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

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; Bloom-idea, 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.

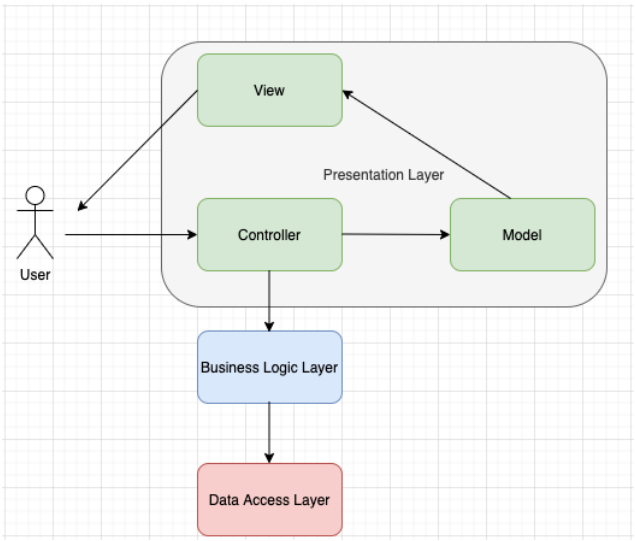


Figure 5. Three Tier Architecture

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 thatcan 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 24 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 withthe data) to the client.

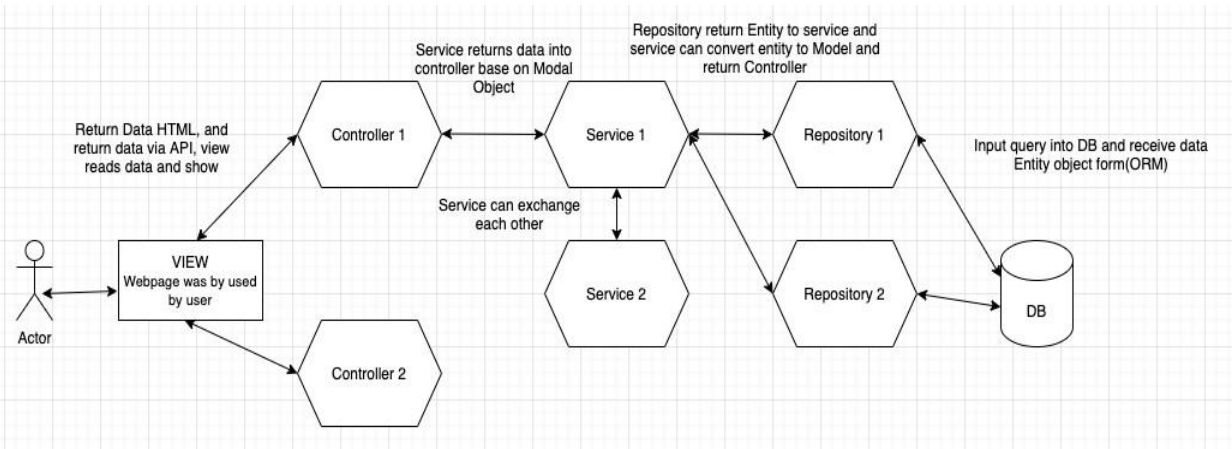


Figure 7. Data flow model.

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 awebsite, 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 extendedJavaScript 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 websitesinterface 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 thatresponsive 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 ofvariation will provide a better and more consistent user experience than a page designedfor 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-de- fined. JavaScript is the convenient and efficient because it is a dynamic programming language. Besides, JavaScript works based on HTML and CSS to a create dynamic con-tent on the website. Moreover, JavaScript is mainly used in client-side programming without installing environment settings whereas with Java or Python environment set- tings would have to be installed. Therefore, JavaScript programming language is trustedmore 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, theuse 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 JavaScriptprogramming 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 ofthree 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 sourcedeveloped 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 applica- tions), 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 packageand 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, andXML configuration.
- With Spring Boot it is simple to communicate applications with Springecosystems, 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 andtest Spring Boot (Java or Groovy) applications from order brief effec- tively and rapidly.
- Spring Boot gives a great deal of modules to create and test Spring Bootapplications 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 relationshipsbetween those tables.
- Client sends SQL requests with a special command on MySQL.
- The application on the server receives and responds to the informationand 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 utilityfunctions.
- High security: MySQL has a lot of security features, including high-levelsecurity types.
- Multi-function: MySQL provides many features that any relational data-base management system should expect.
- Powerful and easily scalable: MySQL is capable of handling largeamounts 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 ef-fects of crashes.

Drawbacks of MySQL:

- Restricted: MySQL is limited to a few features that applications mayneed
- 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 be-comes 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:

Figure:11

In project:

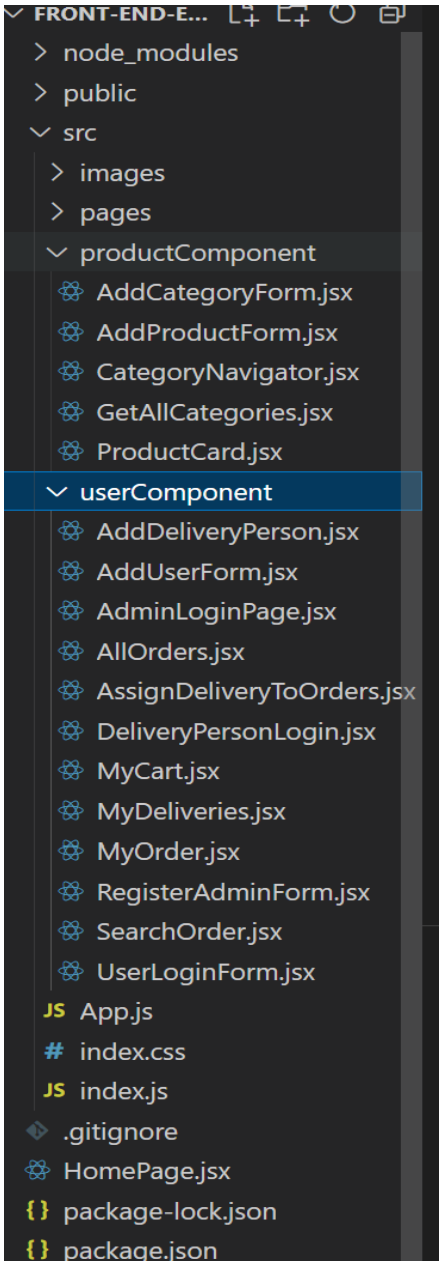


Figure:12

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.

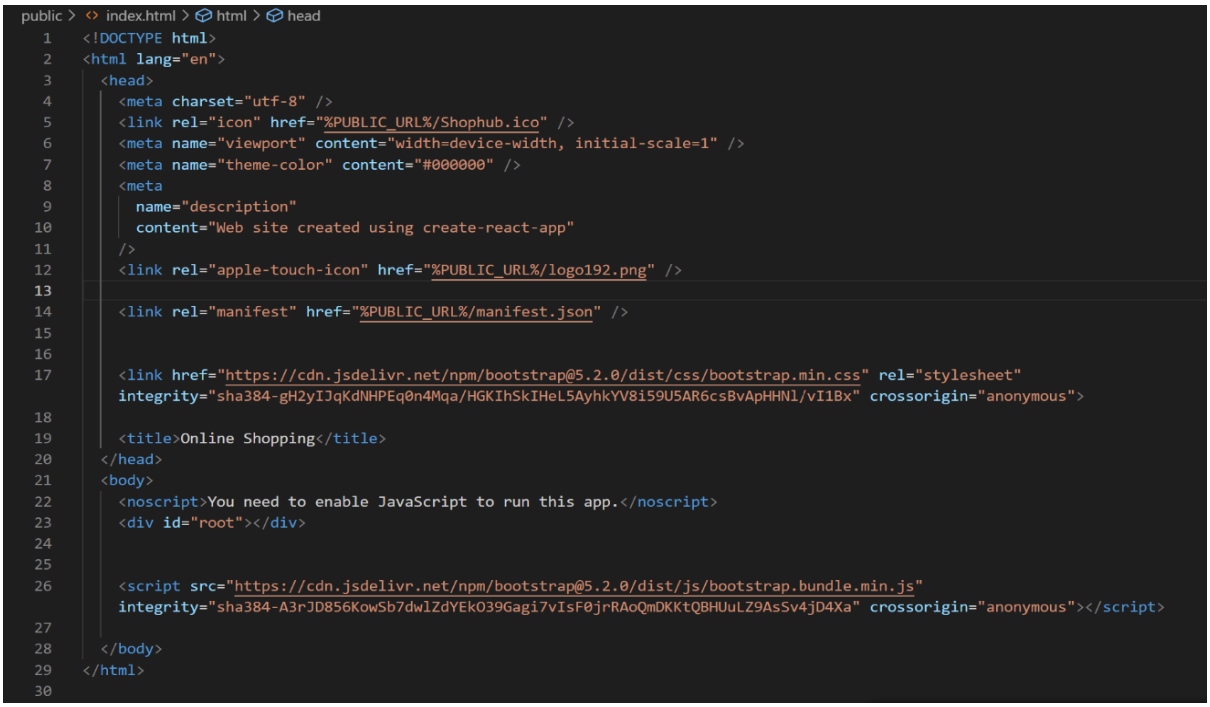


Figure:13

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.

