId;
56    }
57    public void toEntity(ProductAddRequest dto) {
58        Product entity=new Product();
59        BeanUtils.copyProperties(dto, entity, "image", "categoryId");
60        return entity;
61    }
62    @Override
63    public String toString() {
64        return "ProductAddRequest [id=" + id + ", title=" + title + ", description=" + description + ", quantity=" +
65        + quantity + ", price=" + price + ", categoryId=" + categoryId + "]";
66    }
67}
68}
69}
70}
71}
72}
```

Figure:63

UserLoginRequest:

```
1 package com.onlineshopping.dto;
2
3 public class UserLoginRequest {
4
5     private String emailId;
6     private String password;
7     private String role;
8
9     public String getEmailId() {
10         return emailId;
11     }
12     public void setEmailId(String emailId) {
13         this.emailId = emailId;
14     }
15     public String getPassword() {
16         return password;
17     }
18     public void setPassword(String password) {
19         this.password = password;
20     }
21
22     public String getRole() {
23         return role;
24     }
25     public void setRole(String role) {
26         this.role = role;
27     }
28     @Override
29     public String toString() {
30         return "UserLoginRequest [emailId=" + emailId + ", password=" + password + "]";
31     }
32
33 }
34
```

Figure:64

UserResponse:

```
1 package com.onlineshopping.dto;
2
3 public class UserResponse {
4
5 }
6
```

Figure:65

5.2.5 Model

Address:

The Address model could include fields such as street, city, state, pin code, etc.

the Address Form component uses the Address model to manage the state of the form fields. The form fields are connected to the state of the Address object, and when the form is submitted, you can perform actions with the Address object, such as sending it to an API or processing it further in your application. Adjust the fields and structure based on your specific requirements.

```

1 package com.onlineshopping.model;
2
3+ import javax.persistence.Entity;□
4
5 @Entity
6 public class Address {
7
8     @Id
9     @GeneratedValue(strategy=GenerationType.IDENTITY)
10    private int id;
11
12    private String street;
13
14    private String city;
15
16    private int pincode;
17
18    public int getId() {
19        return id;
20    }
21
22    public void setId(int id) {
23        this.id = id;
24    }
25
26    public String getStreet() {
27        return street;
28    }
29
30    public void setStreet(String street) {
31        this.street = street;
32    }
33
34    public String getCity() {
35        return city;
36    }
37
38    public void setCity(String city) {
39        this.city = city;
40    }
41
42    public int getPincode() {
43        return pincode;
44    }
45
46    public void setPincode(int pincode) {
47        this.pincode = pincode;
48    }
49
50    @Override
51    public String toString() {
52        return "Address [id=" + id + ", street=" + street + ", city=" + city + ", pincode=" + pincode + "]";
53    }
54
55}
56
57
58}
59

```

Figure:66

Cart Model:

The Cart model can include fields such as items, total price, and other relevant information.

```

1 package com.onlineshopping.model;
2
3+ import javax.persistence.Entity;□
4
5 @Entity
6 public class Cart {
7
8     @Id
9     @GeneratedValue(strategy=GenerationType.IDENTITY)
10    private int id;
11
12    @OneToOne
13    @JoinColumn(name = "product_id")
14    private Product product;
15
16    @OneToOne
17    @JoinColumn(name = "user_id")
18    private User user;
19
20    private int quantity;
21
22    public int getId() {
23        return id;
24    }
25
26    public void setId(int id) {
27        this.id = id;
28    }
29
30    public Product getProduct() {
31        return product;
32    }
33
34    public void setProduct(Product product) {
35        this.product = product;
36    }
37
38    public User getUser() {
39        return user;
40    }
41
42    public void setUser(User user) {
43        this.user = user;
44    }
45
46    public int getQuantity() {
47        return quantity;
48    }
49
50    public void setQuantity(int quantity) {
51        this.quantity = quantity;
52    }
53
54    public double getTotalPrice() {
55        return totalPrice;
56    }
57
58    public void setTotalPrice(double totalPrice) {
59        this.totalPrice = totalPrice;
60    }
61
62    public void calculateTotalPrice() {
63        totalPrice = product.getPrice() * quantity;
64    }
65
66    private double totalPrice;
67
68}
69

```

```

34 |
35@     public Product getProduct() {
36         return product;
37     }
38
39@     public void setProduct(Product product) {
40         this.product = product;
41     }
42
43@     public int getQuantity() {
44         return quantity;
45     }
46
47@     public void setQuantity(int quantity) {
48         this.quantity = quantity;
49     }
50
51@     public User getUser() {
52         return user;
53     }
54
55@     public void setUser(User user) {
56         this.user = user;
57     }
58
59 }
60

```

Figure:67

Category Model:

A simple Category model in JavaScript for a React application. The Category model can include fields such as title, description, and other relevant information.

Category is a class representing a product category.

The class has fields like title and description.

The constructor is used to initialize the instance of the Category class.

```

1 package com.onlineshopping.model;
2
3@ import javax.persistence.Entity;□
4
5 @Entity
6 public class Category {
7
8     @Id
9     @GeneratedValue(strategy=GenerationType.IDENTITY)
10    private int id;
11
12    private String title;
13
14    private String description;
15
16    public int getId() {
17        return id;
18    }
19
20    public void setId(int id) {
21        this.id = id;
22    }
23
24    public String getTitle() {
25        return title;
26    }
27
28    public void setTitle(String title) {
29        this.title = title;
30    }
31
32    public String getDescription() {
33        return description;
34    }
35
36    public void setDescription(String description) {
37        this.description = description;
38    }
39
40
41
42
43 }
44

```

Figure:68

Order Model:

A simple Order model in JavaScript for a React application. The Order model can include fields such as items, total price, customer information, and other relevant details.

```
1 package com.onlineshopping.model;
2
3+ import javax.persistence.Entity;□
4
5 @Entity
6 public class Orders {
7
8     @Id
9     @GeneratedValue(strategy=GenerationType.IDENTITY)
10    private int id;
11
12    private String orderId;
13
14    @OneToOne
15    @JoinColumn(name = "product_id")
16    private Product product;
17
18    @OneToOne
19    @JoinColumn(name = "user_id")
20    private User user;
21
22    private int quantity;
23
24    private String orderDate; // 13-01-2022 10:00 PM
25
26    private String deliveryStatus;
27
28    private String deliveryDate;
29
30    private String deliveryTime; // evening, afternoon...
31
32    private String deliveryAssigned;
33
34    private int deliveryPersonId;
35
36    public int getId() {
37        return id;
38    }
39
40    public void setId(int id) {
41        this.id = id;
42    }
43
44
45    public String getOrderId() {
46        return orderId;
47    }
48
49    public void setOrderId(String orderId) {
50        this.orderId = orderId;
51    }
52
53    public Product getProduct() {
54        return product;
55    }
56
57    public void setProduct(Product product) {
58        this.product = product;
59    }
60
61    public User getUser() {
62        return user;
63    }
64
65    public void setUser(User user) {
66        this.user = user;
67    }
68
69    public int getQuantity() {
70        return quantity;
71    }
72
73    public void setQuantity(int quantity) {
74        this.quantity = quantity;
75    }
76
77    public String getOrderDate() {
78        return orderDate;
79    }
80
81    public void setOrderDate(String orderDate) {
82        this.orderDate = orderDate;
83    }
84
```