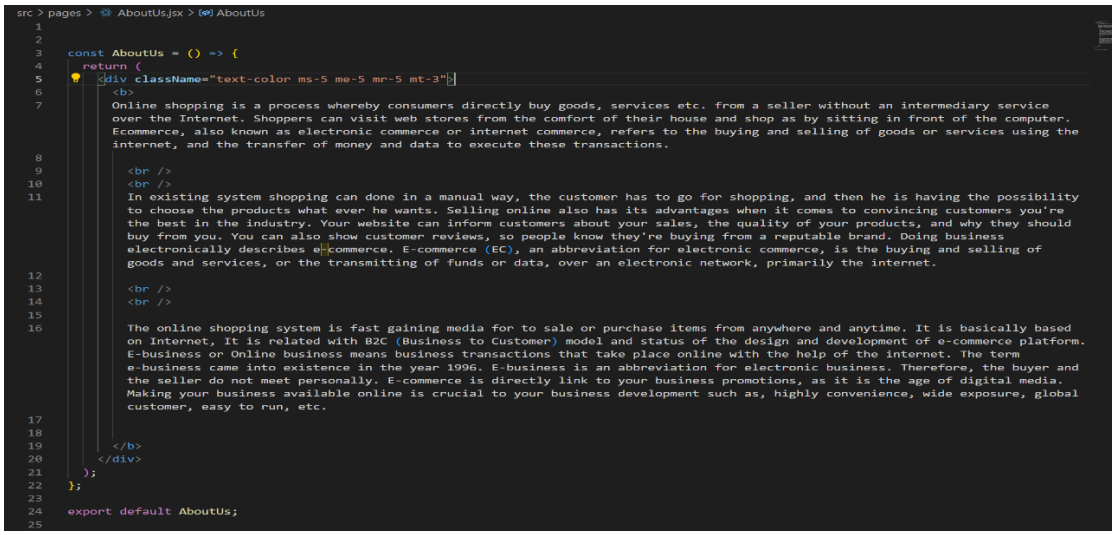


Figure:14

- **AddCardDetails**

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

```
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 aligns-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 class="card-title text-center">Payment Details</h5>
        </div>
        <div class="card-body text-color custom-bg">
          <form onSubmit={payForOrder}>
            <div class="mb-3">
              <label for="name" class="form-label">
                <b> Name on Card</b>
              </label>
              <input
                type="text"
                class="form-control"
                id="name"
                name="cardName"
                onChange={handleCardInput}
                value={card.cardName}
                required
              />
            </div>
```

```
77              <div class="mb-3">
78                <label for="cardNumber" class="form-label">
79                  <b> Card Number</b>
80                </label>
81                <input
82                  type="text"
83                  class="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 for="validThrough" class="form-label">
94                  <b>Valid Through</b>
95                </label>
96                <input
97                  type="text"
98                  class="form-control"
99                  id="validThrough"
100                 name="validThrough"
101                 onChange={handleCardInput}
102                 value={card.validThrough}
103                 required
104               />
105             </div>
```

```
                <div class="mb-3">
                  <label for="cvv" class="form-label">
                    <b>CVV</b>
                  </label>
                  <input
                    type="text"
                    class="form-control"
                    id="cvv"
                    name="cvv"
                    onChange={handleCardInput}
                    value={card.cvv}
                    required
                  />
                </div>

                <input
                  type="submit"
                  class="btn custom-bg-text bg-color"
                  value={"Pay Rs" + priceToPay}
                />

                <ToastContainer />
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  };

  export default AddCardDetails;

```

Figure:15

- **AdminHeader**

  To Display different for the admin(seller)

```
src > pages > AdminHeader.jsx > ...
  import { Link, useNavigate } from "react-router-dom";
  import { ToastContainer, toast } from "react-toastify";
  import "react-toastify/dist/ReactToastify.css";

  const AdminHeader = () => {
    let navigate = useNavigate();

    const user = JSON.parse(sessionStorage.getItem("active-admin"));
    console.log(user);

    const adminLogout = () => {
      to (property) CommonOptions.hideProgressBar?: boolean | undefined
         Hide or show the progress bar. Default: false
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });
      sessionStorage.removeItem("active-admin");
      window.location.reload(true);
    };

    return (
      <ul class="navbar-nav ms-auto mb-2 mb-lg-0 me-5">
        <li class="nav-item">
          <Link to="/addcategory" class="nav-link active" aria-current="page">
            <b className="text-color"> Add Category</b>
          </Link>
        </li>

        <li class="nav-item">
          <Link to="/addproduct" class="nav-link active" aria-current="page">
            <b className="text-color">Add Product</b>
          </Link>
        </li>

```

```
          <li class="nav-item">
            <Link
              to="/user/admin/allorder"
              class="nav-link active"
              aria-current="page"
            >
              <b className="text-color">All Orders</b>
            </Link>
          </li>

          <li class="nav-item">
            <Link
              to="/user/admin/assigndelivery"
              class="nav-link active"
              aria-current="page"
            >
              <b className="text-color">Assign Order Delivery</b>
            </Link>
          </li>

          <li class="nav-item">
            <Link
              to=""
              class="nav-link active"
              aria-current="page"
              onClick={adminLogout}
            >
              <b className="text-color">Logout</b>
            </Link>
            <ToastContainer />
          </li>
        </ul>
    );
  };

  export default AdminHeader;

```

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 > ...
  import carousel1 from "../images/carousel_1.jpeg";

  const Carousel = () => {
    return (
      <div
        id="carouselExampleCaptions"
        class="carousel slide"
        data-bs-ride="false"
      >
        <div class="carousel-indicators">
          <button
            type="button"
            data-bs-target="#carouselExampleCaptions"
            data-bs-slide-to="0"
            class="active"
            aria-current="true"
            aria-label="Slide 1"
          ></button>
          <button
            type="button"
            data-bs-target="#carouselExampleCaptions"
            data-bs-slide-to="1"
            aria-label="Slide 2"
          ></button>
          <button
            type="button"
            data-bs-target="#carouselExampleCaptions"
            data-bs-slide-to="2"
            aria-label="Slide 3"
          ></button>
        </div>
        <div class="carousel-inner">
          <div class="carousel-item active">
            <img src={carousel1} class="d-block w-100" alt="..." />
          </div>
          <div class="carousel-item">
            <img src={carousel1} class="d-block w-100" alt="..." />
          </div>
```

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.

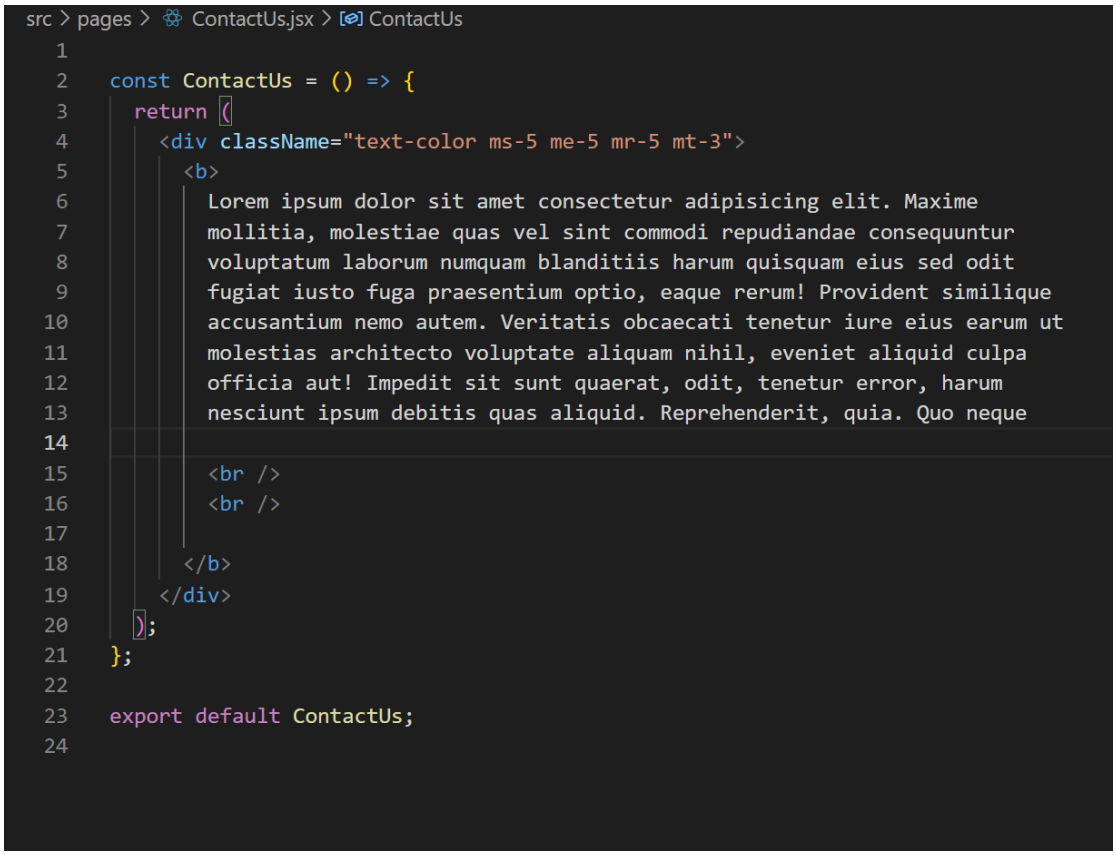


Figure:18

- **Delivery Person header**

  Separate Interface for the delivery person

src > pages > DeliveryPersonHeader.jsx > ...

```jsx
import { Link, useNavigate } from "react-router-dom";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

const DeliveryPersonHeader = () => {
  let navigate = useNavigate();

  const user = JSON.parse(sessionStorage.getItem("active-delivery"));
  console.log(user);

  const userLogout = () => {
    toast.success("logged out!!!", {
      position: "top-center",
      autoClose: 1000,
      hideProgressBar: false,
      closeOnClick: true,
      pauseOnHover: true,
      draggable: true,
      progress: undefined,
    });
    sessionStorage.removeItem("active-delivery");
    window.location.reload(true);
  };

  return (
    <ul class="navbar-nav ms-auto mb-2 mb-lg-0 me-5 text-color">
      <li class="nav-item">
        <Link
          to="/user/delivery/myorders"
          class="nav-link active"
          aria-current="page"
        >
          <b className="text-color">My Deliveries</b>
        </Link>
      </li>

      <li class="nav-item">
        <Link
          to="/user/admin/searchOrder"
          class="nav-link active"
          aria-current="page"
        >
          <b className="text-color">Update Order Delivery</b>
        </Link>
      </li>

      <li class="nav-item">
        <Link
          to=""
          class="nav-link active"
          aria-current="page"
          onClick={userLogout}
        >
          <b className="text-color">Logout</b>
        </Link>
        <ToastContainer />
      </li>
    </ul>
  );
};

export default DeliveryPersonHeader;
```

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 > Header

```jsx
import { Link } from "react-router-dom";
import RoleNav from "./RoleNav";
import logo from "../images/e_logo.png";

const Header = () => {
  return (
    <div>
      <nav class="navbar  navbar-expand-lg custom-bg text-color">
        <div class="container-fluid text-color">
          <img
            src={logo}
            width="50"
            height="40"
            class="d-inline-block align-top"
            alt=""
          />
          <Link to="/" class="navbar-brand">
            <i>
              <b className="text-color">Shophub</b>
              <h6 className=' text-color'>Explore <span className=' text-yellow-300'>the every need </span>  </h6>
            </i>
          </Link>

          <button
            class="navbar-toggler"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent"
            aria-expanded="false"
            aria-label="Toggle navigation"
          >
            <span class="navbar-toggler-icon"></span>
          </button>

          <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
              <li class="nav-item">
                <Link to="/about" class="nav-link active" aria-current="page">
                  <b className="text-color">About Us</b>
                </Link>
              </li>

              <li class="nav-item">
                <Link to="/contact" class="nav-link active" aria-current="page">
                  <b className="text-color">Contact Us</b>
                </Link>
              </li>
            </ul>

            <RoleNav />
          </div>
        </div>
      </nav>
    </div>
  );
};

export default Header;
```

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.

GetAllCategories.jsx | Header.jsx | *HeaderUser.jsx* | UserLoginForm.jsx | NormalHeader.jsx | Shophub.ico

src > pages > HeaderUser.jsx > ...

```jsx
import { Link, useNavigate } from "react-router-dom";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

const HeaderUser = () => {
  let navigate = useNavigate();

  const userLogout = () => {
    toast.success("logged out!!!", {
      position: "top-center",
      autoClose: 1000,
      hideProgressBar: false,
      closeOnClick: true,
      pauseOnHover: true,
      draggable: true,
      progress: undefined,
    });
    sessionStorage.removeItem("active-user");
    window.location.reload(true);
  };

  return (
    <ul class="navbar-nav ms-auto mb-2 mb-lg-0 me-5">
      <li class="nav-item">
        <Link to="/user/mycart" class="nav-link active" aria-current="page">
          <b className="text-color">My Cart</b>
        </Link>
      </li>

      <li class="nav-item">
        <Link to="/user/myorder" class="nav-link active" aria-current="page">
          <b className="text-color">My Order</b>
        </Link>
      </li>

      <li class="nav-item">
        <Link
          to=""
          class="nav-link active"
          aria-current="page"
          onClick={userLogout}
        >
          <b className="text-color">Logout</b>
        </Link>
        <ToastContainer />
      </li>
    </ul>
  );
};

export default HeaderUser;
```

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.

GetAllCategories.jsx | Header.jsx | HomePage.jsx | UserLoginForm.jsx | NormalHeader.jsx

src > pages > HomePage.jsx > ...

```jsx
import Carousel from "./Carousel";
import GetAllCategories from "../productComponent/GetAllCategories";
import ProductCard from "../productComponent/ProductCard";
import axios from "axios";
import { useState, useEffect } from "react";
import { useParams } from "react-router-dom";

const HomePage = () => {
  const [products, setProducts] = useState([]);

  const { categoryId } = useParams();

  useEffect(() => {
    const getAllProducts = async () => {
      const allProducts = await retrieveAllProducts();
      if (allProducts) {
        setProducts(allProducts);
      }
    };

    const getProductsByCategory = async () => {
      const allProducts = await retrieveProductsByCategory();
      if (allProducts) {
        setProducts(allProducts);
      }
    };

    if (categoryId == null) {
      console.log("Category Id is null");
      getAllProducts();
    } else {
      console.log("Category Id is NOT null");
      getProductsByCategory();
    }
  }, [categoryId]);

  const retrieveAllProducts = async () => {
    const response = await axios.get("http://localhost:8080/api/product/all");

    return response.data;
  };

  const retrieveProductsByCategory = async () => {
    const response = await axios.get(
      "http://localhost:8080/api/product/category?categoryId=" + categoryId
    );

    return response.data;
  };

  return (
    <div className="container-fluid mb-2">
      <Carousel />
      <div className="mt-2 mb-5">
        <div className="row">
          <div className="col-md-2">
            <GetAllCategories />
          </div>
          <div className="col-md-10">
            <div className="row row-cols-1 row-cols-md-4 g-4">

              {products.map((product) => {
                return <ProductCard item={product} />;
              })}
            </div>

          </div>
        </div>
      </div>
    </div>
  );
};

export default HomePage;
```

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.

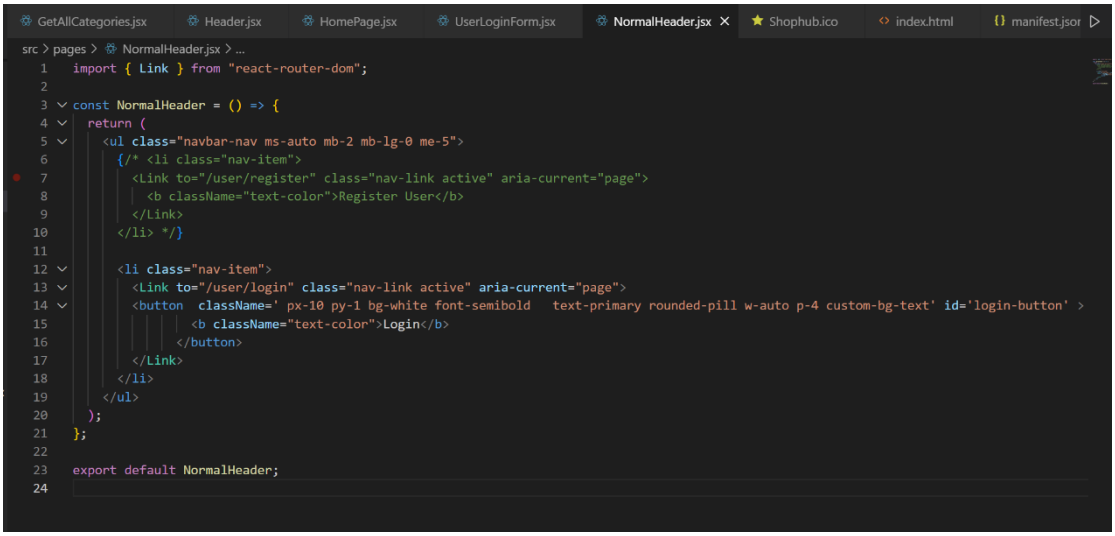
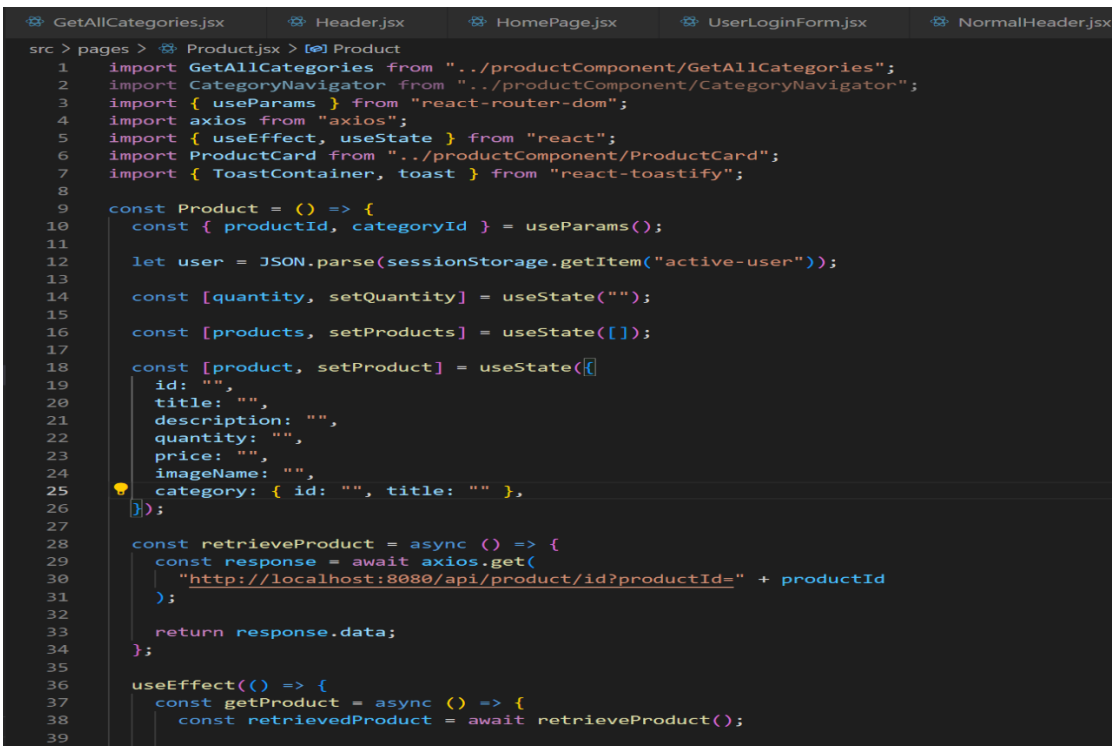


Figure:23

- **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.