```

84     public void setOrderDate(String orderDate) {
85         this.orderDate = orderDate;
86     }
87     public String getDeliveryStatus() {
88         return deliveryStatus;
89     }
90
91     public void setDeliveryStatus(String deliveryStatus) {
92         this.deliveryStatus = deliveryStatus;
93     }
94     public String getDeliveryDate() {
95         return deliveryDate;
96     }
97
98     public void setDeliveryDate(String deliveryDate) {
99         this.deliveryDate = deliveryDate;
100    }
101
102    public String getDeliveryTime() {
103        return deliveryTime;
104    }
105
106    public void setDeliveryTime(String deliveryTime) {
107        this.deliveryTime = deliveryTime;
108    }
109
110    public int getDeliveryPersonId() {
111        return deliveryPersonId;
112    }
113    public void setDeliveryPersonId(int deliveryPersonId) {
114        this.deliveryPersonId = deliveryPersonId;
115    }
116    public String getDeliveryAssigned() {
117        return deliveryAssigned;
118    }
119    public void setDeliveryAssigned(String deliveryAssigned) {
120        this.deliveryAssigned = deliveryAssigned;
121    }
122
123 }

```

Figure:69

Product Model:

A simple Product model in JavaScript for a React application. The Product model can include fields such as title, price, description, and other relevant information.

Product is a class representing a product.

The class has fields like id, title, price, and description.

The constructor is used to initialize the instance of the Product class.

```

1 package com.onlineshopping.model;
2
3+ import java.math.BigDecimal;□
10
11 @Entity
12 public class Product {
13
14@     @Id
15     @GeneratedValue(strategy=GenerationType.IDENTITY)
16     private int id;
17     private String title;
18     private String description;
19     private int quantity;
20     private BigDecimal price;
21     private String imageName;
22
23@     @ManyToOne
24     @JoinColumn(name ="categoryId")
25     private Category category;
26
27
28@     public int getId() {
29         return id;
30     }
31@     public void setId(int id) {
32         this.id = id;
33     }
34@     public String getTitle() {
35         return title;
36     }
37@     public void setTitle(String title) {
38         this.title = title;
39     }
40@     public String getDescription() {
41         return description;
42     }
43@     public void setDescription(String description) {
44         this.description = description;
45     }
46@     public int getQuantity() {
47         return quantity;
48     }
49@     public void setQuantity(int quantity) {
50         this.quantity = quantity;
51     }
52@     public BigDecimal getPrice() {
53         return price;
54     }
55@     public void setPrice(BigDecimal price) {
56         this.price = price;
57     }
58@     public String getImageName() {
59         return imageName;
60     }
61@     public void setImageName(String imageName) {
62         this.imageName = imageName;
63     }
64@     public Category getCategory() {
65         return category;
66     }
67@     public void setCategory(Category category) {
68         this.category = category;
69     }
70
71 }
72

```

Figure:70

User Model:

A simple User model in JavaScript for a React application. The User model can include fields such as first name, last name, email, password, and other relevant information.

```
1 package com.onlineshopping.model;
2
3+ import javax.persistence.Entity;□
10
11 @Entity
12 public class User {
13
14@     @Id
15     @GeneratedValue(strategy=GenerationType.IDENTITY)
16     private int id;
17
18     private String firstName;
19
20     private String lastName;
21
22     private String emailId;
23
24@     @JsonIgnore
25     private String password;
26
27     private String phoneNo;
28
29     private String role;
30
31@     @OneToOne
32     @JoinColumn(name = "address_id")
33     private Address address;
34
35@     public int getId() {
36         return id;
37     }
38
39@     public void setId(int id) {
40         this.id = id;
41     }
42
43@     public String getFirstName() {
44         return firstName;
45     }
46
47@     public void setFirstName(String firstName) {
48         this.firstName = firstName;
49     }
50
51@     public String getLastName() {
52         return lastName;
53     }
54
55@     public void setLastName(String lastName) {
56         this.lastName = lastName;
57     }
58
59@     public String getEmailId() {
60         return emailId;
61     }
62
63@     public void setEmailId(String emailId) {
64         this.emailId = emailId;
65     }
66
67@     public String getPassword() {
68         return password;
69     }
70
71@     public void setPassword(String password) {
72         this.password = password;
73     }
74
75@     public String getPhoneNo() {
76         return phoneNo;
77     }
78
79@     public void setPhoneNo(String phoneNo) {
80         this.phoneNo = phoneNo;
81     }
82
```

```

82
83     public Address getAddress() {
84         return address;
85     }
86
87     public void setAddress(Address address) {
88         this.address = address;
89     }
90
91     public String getRole() {
92         return role;
93     }
94
95     public void setRole(String role) {
96         this.role = role;
97     }
98
99     @Override
100    public String toString() {
101        return "User [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", emailId=" + emailId
102                + ", password=" + password + ", phoneNo=" + phoneNo + ", address=" + address + "]";
103    }
104
105
106 }
107

```

Figure:71

5.2.6 Service:

Product service:

A Product Service can be used to interact with a backend API to perform operations related to products, such as fetching product data, adding new products, updating existing products, and deleting products.

```

1 package com.onlineshopping.model;
2
3 import javax.persistence.Entity;□
4
5 @Entity
6 public class User {
7
8     @Id
9     @GeneratedValue(strategy=GenerationType.IDENTITY)
10    private int id;
11
12    private String firstName;
13
14    private String lastName;
15
16    private String emailId;
17
18    @JsonIgnore
19    private String password;
20
21    private String phoneNo;
22
23    private String role;
24
25    @OneToOne
26    @JoinColumn(name = "address_id")
27    private Address address;
28
29    public int getId() {
30        return id;
31    }
32
33    public void setId(int id) {
34        this.id = id;
35    }
36
37    public String getFirstName() {
38        return firstName;
39    }
40
41    public void setFirstName(String firstName) {
42        this.firstName = firstName;
43    }
44
45
46
47
48
49
50

```

```

50
51@     public String getLastName() {
52         return lastName;
53     }
54
55@     public void setLastName(String lastName) {
56         this.lastName = lastName;
57     }
58
59@     public String getEmailId() {
60         return emailId;
61     }
62
63@     public void setEmailId(String emailId) {
64         this.emailId = emailId;
65     }
66
67@     public String getPassword() {
68         return password;
69     }
70
71@     public void setPassword(String password) {
72         this.password = password;
73     }
74
75@     public String getPhoneNo() {
76         return phoneNo;
77     }
78
79@     public void setPhoneNo(String phoneNo) {
80         this.phoneNo = phoneNo;
81     }
82
83@     public Address getAddress() {
84         return address;
85     }
86
87@     public void setAddress(Address address) {
88         this.address = address;
89     }
90
91@     public String getRole() {
92         return role;
93     }
94
95@     public void setRole(String role) {
96         this.role = role;
97     }
98
99@     @Override
▲100    public String toString() {
101        return "User [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", emailId=" + emailId
102                + ", password=" + password + ", phoneNo=" + phoneNo + ", address=" + address + "]";
103    }
104
105
106 }
107

```

Figure:72

Product serviceimpl:

```
1 package com.onlineshopping.service;
2
3+ import org.springframework.beans.factory.annotation.Autowired;□
4
5
6 @Service
7 public class ProductServiceImpl implements ProductService {
8
9     @Autowired
10    private ProductDao productDao;
11
12    @Autowired
13    private StorageService storageService;
14
15    @Override
16    public void addProduct(Product product, MultipartFile productImage) {
17        String productName = storageService.store(productImage);
18
19        product.setProductName(productName);
20
21        this.productDao.save(product);
22    }
23
24
25
26
27
28
29
30
31 }
```

Figure:73

5.2.7 Utility:

Header code:

A utility or code related to handling headers in a React application. If you're working with HTTP requests and need to manage headers, you might be interested in handling them within a utility or service.

```
1 package com.onlineshopping.utility;
2
3
4 public class HeaderCode {
5 }
```

Figure:74

Helper:

The Helper Utility class contains static methods for generating unique IDs and formatting dates.

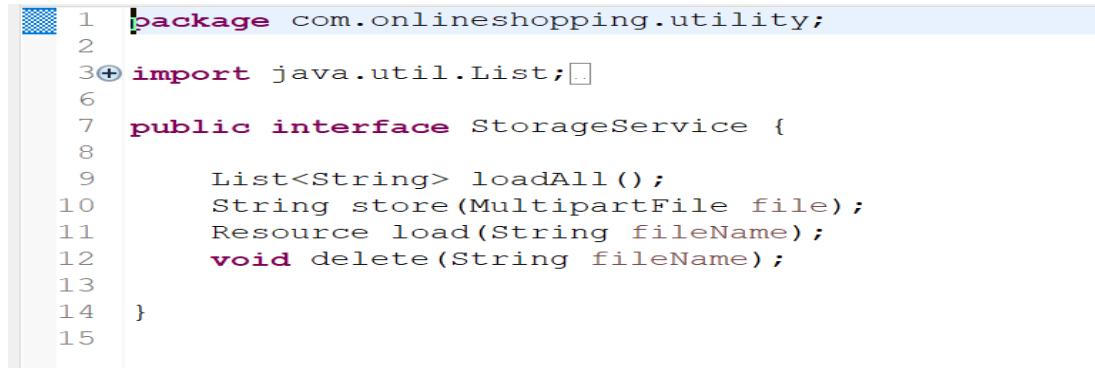
You can add additional methods based on the common functionality or utility functions your application needs.

```
1 package com.onlineshopping.utility;
2
3 public class Helper {
4
5     public static String getAlphaNumericOrderId() {
6
7         String AlphaNumericString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
8             + "0123456789"
9             + "abcdefghijklmnopqrstuvwxyz";
10
11         StringBuilder sb = new StringBuilder(10);
12
13         for (int i = 0; i < 10; i++) {
14
15             int index
16                 = (int) (AlphaNumericString.length()
17                         * Math.random());
18
19             sb.append(AlphaNumericString
20                     .charAt(index));
21
22         }
23
24     return sb.toString().toUpperCase();
25
26
27
28 }
```

Figure:75

Storage Service:

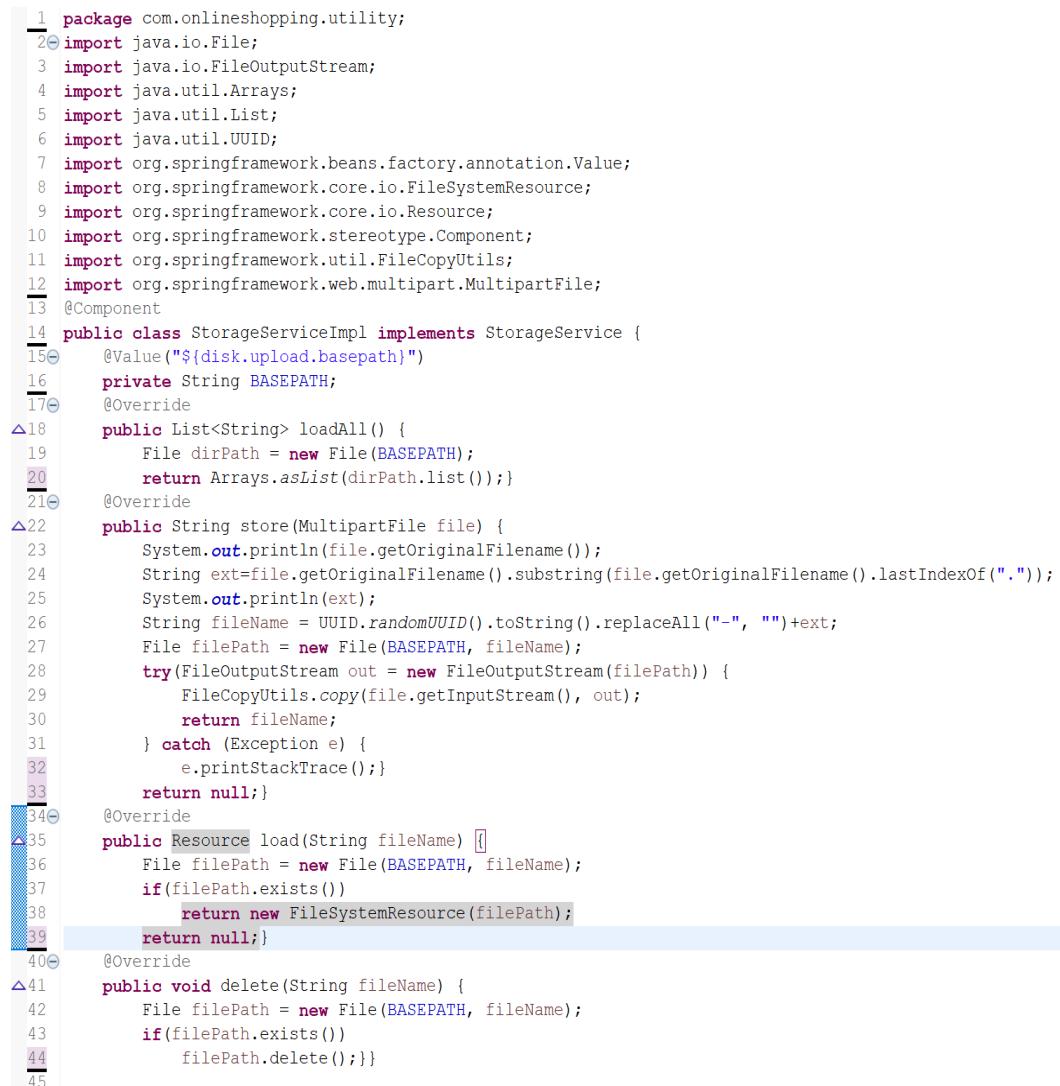
A Storage Service in a React application typically involves interactions with client-side storage mechanisms such as local Storage or session Storage. These services are often used to store and retrieve data on the user's device.