```
      setProduct(retrievedProduct);
    };

    const getProductsByCategory = async () => {
      const allProducts = await retrieveProductsByCategory();
      if (allProducts) {
        setProducts(allProducts);
      }
    };

    getProduct();
    getProductsByCategory();
  }, [productId]);

  const retrieveProductsByCategory = async () => {
    const response = await axios.get(
      "http://localhost:8080/api/product/category?categoryId=" + categoryId
    );
    console.log(response.data);
    return response.data;
  };

  const saveProductToCart = (userId) => {
    fetch("http://localhost:8080/api/user/cart/add", {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        quantity: quantity,
        userId: userId,
        productId: productId,
      }),
    }).then((result) => {
      console.log("result", result);
```

```
      toast.success("Products added to Cart Successfully!!!", {
        position: "top-center",
        autoClose: 1000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        progress: undefined,
      });

      result.json().then((res) => {
        console.log("response", res);
      });
    });
  };

  const addToCart = (e) => {
    if (user == null) {
      alert("Please login to buy the products!!!");
      e.preventDefault();
    } else {
      saveProductToCart(user.id);
      setQuantity("");
      e.preventDefault();
    }
  };
```

```
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 : &amp;#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
```

```
                    <p class="ml-2 text-color">
                      <b>Stock : {product.quantity}</b>
                    </p>
                  </div>
                </div>
              </div>
            </div>

            <div className="row mt-2">
              <div className="col-sm-2"></div>

              <div className="col-sm-10">
                <h2>Related Products:</h2>
                <div className="row row-cols-1 row-cols-md-4 g-4">

                  {products.map((product) => {
                    return <ProductCard item={product} />;
                  })}
                </div>
              </div>
            </div>
          </div>
        );
      };

      export default Product;

```

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 useStae 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

src > productComponent > AddCategoryForm.jsx > ...

```jsx
import { useState } from "react";

const AddCategoryForm = () => {
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");

  const saveCategory = () => {
    let data = { title, description };

    fetch("http://localhost:8080/api/category/add", {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    }).then((result) => {
      console.warn("result", result);
      result.json().then((res) => {
        console.log("response", res);
      });
    });
  };

  return (
    <div>
      <div class="mt-2 d-flex aligns-items-center justify-content-center">
        <div
          class="card form-card border-color custom-bg"
          style={{ width: "25rem" }}
        >
          <div className="card-header bg-color text-center custom-bg-text">
            <h5 class="card-title">Add Category</h5>
          </div>
          <div class="card-body text-color">
            <form>
              <div class="mb-3">
                <label for="title" class="form-label">
                  <b>Category Title</b>
                </label>
                <input
                  type="text"
                  class="form-control"
                  id="title"
                  placeholder="enter title.."
                  onChange={(e) => {
                    setTitle(e.target.value);
                  }}
                  value={title}
                />
              </div>
              <div class="mb-3">
                <label for="description" class="form-label">
                  <b>Category Description</b>
                </label>
                <textarea
                  class="form-control"
                  id="description"
                  rows="3"
                  placeholder="enter description.."
                  onChange={(e) => {
                    setDescription(e.target.value);
                  }}
                  value={description}
                />
              </div>
```

```
                <button
                  type="submit"
                  onClick={saveCategory}
                  class="btn bg-color custom-bg-text"
                >
                  Add Category
                </button>
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  };

  export default AddCategoryForm;
```

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 > ...
import { useState, useEffect } from "react";
import axios from "axios";

const AddProductForm = () => {
  const [categories, setCategories] = useState([]);

  const retrieveAllCategories = async () => {
    const response = await axios.get("http://localhost:8080/api/category/all");
    return response.data;
  };

  useEffect(() => {
    const getAllCategories = async () => {
      const allCategories = await retrieveAllCategories();
      if (allCategories) {
        setCategories(allCategories);
      }
    };

    getAllCategories();
  }, []);

  const [selectedPhoto, setSelectedPhoto] = useState(null);
  const [product, setProduct] = useState({
    title: "",
    description: "",
    price: "",
    quantity: "",
    categoryId: "",
  });

  const handleInput = (e) => {
    setProduct({ ...product, [e.target.name]: e.target.value });
  };

```

```
const saveProduct = () => {
  const formData = new FormData();
  formData.append("image", selectedPhoto);
  formData.append("title", product.title);
  formData.append("description", product.description);
  formData.append("price", product.price);
  formData.append("quantity", product.quantity);
  formData.append("categoryId", product.categoryId);

  axios
    .post("http://localhost:8080/api/product/add", formData)
    .then((resp) => {
      let result = resp.data.data;
      alert("Product saved successfully");
    })
    .catch((error) => {
      console.log("Error", error);
      alert("Error saving product");
    });
};

return (
  <div>
    <div class="mt-2 d-flex aligns-items-center justify-content-center">
      <div
        class="card form-card border-color custom-bg"
        style={{ width: "25rem" }}
      >
        <div className="card-header bg-color custom-bg-text text-center">
          <h5 class="card-title">Add Product</h5>
        </div>
        <div class="card-body text-color">
          <form>
            <div class="mb-3">
              <label for="title" class="form-label">
                <b>Product Title</b>
              </label>
```

```
              <input
                type="text"
                class="form-control"
                id="title"
                name="title"
                onChange={handleInput}
                value={product.title}
              />
            </div>
            <div class="mb-3">
              <label for="description" class="form-label">
                <b>Product Description</b>
              </label>
              <textarea
                class="form-control"
                id="description"
                name="description"
                rows="3"
                onChange={handleInput}
                value={product.description}
              />
            </div>

            <div className="mb-3">
              <label className="form-label">
                <b>Category</b>
              </label>

              <select
                name="categoryId"
                onChange={handleInput}
                className="form-control"
              >
```

```
                    <option value="">Select Category</option>

                    {categories.map((category) => {
                      return (
                        <option value={category.id}> {category.title} </option>
                      );
                    })}
                  </select>
                </div>

                <div class="mb-3 mt-1">
                  <label for="quantity" class="form-label">
                    <b>Product Quantity</b>
                  </label>
                  <input
                    type="number"
                    class="form-control"
                    id="quantity"
                    name="quantity"
                    onChange={handleInput}
                    value={product.quantity}
                  />
                </div>

                <div class="mb-3">
                  <label for="price" class="form-label">
                    <b>Product Price</b>
                  </label>
```

src > productComponent > AddProductForm.jsx > ...

```
                    type="number"
                    class="form-control"
                    id="price"
                    name="price"
                    onChange={handleInput}
                    value={product.price}
                  />
                </div>

                <div class="mb-3">
                  <label for="formFile" class="form-label">
                    <b> Select Product Image</b>
                  </label>
                  <input
                    class="form-control"
                    type="file"
                    id="formFile"
                    name="photo"
                    value={product.photo}
                    onChange={(e) => setSelectedPhoto(e.target.files[0])}
                  />
                </div>

                <button
                  type="submit"
                  class="btn bg-color custom-bg-text"
                  onClick={saveProduct}
                >
                  Add Product
                </button>
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  };

  export default AddProductForm;
```