```
1 package com.onlineshopping.utility;
2
3 import java.util.List;
4
5 public interface StorageService {
6
7     List<String> loadAll();
8     String store(MultipartFile file);
9     Resource load(String fileName);
10    void delete(String fileName);
11
12 }
13
14 }
```

Figure:76

Storage Serviceimpl:



```
1 package com.onlineshopping.utility;
2
3 import java.io.File;
4 import java.io.FileOutputStream;
5 import java.util.Arrays;
6 import java.util.List;
7 import java.util.UUID;
8 import org.springframework.beans.factory.annotation.Value;
9 import org.springframework.core.io.FileSystemResource;
10 import org.springframework.core.io.Resource;
11 import org.springframework.stereotype.Component;
12 import org.springframework.util.FileCopyUtils;
13 import org.springframework.web.multipart.MultipartFile;
14 @Component
15 public class StorageServiceImpl implements StorageService {
16     @Value("${disk.upload.basepath}")
17     private String BASEPATH;
18     @Override
19     public List<String> loadAll() {
20         File dirPath = new File(BASEPATH);
21         return Arrays.asList(dirPath.list());
22     }
23     @Override
24     public String store(MultipartFile file) {
25         System.out.println(file.getOriginalFilename());
26         String ext=file.getOriginalFilename().substring(file.getOriginalFilename().lastIndexOf("."));
27         System.out.println(ext);
28         String fileName = UUID.randomUUID().toString().replaceAll("-", "")+ext;
29         File filePath = new File(BASEPATH, fileName);
30         try(FileOutputStream out = new FileOutputStream(filePath)) {
31             FileCopyUtils.copy(file.getInputStream(), out);
32             return fileName;
33         } catch (Exception e) {
34             e.printStackTrace();
35         }
36         return null;
37     }
38     @Override
39     public Resource load(String fileName) {
40         File filePath = new File(BASEPATH, fileName);
41         if(filePath.exists())
42             return new FileSystemResource(filePath);
43         return null;
44     }
45     @Override
46     public void delete(String fileName) {
47         File filePath = new File(BASEPATH, fileName);
48         if(filePath.exists())
49             filePath.delete();
50     }
51 }
```

Figure:77

5.2.8 Resources:

```
1 # MySQL Properties
2 spring.datasource.url=jdbc:mysql://localhost:3306/online_shopping?createDatabaseIfNotExist=true&useUnicode=true
3 spring.datasource.username=root
4 spring.datasource.password=9666516041@vv
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
6 spring.jpa.hibernate.ddl-auto=update
7 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
8 disk.upload.basepath=C:/Users/91950/Downloads/Ecommerce_Project_Fullstack/images
9
10 spring.jackson.serialization.fail-on-empty-beans=false
```

Figure:78

5.3 Server-side Logic (Using Spring Boot):

5.3.1 User Authentication (both user and admin):

- Used Spring Security for authentication and authorization.
- Implemented a custom *UserController.java* to load user details from the database in *src/main/java*.

5.3.2 Product Management:

- Created Product controllers (*ProductController.java*) for managing products in *src/main/java*.
- Implement CRUD operations for products.

5.3.3 Order Processing:

- Implement Order controllers (*OrderController.java*) for handling orders in *src/main/java*.
- To Create an order, associate products with orders, and calculate totals.

5.3.4 User Management:

- Implemented controllers *UserController.java* for user management.
- Allowed users to register and view order history (*MyOrderResponse.java*).

5.4 Database design:

Entities that represent your e-commerce domain and create relationships between them shown below.

Class Diagram:

Figure 79 shows the Example class diagram of the application.

Users: The user table is responsible for containing the necessary information when starting to register a member of a website. The fields in the user table are suitable for each user's role.

Roles: The roles table is used to store role information and the roles table has a relationship with the users table to determine which role each user belongs to.

Order: The order table is used to store order information, including shipping information fields, such as name, address, and phone number when a user starts an order from the website.

Products: The products table is used to store information of a product including fields such as name, price, and quantity. The field information in the product table is relevant for the different product categories.

Categories: The product category table is used to contain product type information including field name and category type codes. All category type codes are unique.

Store: The store table is used to store all store information for each branch.

Brands: The product brand table is used to contain manufacturer information and all manufacturer-related information such as image and the manufacturer's name

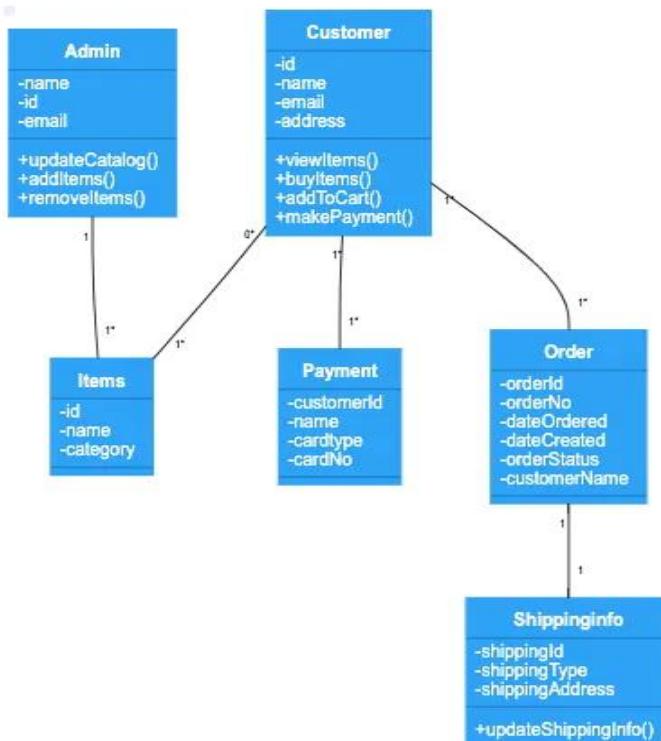


Figure:79 Class Diagram

Sequence Diagram

Login

The sequence diagram in Figure 80 shows sequences in login-functionality. First of all, the user will enter the email and password, and then click on the login button. The system will check that the email and password in the database are correct. If the login is not successful, the user will need to log in again. In case the user logs in successfully, the system grants permissions to access the feature of the website, the user can use the functions in the website.

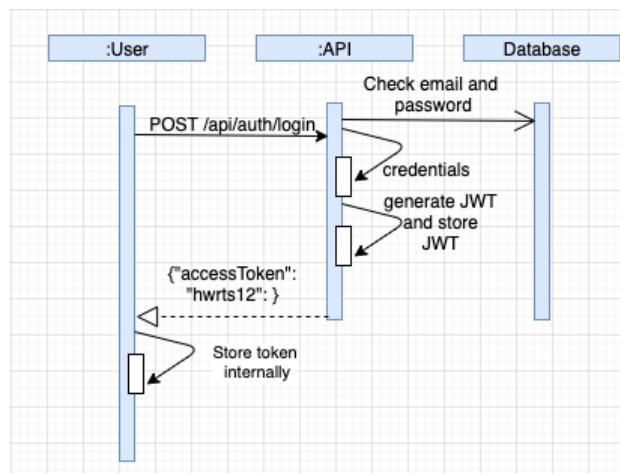


Figure:80 Login

Logout

The sequence diagram in Figure 81 shows the sequence in logout functionality. The user can log out from the account by clicking the logout button. After the user logout, the system will automatically redirect the user to the home page.

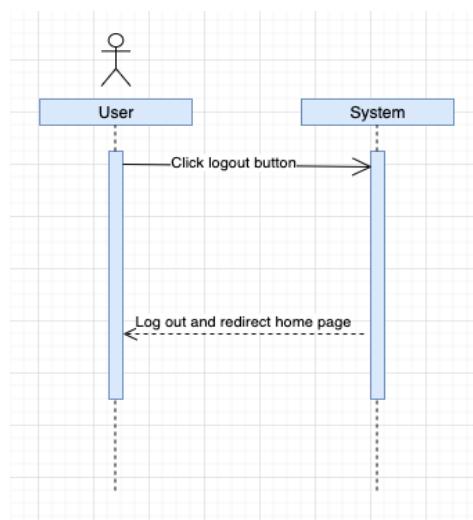


Figure:81 Logout

Add Product

The sequence diagram in Figure 82 shows the steps to add a product. The staff must login to the management page and click the add product button to enter the product information. If the product information is wrong, the system will notify the user to enter the information again. The system will save the product to the database when the information is correct.

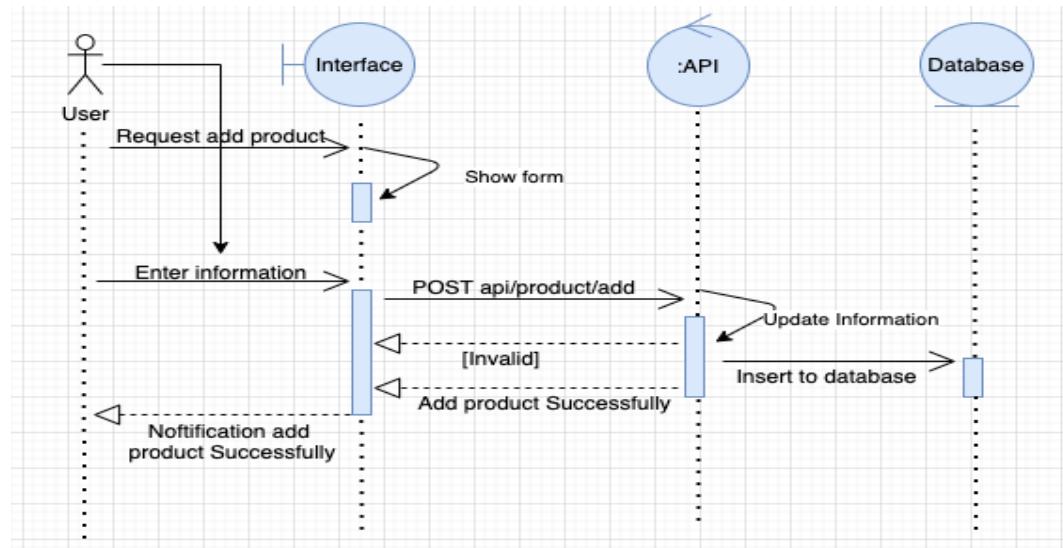


Figure:82 Add Product

Add Categories

The sequence diagram in Figure 83 shows the steps to add a category. The staff must login to the management page and click the add category button to add the category information to be added. If the category information is wrong, the system will notify the user to enter the information again. The system will save the category to the database when the information is correct.

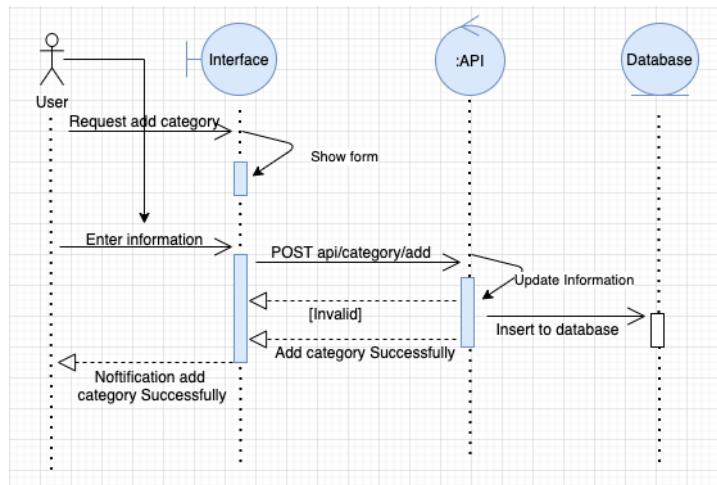


Figure:83 Categories

Add User

The sequence diagram in Figure 84 shows the steps in adding a user. The user accesses the account registration page to fill in the account information. If the account information is valid, it is updated to the database. In case, the information is wrong, the system will request to review the account information.

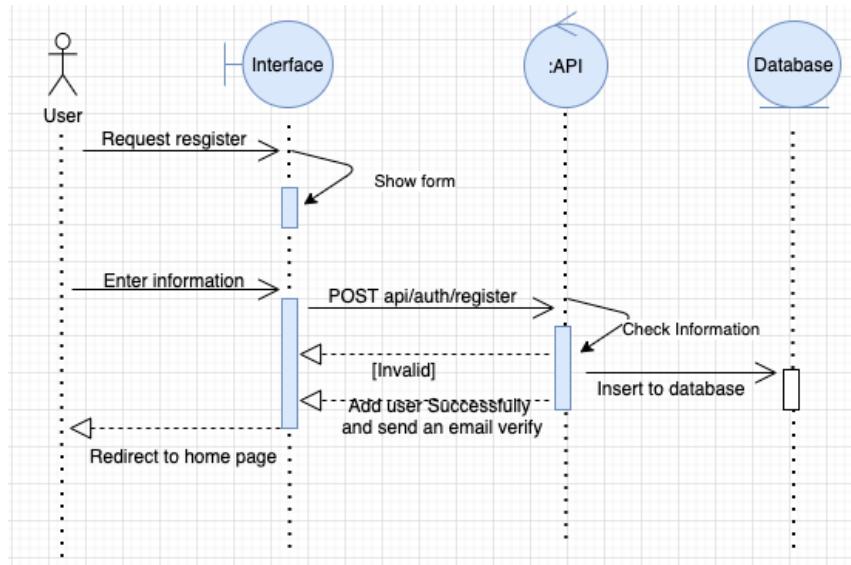


Figure:84 Add User

Assign Role

The sequence diagram in Figure 85 shows the steps in assigning roles. The Admin logs in to the system to decentralize accounts for users. Admin is allowed to grant permissions for the role manager and the author permissions for the product management. After the admin adds, modifies and deletes accounts for the user and if that account information is valid, it is allowed to update the database.

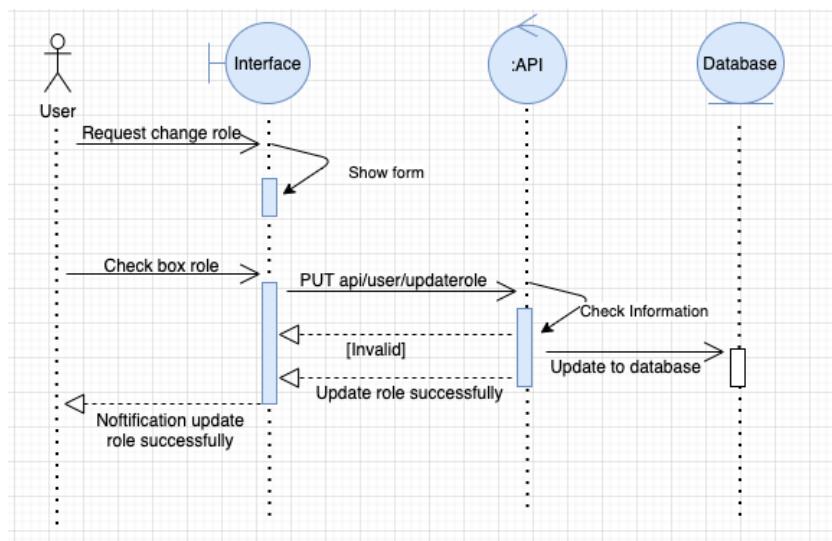


Figure:85 Assign Role

Order

The sequence diagram in Figure 86 shows the steps in order confirm. The user needs to login to the system to make the payment; the user clicks check out button and needs to fill the information before starting payment for the user's order. If the user information is wrong, the system will notify the user to enter the information again. The system will send an email to the user when the information is correct.

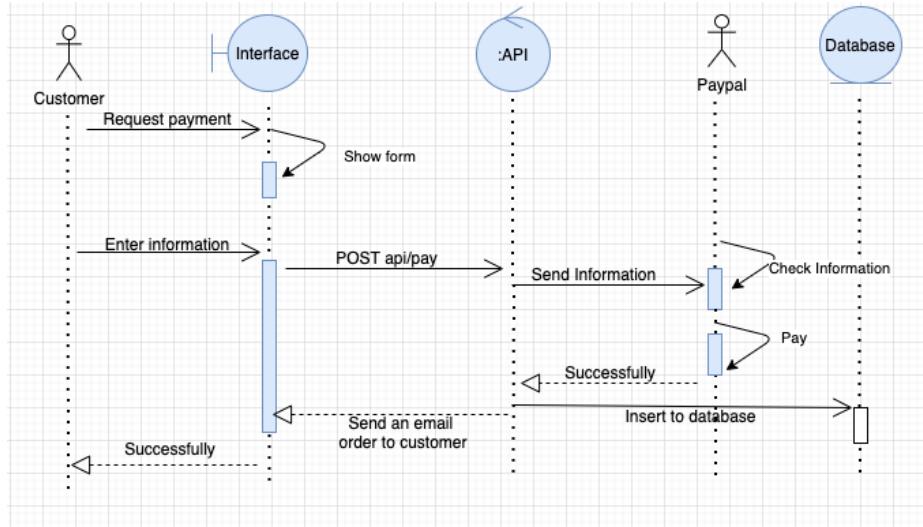


Figure:86 Order

5.5 APIs and endpoints used for communication with the front-end:

5.5.1 User Authentication:

User Registration:

- **Endpoint:** /api/users/register
- **Method:** POST
- **Description:** Allows users to register by providing necessary information.

User Login:

- **Endpoint:** /api/users/login
- **Method:** POST
- **Description:** Authenticates users and returns an authentication token.

User Logout:

- **Endpoint:** /api/users/logout
- **Method:** POST
- **Description:** Logs out the authenticated user by invalidating the session.