Figure:26

- **Category Navigator**

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

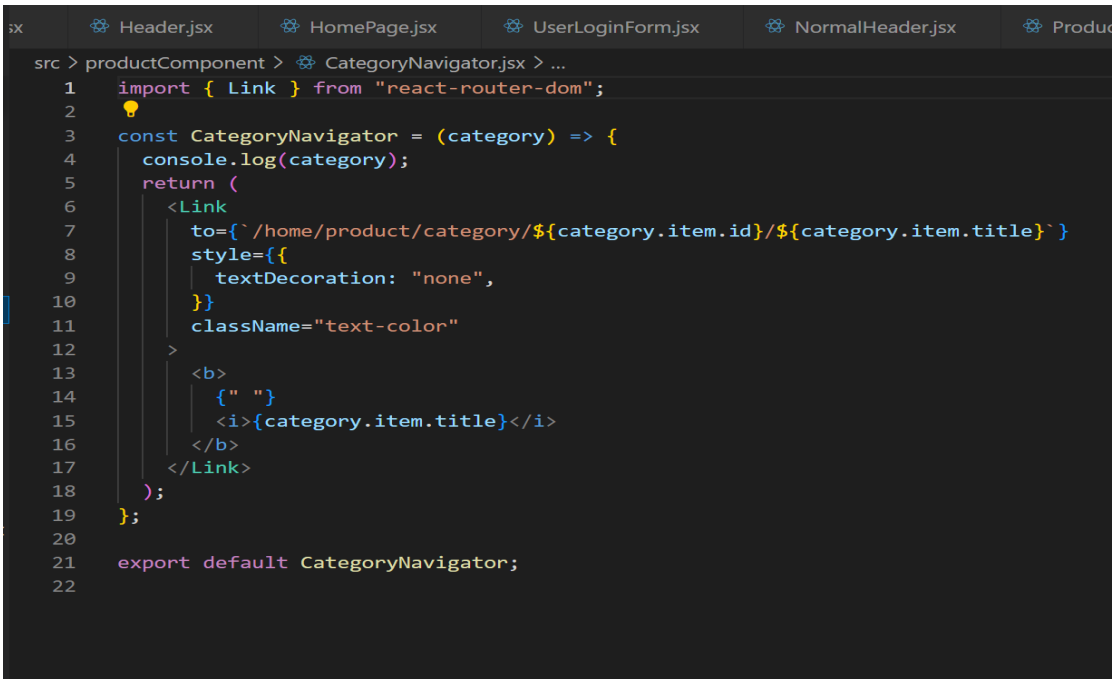
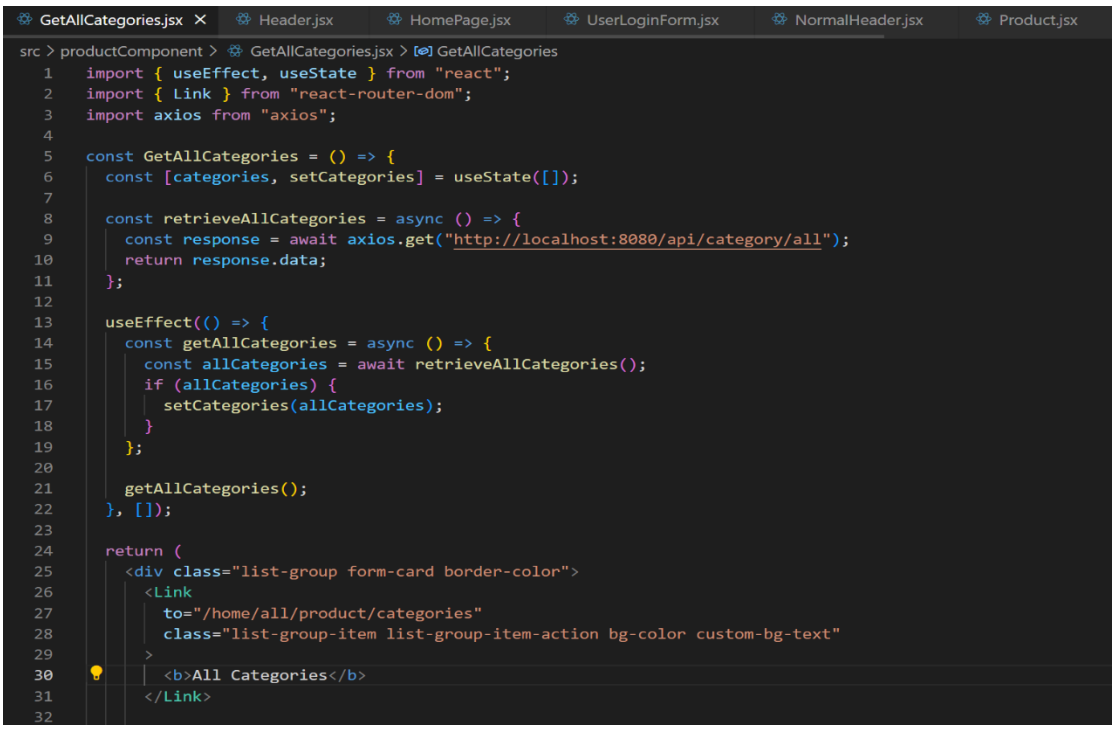


Figure:27

- **GetAllCategories**

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



```
        {categories.map((category) => {
          return (
            <Link
              to={`/home/product/category/${category.id}/${category.title}`}
              class="list-group-item list-group-item-action text-color custom-bg"
            >
              <b>{category.title}</b>
            </Link>
          );
        })}
      </div>
    );
  };

  export default GetAllCategories;
```

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. TheProductCard 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 > ...

```
import { Link } from "react-router-dom";
import CategoryNavigator from "./CategoryNavigator";

const ProductCard = (product) => {
  return (
    <div className="col">
    <div class="card border-color rounded-card card-hover product-card custom-bg h-100">
      <img
        src={"http://localhost:8080/api/product/" + product.item.imageName}
        class="card-img-top rounded mx-auto d-block m-2"
        alt="img"
        style={{
          maxHeight: "270px",
          maxWidth: "100%",
          width: "auto",
        }}
      />

      <div class="card-body text-color">
        <h5 class="card-title d-flex justify-content-between">
          <div>
            <b>{product.item.title}</b>
          </div>
          <CategoryNavigator
            item={{
              id: product.item.category.id,
              title: product.item.category.title,
            }}
          />
        </h5>
        <p className="card-text">
          <b>{product.item.description}</b>
        </p>
      </div>
```

```
      <div class="card-footer">
        <div className="text-center text-color">
          <p>
            <span>
              <h4>Price : &#8377;{product.item.price}</h4>
            </span>
          </p>
        </div>
        <div className="d-flex justify-content-between">
          <Link
            to={`/product/${product.item.id}/category/${product.item.category.id}`}
            className="btn bg-color custom-bg-text"
          >
            Add to Cart
          </Link>

          <p class="text-color">
            <b>
              <i>Stock :</i> {product.item.quantity}
            </b>
          </p>
        </div>
      </div>
    </div>
    </div>
  );
};

export default ProductCard;

```

Figure:29

Src/userComponents

- **AdduseFrom**

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

```
src > userComponent > ⚛ AddUserForm.jsx > ...
import { useState } from "react";
import "react-toastify/dist/ReactToastify.css";
import { ToastContainer, toast } from "react-toastify";

const AddUserForm = () => {
  const [user, setUser] = useState({
    firstName: "",
    lastName: "",
    emailId: "",
    password: "",
    phoneNo: "",
    street: "",
    city: "",
    pincode: "",
    role: "",
  });

  const handleUserInput = (e) => {
    setUser({ ...user, [e.target.name]: e.target.value });
  };

  const saveUser = (event) => {
    event.preventDefault();
    fetch("http://localhost:8080/api/user/register", {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      body: JSON.stringify(user),
    }).then((result) => {
      console.log("******near toast thing");
      toast.success("Registered Successfully!!!", {
        position: "top-center",
        autoClose: 1000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
```

```
        progress: undefined,
      });
      console.warn("result", result);
      result
        .json()
        .then((res) => {
          console.log("response", res);
        })
        .catch((error) => {
          console.log("******", error);
          console.log(error);
        });
    });
  };

  return (
    <div>
      <div class="mt-2 d-flex aligns-items-center justify-content-center ms-2 me-2 mb-2">
        <div
          class="card form-card border-color text-color custom-bg"
          style={{ width: "25rem" }}
        >
          <div className="card-header bg-color custom-bg-text text-center">
            <h5 class="card-title">Add User</h5>
          </div>
          <div class="card-body">
            <form onSubmit={saveUser}>
              <div class="mb-3 text-color">
                <label for="role" class="form-label">
                  <b>User Role</b>
                </label>
                <select
                  onChange={handleUserInput}
                  className="form-control"
                  name="role"
                >
                  <option value="0">Select Role</option>
                  <option value="Admin"> Admin </option>
                  <option value="Customer"> Customer </option>
                  <option value="Delivery"> Delivery Person </option>
                </select>
              </div>

              <div class="mb-3 text-color">
                <label for="title" class="form-label">
                  <b> First Name</b>
                </label>
                <input
                  type="text"
                  class="form-control"
                  id="firstName"
                  name="firstName"
                  onChange={handleUserInput}
                  value={user.firstName}
                />
              </div>
              <div class="mb-3 text-color">
                <label for="description" class="form-label">
                  <b>Last Name</b>
                </label>
                <input
                  type="text"
                  class="form-control"
                  id="lastName"
                  name="lastName"
                  onChange={handleUserInput}
                  value={user.lastName}
                />
              </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                     class="form-control"
117                     id="emailId"
118                     name="emailId"
119                     onChange={handleUserInput}
120                     value={user.emailId}
121                   />
122                 </div>
123
124                 <div class="mb-3 mt-1">
125                   <label for="quantity" class="form-label">
126                     Password
127                   </label>
128                   <input
129                     type="password"
130                     class="form-control"
131                     id="password"
132                     name="password"
133                     onChange={handleUserInput}
134                     value={user.password}
135                   />
136                 </div>
137
138                 <div class="mb-3">
139                   <label for="price" class="form-label">
140                     <b>Mobile No</b>
141                   </label>
142                   <input
143                     type="number"
144                     class="form-control"
145                     id="phoneNo"
146                     name="phoneNo"
```

```
147                     onChange={handleUserInput}
148                     value={user.phoneNo}
149                   />
150                 </div>
151
152                 <div class="mb-3">
153                   <label for="description" class="form-label">
154                     <b> Street</b>
155                   </label>
156                   <textarea
157                     class="form-control"
158                     id="street"
159                     name="street"
160                     rows="3"
161                     onChange={handleUserInput}
162                     value={user.street}
163                   />
164                 </div>
165
166                 <div class="mb-3">
167                   <label for="price" class="form-label">
168                     <b>City</b>
169                   </label>
170                   <input
171                     type="text"
172                     class="form-control"
173                     id="city"
174                     name="city"
175                     onChange={handleUserInput}
176                     value={user.city}
177                   />
178                 </div>
```

```
                <div class="mb-3">
                  <label for="pincode" class="form-label">
                    <b>Pincode</b>
                  </label>
                  <input
                    type="number"
                    class="form-control"
                    id="pincode"
                    name="pincode"
                    onChange={handleUserInput}
                    value={user.pincode}
                  />
                </div>

                <input
                  type="submit"
                  class="btn bg-color custom-bg-text"
                  value="Register User"
                />

                <ToastContainer />
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  };

  export default AddUserForm;
```

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.j

src > userComponent > AdminLoginPage.jsx > ...

```jsx
import { useState } from "react";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import { useNavigate } from "react-router-dom";

const AdminLoginPage = () => {
  const [loginRequest, setLoginRequest] = useState({
    emailId: "",
    password: "",
  });

  const [response, setResponse] = useState({});

  const handleUserInput = (e) => {
    setLoginRequest({ ...loginRequest, [e.target.name]: e.target.value });
  };

  const loginAction = (e) => {
    fetch("http://localhost:8080/api/user/admin/login", {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      body: JSON.stringify(loginRequest),
    }).then((result) => {
      console.log("result", result);
      result.json().then((res) => {
        sessionStorage.setItem("active-admin", JSON.stringify(res));
        toast.success("logged in successfully!!!", {
          position: "top-center",
          autoClose: 1000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      });
    });

    e.preventDefault();
  };

  return (
    <div>
      <div className="mt-2 d-flex aligns-items-center justify-content-center">
        <div className="card form-card border-color" style={{ width: "25rem" }}>
          <div className="card-header bg-color text-center">
            <h4 className="card-title">Admin Login</h4>
          </div>
          <div className="card-body">
            <form>
              <div className="mb-3">
                <label for="emailId" class="form-label custom-bg-text">
                  Email Id
                </label>
                <input
                  type="email"
                  className="form-control"
                  id="emailId"
                  name="emailId"
                  onChange={handleUserInput}
                  value={loginRequest.emailId}
                />
              </div>
              <div className="mb-3">
                <label for="password" className="form-label custom-bg-text">
                  Password
                </label>
                <input
                  type="password"
                  className="form-control"
                  id="password"
                  name="password"
                  onChange={handleUserInput}
                  value={loginRequest.password}
                  autoComplete="on"
```

```
                    autoComplete="on"
                  />
                </div>

                <button
                  type="submit"
                  className="btn custom-bg text-color"
                  onClick={loginAction}
                >
                  Login
                </button>
                <ToastContainer />
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  };

  export default AdminLoginPage;

```

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 > ...
import { useState, useEffect } from "react";
import axios from "axios";
import React from "react";

const AllOrders = () => {
  const [allOrderData, setAllOrderData] = useState([]);

  useEffect(() => {
    const getAllOrder = async () => {
      const allOrder = await retrieveAllOrder();
      if (allOrder) {
        setAllOrderData(allOrder);
      }
    };

    getAllOrder();
  }, []);

  const retrieveAllOrder = async () => {
    const response = await axios.get(
      "http://localhost:8080/api/user/admin/allorder"
    );
    console.log(response.data);
    return response.data;
  };

  return (
    <div className="mt-3">
      <div
        className="card form-card ms-2 me-2 mb-5 custom-bg border-color "
        style={{
          height: "45rem",
        }}
      >
        <div className="card-header custom-bg-text text-center bg-color">
          <h2>All Orders</h2>
        </div>
        <div
          className="card-body"
```

```
          className="card-body"
          style={{
            overflowY: "auto",
          }}
        >
          <div className="table-responsive">
            <table className="table table-hover text-color text-center">
              <thead className="table-bordered border-color bg-color custom-bg-text">
                <tr>
                  <th scope="col">Order Id</th>
                  <th scope="col">Product</th>
                  <th scope="col">Name</th>
                  <th scope="col">Description</th>
                  <th scope="col">Quantity</th>
                  <th scope="col">Total Price</th>
                  <th scope="col">Customer Name</th>
                  <th scope="col">Street</th>
                  <th scope="col">City</th>
                  <th scope="col">Pin code</th>
                  <th scope="col">Mobile No.</th>
                  <th scope="col">Order Date</th>
                  <th scope="col">Delivery Date</th>
                  <th scope="col">Delivery Status</th>
                  <th scope="col">Delivery Person</th>
                  <th scope="col">Delivery Mobile No</th>
                </tr>
              </thead>
              <tbody>
                {allOrderData.map((orderData) => {
                  return (
                    <tr>
                      <td>
                        <b>{orderData.orderId}</b>
                      </td>
                      <td>
                        <img
                          src={
                            "http://localhost:8080/api/product/" +
                            orderData.productImage
                          }
                          class="img-fluid"
                          alt="product_pic"
                          style={{
                            maxWidth: "90px",
                          }}
                        />
                      </td>
                      <td>
                        <b>{orderData.productName}</b>
                      </td>
                      <td>
                        <b>{orderData.productDescription}</b>
                      </td>
                      <td>
                        <b>{orderData.quantity}</b>
                      </td>
                      <td>
                        <b>{orderData.totalPrice}</b>
                      </td>
                      <td>
                        <b>{orderData.userName}</b>
                      </td>
                      <td>
                        <b>{orderData.address.street}</b>
                      </td>

                      <td>
                        <b>{orderData.address.city}</b>
                      </td>
                      <td>
                        <b>{orderData.address.pincode}</b>
                      </td>
```

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.

```
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>
```

```
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>
```

```
                                    </tr>
                                );
                            })}
                        </tbody>
                    </table>
                </div>
            </div>
            <div className="card-footer custom-bg">
                <div className="float-right">
                    <div
                        className="text-color me-2"
                        style={{
                            textAlign: "right",
                        }}
                    >
                        <h5>Total Price: &#8377; {totatPrice}/-</h5>
                    </div>

                    <div className="float-end me-2">
                        <button
                            type="submit"
                            className="btn bg-color custom-bg-text mb-3"
                            onClick={checkout}
                        >
                            Checkout
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </div>
    );
};

export default MyCart;

```

Figure:33

- **MyOrders**

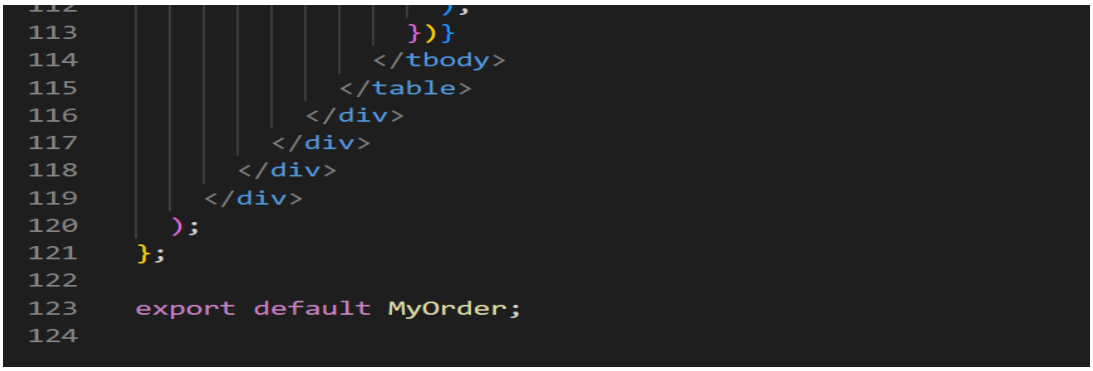
  The MyOrders component takes a userId prop, representing the ID of the user whose orders are being displayed.

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