5.5.2 Product Management:

Get All Products:

- **Endpoint:** /api/products
- **Method:** GET
- **Description:** Retrieves a list of all products.

Get Product by ID:

- **Endpoint:** /api/products/:productId
- **Method:** GET
- **Description:** Retrieves details of a specific product based on the provided ID.

Add a New Product:

- **Endpoint:** /api/products/add
- **Method:** POST
- **Description:** Adds a new product to the system.

5.5.3 Order Management:

Place an Order:

- **Endpoint:** /api/orders/place
- **Method:** POST
- **Description:** Allows users to place an order for selected products.

Get User's Orders:

- **Endpoint:** /api/orders/user/: userId
- **Method:** GET
- **Description:** Retrieves a list of orders associated with a specific user.

Get Order Details:

- **Endpoint:** /api/orders/: orderId
- **Method:** GET
- **Description:** Retrieves details of a specific order based on the provided ID.

Miscellaneous:

Upload Image:

- **Endpoint:** /api/upload
- **Method:** POST
- **Description:** Allows users to upload images, for example, for product .

6. Database Design

Database Schema:

The database schema looks like this:

Users Table:

- Fields: user_id, username, email, password_hash, created_at, updated_at

Products Table:

- Fields: product_id, name, description, price, stock_quantity, created_at, updated_at

Orders Table:

- Fields: order_id, user_id (foreign key referencing Users Table), order_date, total_amount, status

Order Items Table:

- Fields: order_item_id, order_id (foreign key referencing Orders Table), product_id (foreign key referencing Products Table), quantity, subtotal

Data Model:

Users:

- Each user is uniquely identified by a user_id.
- Users have attributes such as username, email, and password_hash for authentication.
- Timestamps created_at and updated_at track when the user record was created or last updated.

Products:

- Each product is uniquely identified by a product_id.
- Products have attributes like name, description, price, and stock_quantity.
- Timestamps created_at and updated_at track when the product record was created or last updated.

Orders:

- Each order is uniquely identified by an order_id.

- Orders are associated with a user through the user_id foreign key.
- The order_date, total_amount, and status are attributes of an order.

Order Items:

- Each order item is uniquely identified by an order_item_id.
- Order items are associated with an order through the order_id foreign key.
- Each order item references a product through the product_id foreign key.
- Attributes include quantity and subtotal.

Choice of database management system and its justification:

Choosing MySQL as a Database Management System (DBMS) involves considering various factors aligned with the specific requirements of your application. Below are some justifications for choosing MySQL:

1. Mature and Stable:

Justification:

- MySQL is one of the oldest and most well-established relational database systems, offering stability and maturity.
- It has a proven track record in various industries and has been used in production environments for many years.

2. Open Source and Cost-Effective:

Justification:

- MySQL is an open-source database, providing cost advantages, especially for startups or projects with budget constraints.
- The community edition is freely available, reducing the total cost of ownership.

3. Community Support:

Justification:

- MySQL has a large and active community of developers and users.
- A vibrant community ensures good support, frequent updates, and a wealth of online resources for troubleshooting and learning.

4. Cross-Platform Compatibility:

Justification:

- MySQL is compatible with various operating systems, including Linux, Windows, and macOS.
- This cross-platform support makes it versatile and easy to integrate into diverse

environments.

5. Ease of Use and Administration:

Justification:

- MySQL is known for its simplicity and ease of use, making it a great choice for developers who may not be database experts.
- The management tools, such as MySQL Workbench, provide a user-friendly interface for database administration.

6. Scalability:

Justification:

- MySQL supports both vertical and horizontal scalability.
- Vertical scaling involves adding resources to a single server, while horizontal scaling involves distributing data across multiple servers using techniques like sharding.

7. Robust Feature Set:

Justification:

- MySQL offers a comprehensive set of features, including support for transactions, subqueries, triggers, stored procedures, and more.
- It is suitable for a wide range of applications, from small-scale projects to large-scale enterprise solutions.

8. ACID Compliance:

Justification:

- MySQL adheres to the principles of ACID (Atomicity, Consistency, Isolation, Durability), ensuring data integrity and consistency in transactional operations.

9. Industry Adoption:

Justification:

- MySQL is widely adopted across various industries, including web development, e-commerce, finance, and telecommunications.
- Its popularity indicates a level of trust and reliability among developers and organizations.

10. Integration with Popular Technologies:

Justification:

- MySQL integrates seamlessly with popular programming languages, frameworks, and platforms.

7. Authentication and Security

7.1 User Authentication:

Frontend (React):

- Implement secure authentication workflows, preferably using token-based authentication like JWT.
- Use HTTPS to encrypt data transmitted between the client and the server.
- Store tokens securely, such as in HTTP-only cookies.
- Implement secure password recovery mechanisms.

Backend (Spring Boot):

- Use Spring Security for user authentication and authorization.
- Implement secure password hashing (e.g., bcrypt) to store passwords.
- Set up proper session management and handle user sessions securely.
- Implement account lockout mechanisms to prevent brute force attacks.

Database (MySQL):

- Store only hashed and salted passwords.
- Limit database access and permissions to the principle of least privilege.
- Regularly audit database access logs.

7.2 Session Management:

Frontend (React):

- Use secure, HTTP-only cookies for session management.
- Implement session timeouts and provide users with the ability to log out.

Backend (Spring Boot):