```
src > userComponent > MyOrder.jsx > ...
import { useState, useEffect } from "react";
import axios from "axios";
import React from "react";

const MyOrder = () => {
  const user = JSON.parse(sessionStorage.getItem("active-user"));
  const [myOrderData, setMyOrderData] = useState([]);

  useEffect(() => {
    const getMyOrder = async () => {
      const myOrder = await retrieveMyOrder();
      if (myOrder) {
        console.log("my order data is present :)");

        setMyOrderData(myOrder);
      }
    };

    getMyOrder();
  }, []);

  const retrieveMyOrder = async () => {
    const response = await axios.get(
      "http://localhost:8080/api/user/myorder?userId=" + user.id
    );
    console.log(response.data);
    return response.data;
  };

  return (
    <div className="mt-3">
      <div
        className="card form-card ms-2 me-2 mb-5 custom-bg"
        style={{
          height: "45rem",
        }}
      >
        <div className="card-header text-center bg-color custom-bg-text">
          <h2>My Orders</h2>
```

```
          <h2>My Orders</h2>
        </div>
        <div
          className="card-body"
          style={{
            overflowY: "auto",
          }}
        >
          <div className="table-responsive">
            <table className="table table-hover custom-bg-text text-center">
              <thead className="bg-color table-bordered border-color">
                <tr>
                  <th scope="col">Order Id</th>
                  <th scope="col">Product</th>
                  <th scope="col">Name</th>
                  <th scope="col">Description</th>
                  <th scope="col">Quantity</th>
                  <th scope="col">Total Price</th>
                  <th scope="col">Order Date</th>
                  <th scope="col">Delivery Date</th>
                  <th scope="col">Delivery Status</th>
                  <th scope="col">Delivery Person</th>
                  <th scope="col">Delivery Mobile No</th>
                </tr>
              </thead>
              <tbody className="text-color">
                {myOrderData.map((orderData) => {
                  return (
                    <tr>
                      <td>
                        <b>{orderData.orderId}</b>
                      </td>
                      <td>
                        <img
                          src={
                            "http://localhost:8080/api/product/" +
                            orderData.productImage
                          }
```

```
                          }
                          class="img-fluid"
                          alt="product_pic"
                          style={{
                            maxWidth: "90px",
                          }}
                        />
                      </td>
                      <td>
                        <b>{orderData.productName}</b>
                      </td>
                      <td>
                        <b>{orderData.productDescription}</b>
                      </td>
                      <td>
                        <b>{orderData.quantity}</b>
                      </td>
                      <td>
                        <b>{orderData.totalPrice}</b>
                      </td>
                      <td>
                        <b>{orderData.orderDate}</b>
                      </td>
                      <td>
                        <b>{orderData.deliveryDate}</b>
                      </td>
                      <td>
                        <b>{orderData.deliveryStatus}</b>
                      </td>
                      <td>
                        <b>{orderData.deliveryPersonName}</b>
                      </td>
                      <td>
                        <b>{orderData.deliveryPersonContact}</b>
                      </td>
                    </tr>
                  );
```



```
                    })}
                  </tbody>
                </table>
              </div>
            </div>
          </div>
        </div>
      );
    };

    export default MyOrder;
```

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 > ...
import { useState } from "react";

const RegisterAdminForm = () => {
  const [user, setUser] = useState({
    firstName: "",
    lastName: "",
    emailId: "",
    password: "",
    phoneNo: "",
    street: "",
    city: "",
    pincode: "",
  });

  const handleUserInput = (e) => {
    setUser({ ...user, [e.target.name]: e.target.value });
  };

  const sav
    fetch("http://localhost:8080/api/user/admin/register", {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      body: JSON.stringify(user),
    }).then((result) => {
      console.warn("result", result);
      result.json().then((res) => {
        console.log("response", res);
      });
    });
  };

See Real World Examples From GitHub
Follow link (ctrl + click)
```

```
return (
  <div>
    <div class="mt-2 d-flex aligns-items-center justify-content-center ms-2 me-2 mb-2">
      <div class="card form-card border-color" style={{ width: "25rem" }}>
        <div className="card-header bg-color">
          <h5 class="card-title">Add Admin</h5>
        </div>
        <div class="card-body">
          <form>
            <div class="mb-3">
              <label for="title" class="form-label">
                First Name
              </label>
              <input
                type="text"
                class="form-control"
                id="firstName"
                name="firstName"
                onChange={handleUserInput}
                value={user.firstName}
              />
            </div>
            <div class="mb-3">
              <label for="description" class="form-label">
                Last Name
              </label>
              <input
                type="text"
                class="form-control"
                id="lastName"
                name="lastName"
                onChange={handleUserInput}
                value={user.lastName}
              />
            </div>
```

```
            <div className="mb-3">
              <label className="form-label">Email Id</label>
              <input
                (property) class: string
                class="form-control"
                id="emailId"
                name="emailId"
                onChange={handleUserInput}
                value={user.emailId}
              />
            </div>

            <div class="mb-3 mt-1">
              <label for="quantity" class="form-label">
                Password
              </label>
              <input
                type="password"
                class="form-control"
                id="password"
                name="password"
                onChange={handleUserInput}
                value={user.password}
              />
            </div>

            <div class="mb-3">
              <label for="price" class="form-label">
                Mobile No
              </label>
              <input
                type="number"
                class="form-control"
                id="phoneNo"
                name="phoneNo"
                onChange={handleUserInput}
                value={user.phoneNo}
              />
            </div>
```

```
                <div class="mb-3">
                  <label for="description" class="form-label">
                    Street
                  </label>
                  <textarea
                    class="form-control"
                    id="street"
                    name="street"
                    rows="3"
                    onChange={handleUserInput}
                    value={user.street}
                  />
                </div>

                <div class="mb-3">
                  <label for="price" class="form-label">
                    City
                  </label>
                  <input
                    type="text"
                    class="form-control"
                    id="city"
                    name="city"
                    onChange={handleUserInput}
                    value={user.city}
                  />
                </div>

                <div class="mb-3">
                  <label for="pincode" class="form-label">
                    Pincode
                  </label>
                  <input
                    type="number"
                    class="form-control"
                    id="pincode"
                    name="pincode"
                    onChange={handleUserInput}
                    value={user.pincode}
                  />
                </div>

                <button
                  type="submit"
                  class="btn custom-bg text-color"
                  onClick={saveUser}
                >
                  Register
                </button>
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  };

  export default RegisterAdminForm;
```