- Implement secure session management using Spring Session.
- Store session data securely and validate session tokens on each request.

Database (MySQL):

- Regularly clean up expired sessions and implement proper session database design.

7.3 Data Validation and Sanitization:

Frontend (React):

- Validate user input on the client side to prevent common security vulnerabilities.
- Sanitize user input before rendering it on the client.

Backend (Spring Boot):

- Implement server-side validation and sanitize input to prevent injection attacks.
- Use Hibernate Validator for input validation.

Database (MySQL):

- Validate and sanitize data before storing it in the database.
- Implement input validation checks in stored procedures.

7.4 HTTPS and Secure Communication:

Frontend (React) and Backend (Spring Boot):

- Use HTTPS to encrypt data in transit.
- Implement secure headers, including Content Security Policy (CSP) and HTTP Strict Transport Security (HSTS).

Database (MySQL):

- Configure MySQL to use encrypted connections.

7.5 Payment Security:

Frontend (React):

- Avoid storing sensitive payment information on the client.
- Use secure APIs for payment processing.

Backend (Spring Boot):

- Implement PCI DSS compliance for handling payment information.
- Use secure payment gateways and comply with industry standards.

Database (MySQL):

- Encrypt sensitive payment data before storing it in the database.
- .

8. Project Management

- 6 Detail the project timeline, milestones, and sprints (if applicable).
- 7 Discuss the project management approach used during development.

8.1 Login Use Case Specification

The table 1 shows the use case specification for Login functionality

Table 1. Login.

Name	Login
Summary	Function log into the system.
Event	<ol style="list-style-type: none">1. The system displays the login form.2. Enter your username and account (both fields are required) and click "Login".3. The system checks the login information (Other event stream: Wrong credentials).4. The system displays the main form.
	1. Wrong password or Email
Other Event	The system displays a message that the login account is not valid.
Special Required	Not available
System state before use case execution	Actor: all of the actors Require: Not available
System state after use case execution	The user who logs into the system can use all the permission of the system allowed.

8.2 Logout Use Case Specification

The table 2 shows the use case specification for Logout functionality.

Table 2. Logout.

Name	Logout
Summary	Function log out to the system.
Event	1. The user clicks on log out 2. The system logs out and returns to the home page
Other Event	Not available
Special Required	Not available
System state before use case execution	Actor: all of actors Require: The user has logged on to the system.
System state after use case execution	User logs out of the system.

8.3 Product Management Use Case Specification

The table 3 shows the use case specification for adding functionality.

Table 3. Add Product.

Name	Add product
Summary	Add product information such as: product name, model name, manufacturer, supplier and other details.
Event	1. Go to product management, press the button “Add product”.

	<p>2. The user enters the necessary information (including some required information) and presses "Save".</p> <p>3. The system checks the information, if the information is valid, the next step will be taken.</p> <p>(Other event stream: Invalid information).</p> <p>4. The system saves the data and reports it successfully.</p> <p>(Other event: Cannot add product to database).</p>
Other Event	<p>1. Invalid information:</p> <p>The system displays a red message on the spot of an error and asks to re-enter information.</p> <p>2. Cannot save to database: Error while adding => Ask user to re-enter information, if still error contact the development team.</p>
Special Required	Not available
System state before use case execution	<p>Actor: Admin, Staff</p> <p>Require: The user has logged into the system and has the right to use this function.</p>
System state after use case execution	User successfully added product information to the system.

8.4 Categories Management Use Case Specification

The table 5 shows the use case specification for adding category functionality.

Table 5. Add Categories.

Name	Add categories type
Summary	Add categories type information such as name, code.
Event	<p>1. Go to the product category manager, click the button "Add categories."</p> <p>2. The user enters the necessary information (including some required information) and presses "Save".</p> <p>3. The system checks the information, if the information is valid, the next step will be taken.</p> <p>(Other event: Invalid information).</p> <p>4. The system saves the data and notify it of success.</p> <p>(Other event stream: Cannot update to the database)</p>
Other Event	<p>1. Invalid information:</p> <p>The system displays a message asking for information re-input.</p> <p>2. Cannot update the database: Error while updating => Request user to re-enter information; if still error, contact development team.</p>
Special Required	Not available
System state before usecase execution	Actor: Staff, Admin
	Require: The user has logged into the system and has the right to use this function.
System state after use case execution	User successfully added the information to the system.

9. Results and Evaluation

- 9 Discuss the achieved results and whether they meet the project's goals.
- 10 Evaluate the performance and efficiency of the application.

9.1 List of Pages

The table 10 shows the pages that were implemented for the E-commerce web shop.

Table 10. Page List.

STT	Page
1	Home Page
2	Login
3	Admin Page
4	Register Page
6	Add Product Page
8	Add Categories Page
11	Cart Page
12	Checkout Page
13	My Orders

HomePage

The Figure homepage of an e-commerce website serves as the virtual storefront, the first point of contact for visitors, and a gateway to the various products and services offered.

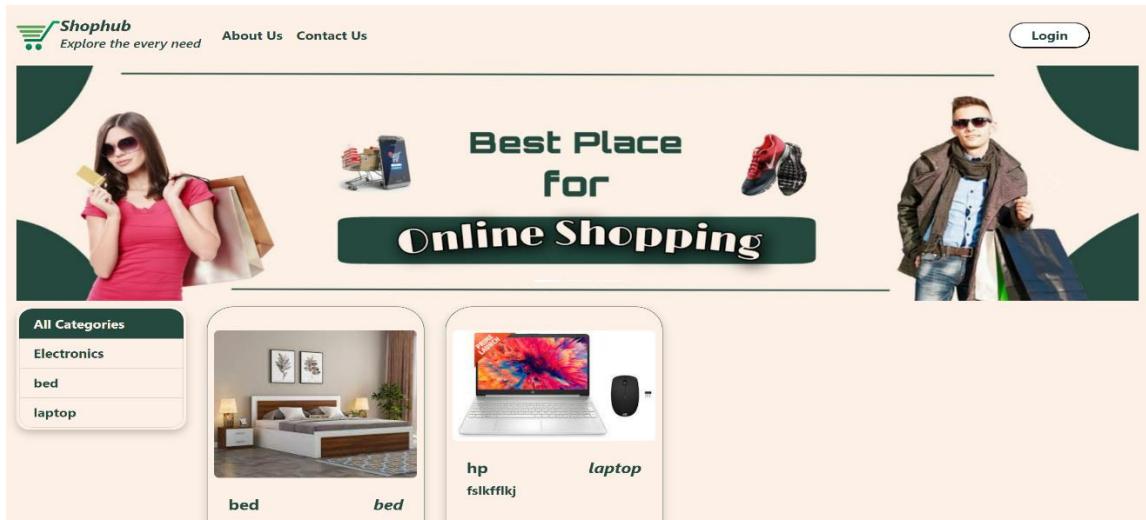


Figure:87 HomePage

Login Page

The Login page which is shown in Figure allows users to log in to the websites using email and password when user want to order products through websites store. After successfully logging in, the user can buy products.

Input: Email, password.

Process: Enter your username and password and check if the username and password are valid.

Output: If the username and password are correct, the user can use the system. If wrong, the system will request to re-enter.

A screenshot of a 'User Login' form. The title 'User Login' is at the top in a dark green bar. Below it is a 'User Role' section with a dropdown menu labeled 'Select Role'. The next sections are 'Email Id' and 'Password', each with an input field. At the bottom is a large green 'Login' button. Below the button is a link 'New User? Register Here'.

Figure: 88 Login

Register Page

The register page which is shown in Figure 28 allows user can register their account.

After successfully register, user can receive an email to active account.

Input: Name, Email, Password, Confirm Password.

Process: Enter information necessary and check if the information are valid.

Output: If the user information are correct, the user will be received an email.

The screenshot shows a registration form titled "Add User". The form is divided into two main sections: "User Role" and "Street".

User Role: A dropdown menu labeled "Select Role".

Street: A large text input field for street address.

First Name: A text input field.

Last Name: A text input field.

Email Id: A text input field.

Password: A text input field.

Mobile No: A text input field.

City: A text input field.

Pincode: A text input field.

Register User: A green button at the bottom of the "Street" section.

Figure:89 Register

Admin Page

The admin page which is shown in Figure allows admin to manage products data for some method adds, update and remove product through websites admin. After successfully logging in, admin has full control data of website.

Input: Email, password.

Process: Enter your username and password and check if the username and password are valid.

Output: If the username and password are correct, the admin can use the system. If wrong, the system will request to re-enter



Figure:90 Admin Page

Add Product

The Add/Update page, which is shown in Figure 29 allows admin to add or update product through websites admin.

Input: product information.

Output: If the product information is correct, the admin will be received notification.

A screenshot of the "Add Product" form. It consists of several input fields: "Product Title" (text input), "Product Description" (text area), "Category" (dropdown menu with placeholder "Select Category"), "Product Quantity" (text input), "Product Price" (text input), and "Select Product Image" (file input field showing "No file chosen"). A "Choose File" button is next to the input field. At the bottom is a "Add Product" button.

Figure:91 Add Product

Add Category Page

The category page which is shown in Figure 30 allows admin to manage categories data for some methods such as add, update and remove categories through websites admin.

After successfully logging in, admin has full control data of website.

Input: Email, password.

Process: Enter your username and password and check if the username and password are valid.

Output: If the username and password are correct, the admin can use the system. If wrong, the system will request to re-enter.

Add Category

Category Title

enter title..

Category Description

enter description..

Add Category

Figure:92 Add Category

Cart Page

The Cart Page which is shown in Figure 39 allows where users can review all the products which were interested in before adding them to the shopping cart.

My Cart				
Product	Name	Description	Quantity	Action
	bed	sdfsd	1	<button>Delete</button>
	hp	fslkfflkj	1	<button>Delete</button>

Figure:93 Cart Page

Checkout Page

The Checkout page which is shown in Figure 37 allows user to enter their information data for order product through web page. Client can review all products which was ordered already.

Input: Information fields.

Process: The system will save client information in database.

Output: The system will be redirect to payment page.

The screenshot shows a form titled "Payment Details". It contains four input fields: "Name on Card" (empty), "Card Number" (empty), "Valid Through" (empty), and "CVV" (empty). Below these fields is a button labeled "Pay Rs602101.0".

Figure:94 Checkout Page

My Orders

Which we can see All Orders in the My Orders pages

My Orders										
Order Id	Product	Name	Description	Quantity	Total Price	Order Date	Delivery Date	Delivery Status	Delivery Person	Delivery Mobile No
R4EHIGUIF4		hp	fslkfflkj	1	100000.0	19-11-2023 15:11	2023-11-19 Afternoon	Delivered	v	9876543209
K60UYODZGU		bed	sdfsd	10	5021010.0	19-11-2023 15:38	2023-11-19 Afternoon	Delivered	v	9876543209

Figure:95 MyOrders Page

10. Conclusion

After implementing the topic and systematically presenting the basic contents of Spring Boot, ReactJS, MySQL and a number of other technologies and techniques in building enterprise Java applications on the web, helping to understand overview of Spring Framework as well as the basic principles and working mechanism of this framework. On the other hand, the websites has a three tier web model architecture in Spring and the mechanisms for securing a web application are supported in the Spring Security module. The thesis gave understanding how the communication mechanism between client and server in modern RESTful service-oriented web model works and how to communication between components of a system.

Commercial web design with separate web server and data server model to increase performance accessing the website. The websites is built to communicate with the server through a RESTful service with the help of ReactJS. UI design with Bootstrap with responsive support makes the websites responsive and user experienced. Building a successful e-commerce website with full basic functions so that users can buy product easily. The web application is simple, elegant and functional with important features for an online store.

The theme of E-commerce web shop is quite popular and highly capable in practical application. However, due to the limited time of research and experience, the E-commerce web only develops at the level of completing the requirements of the topic, the processing speed is not yet completed. The further research will focus on understanding the method of managing the system as well as handling large data blocks with high efficiency, expanding the scope of this project.