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 > ...

```jsx
import { useState, useEffect } from "react";
import axios from "axios";
import "react-datepicker/dist/react-datepicker.css";

const SearchOrder = () => {
  const [allOrderData, setAllOrderData] = useState([]);
  const [orderId, setOrderId] = useState("");

  const deliveryStatus = ["Delivered", "On the Way", "Processing"];
  const deliveryTime = ["Morning", "Afternoon", "Evening", "Night"];

  const [orderDeliveryStatus, setOrderDeliveryStatus] = useState({
    orderId: "",
    deliveryStatus: "",
    deliveryTime: "",
    deliveryDate: "",
  });

  const handleOrderDelivery = (e) => {
    setOrderDeliveryStatus({
      ...orderDeliveryStatus,
      [e.target.name]: e.target.value,
    });
  };

  const getAllOrder = async () => {
    const allOrder = await retrieveAllOrder();
    if (allOrder) {
      setAllOrderData(allOrder);
    }
  };

  const retrieveAllOrder = async () => {
    const response = await axios.get(
      "http://localhost:8080/api/user/admin/showorder?orderId=" + orderId
    );
    console.log(response.data);
    return response.data;
  };
```

```jsx

  const searchOrderById = (e) => {
    getAllOrder();
    setOrderId("");
    e.preventDefault();
  };

  const updateDeliveryStatus = (e) => {
    console.log("CLICKED DELIVERY STATUS UPDATED");
    fetch("http://localhost:8080/api/user/admin/order/deliveryStatus/update", {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",
      },
      body: JSON.stringify(orderDeliveryStatus),
    }).then((result) => {
      console.log("result", result);
      result.json().then((res) => {
        console.log("response", res);
        setAllOrderData({
          orderId: "",
          deliveryStatus: "",
          deliveryTime: "",
          deliveryDate: "",
        });

        setAllOrderData(res);
      });
    });
```

```
    e.preventDefault();
  };

  return (
    <div>
      <d  (property) React.HTMLAttributes<HTMLDivElement>.className?: string | undefined
        className="card form-card mt-1 ms-2 me-2 mb-2 custom-bg"
        style={{
          height: "35rem",
        }}
      >
        <div className="card-header text-center bg-color custom-bg-text">
          <h4>Searh Customer Orders</h4>
        </div>
        <div
          className="card-body"
          style={{
            overflowY: "auto",
          }}
        >
          <form class="row g-3">
            <div class="col-auto">
              <input
                type="text"
                class="form-control"
                id="inputPassword2"
                placeholder="Enter Order Id..."
                onChange={(e) => setOrderId(e.target.value)}
                value={orderId}
              />
            </div>
            <div class="col-auto">
              <button
                type="submit"
                class="btn bg-color custom-bg-text mb-3"
                onClick={searchOrderById}
              >
                Search
              </button>
            </div>
          </form>
          <div className="table-responsive">
            <table className="table table-hover custom-bg-text text-center">
              <thead className="bg-color table-bordered border-color">
                <tr>
                  <th scope="col">Order Id</th>
                  <th scope="col">Product</th>
                  <th scope="col">Name</th>
                  <th scope="col">Description</th>
                  <th scope="col">Quantity</th>
                  <th scope="col">Total Price</th>
                  <th scope="col">Customer Name</th>
                  <th scope="col">Street</th>
                  <th scope="col">City</th>
                  <th scope="col">Pin code</th>
                  <th scope="col">Mobile No.</th>
                  <th scope="col">Order Date</th>
                  <th scope="col">Delivery Date</th>
                  <th scope="col">Delivery Status</th>
                  <th scope="col">Delivery Person</th>
                  <th scope="col">Delivery Mobile No</th>
                </tr>
              </thead>
```

```
<tbody className="text-color">
  {allOrderData.map((orderData) => {
    return (
      <tr>
        <td>
          <b>{orderData.orderId}</b>
        </td>
        <td>
          <img
            src={
              "http://localhost:8080/api/product/" +
              orderData.productImage
            }
            class="img-fluid"
            alt="product_pic"
            style={{
              maxWidth: "90px",
            }}
          />
        </td>
        <td>
          <b>{orderData.productName}</b>
        </td>
        <td>
          <b>{orderData.productDescription}</b>
        </td>
        <td>
          <b>{orderData.quantity}</b>
        </td>
        <td>
          <b>{orderData.totalPrice}</b>
        </td>
        <td>
          <b>{orderData.userName}</b>
        </td>
        <td>
          <b>{orderData.address.street}</b>
        </td>
```

```
        <td>
          <b>{orderData.address.city}</b>
        </td>
        <td>
          <b>{orderData.address.pincode}</b>
        </td>
        <td>
          <b>{orderData.userPhone}</b>
        </td>
        <td>
          <b>{orderData.orderDate}</b>
        </td>
        <td>
          <b>{orderData.deliveryDate}</b>
        </td>
        <td>
          <b>{orderData.deliveryStatus}</b>
        </td>
        <td>
          <b>{orderData.deliveryPersonName}</b>
        </td>
        <td>
          <b>{orderData.deliveryPersonContact}</b>
        </td>
      </tr>
    );
  })}
</tbody>
</table>
</div>
</div>
</div>
```

```jsx
<div>
  <div className="card form-card ms-2 me-2 mb-2 custom-bg">
    <div className="card-header text-center bg-color custom-bg-text">
      <h4>Update Delivery Status</h4>
    </div>
    <div className="card-body">
      <form class="row g-3">
        <div class="col-auto">
          <label className="text-color">
            <b>Order Id</b>
          </label>
          <input
            type="text"
            class="form-control"
            id="inputPassword2"
            placeholder="Enter Order Id..."
            name="orderId"
            onChang const handleOrderDelivery: (e: any) => void
            value={handleOrderDelivery.orderId}
          />
        </div>
        <div class="col-auto">
          <label className="text-color">
            <b>Select Delivery Date</b>
          </label>
          <input
            type="date"
            class="form-control"
            id="inputPassword2"
            name="deliveryDate"
            placeholder="dd-mm-yyyy"
            min="1997-01-01"
            max="2030-12-31"
            value={handleOrderDelivery.deliveryDate}
            onChange={handleOrderDelivery}
          />
        </div>
```

```jsx
        </div>
        <div class="col-auto">
          <label className="text-color">
            <b>Delivery Time</b>
          </label>

          <select
            name="deliveryTime"
            value={handleOrderDelivery.deliveryTime}
            onChange={handleOrderDelivery}
            className="form-control"
          >
            <option value="">Select Delivery Time</option>

            {deliveryTime.map((time) => {
              return <option value={time}> {time} </option>;
            })}
          </select>
        </div>
        <div class="col-auto">
          <label className="text-color">
            <b>Delivery Status</b>
          </label>
          <select
            name="deliveryStatus"
            value={handleOrderDelivery.deliveryStatus}
            onChange={handleOrderDelivery}
            className="form-control"
          >
            <option value="">Select Delivery Status</option>

            {deliveryStatus.map((status) => {
              return <option value={status}> {status} </option>;
            })}
          </select>
        </div>
```

```
                </div>
                <div class="col-auto">
                  <button
                    type="submit"
                    class="btn bg-color custom-bg-text mt-4"
                    onClick={updateDeliveryStatus}
                  >
                    Update Delivery Status
                  </button>
                </div>
              </form>
            </div>
          </div>
        </div>
      </div>
    );
  };

  export default SearchOrder;

```

Figure:36

- **UerLoginForm**

  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          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 aligns-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 class="mb-3 text-color">
72                  <label for="role" class="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
```

```
            <div className="mb-3 text-color">
              <label for="emailId" class="form-label">
                <b>Email Id</b>
              </label>
              <input
                type="email"
                className="form-control"
                id="emailId"
                name="emailId"
                onChange={handleUserInput}
                value={loginRequest.emailId}
              />
            </div>
            <div className="mb-3 text-color">
              <label for="password" className="form-label">
                <b>Password</b>
              </label>
              <input
                type="password"
                className="form-control"
                id="password"
                name="password"
                onChange={handleUserInput}
                value={loginRequest.password}
                autoComplete="on"
              />
            </div>
            <button
              type="submit"
              className="btn bg-color custom-bg-text w-50 p-2.5 rounded-pill"
              onClick={loginAction}
            >
              Login
            </button>
```

```
            <Link to="/user/register" className="nav-link active m-1" aria-current="page"> New User?
          <b className="text-color">Register Here</b>
           </Link>
             <ToastContainer />
            </form>
          </div>
        </div>
      </div>
    </div>
  );
};

export default UserLoginForm;

```

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 > ...

```jsx
import Carousel from "./Carousel";
import GetAllCategories from "../productComponent/GetAllCategories";
import ProductCard from "../productComponent/ProductCard";
import axios from "axios";
import { useState, useEffect } from "react";
import { useParams } from "react-router-dom";

const HomePage = () => {
  const [products, setProducts] = useState([]);

  const { categoryId } = useParams();

  useEffect(() => {
    const getAllProducts = async () => {
      const allProducts = await retrieveAllProducts();
      if (allProducts) {
        setProducts(allProducts);
      }
    };

    const getProductsByCategory = async () => {
      const allProducts = await retrieveProductsByCategory();
      if (allProducts) {
        setProducts(allProducts);
      }
    };

    if (categoryId == null) {
      console.log("Category Id is null");
      getAllProducts();
    } else {
      console.log("Category Id is NOT null");
      getProductsByCategory();
    }
  }, [categoryId]);

  const retrieveAllProducts = async () => {
    const response = await axios.get("http://localhost:8080/api/product/all");

    return response.data;
  };

  const retrieveProductsByCategory = async () => {
    const response = await axios.get(
      "http://localhost:8080/api/product/category?categoryId=" + categoryId
    );

    return response.data;
  };

  return (
    <div className="container-fluid mb-2">
      <Carousel />
      <div className="mt-2 mb-5">
        <div className="row">
          <div className="col-md-2">
            <GetAllCategories />
          </div>
          <div className="col-md-10">
            <div className="row row-cols-1 row-cols-md-4 g-4">

              {products.map((product) => {
                return <ProductCard item={product} />;
              })}
            </div>

          </div>
        </div>
      </div>
    </div>
  );
};

export default HomePage;
```

```
                <Link to="/user/register" className="nav-link active m-1" aria-current="page"> New User?
              <b className="text-color">Register Here</b>
               </Link>
                 <ToastContainer />
               </form>
              </div>
            </div>
          </div>
        </div>
      );
    };

    export default UserLoginForm;

```

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 > ...
import Header from "./pages/Header";
import AddProductForm from "./productComponent/AddProductForm";
import { Routes, Route } from "react-router-dom";
import ContactUs from "./pages/ContactUs";
import AboutUs from "./pages/AboutUs";
import AddCategoryForm from "./productComponent/AddCategoryForm";
import HomePage from "./pages/HomePage";
import Product from "./pages/Product";
import AddUserForm from "./userComponent/AddUserForm";
import UserLoginForm from "./userComponent/UserLoginForm";
import MyCart from "./userComponent/MyCart";
import AddCardDetails from "./pages/AddCardDetails";
import MyOrder from "./userComponent/MyOrder";
import AllOrders from "./userComponent/AllOrders";
import SearchOrder from "./userComponent/SearchOrder";
import RegisterAdminForm from "./userComponent/RegisterAdminForm";
import AdminLoginPage from "./userComponent/AdminLoginPage";
import AddDeliveryPerson from "./userComponent/AddDeliveryPerson";
import DeliveryPersonLogin from "./userComponent/DeliveryPersonLogin";
import AssignDeliveryToOrders from "./userComponent/AssignDeliveryToOrders";
import MyDeliveries from "./userComponent/MyDeliveries";

function App() {
  return (
    <div>
      <Header />
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/home" element={<HomePage />} />
        <Route path="/home/all/product/categories" element={<HomePage />} />
        <Route path="contact" element={<ContactUs />} />
        <Route path="about" element={<AboutUs />} />
        <Route path="addproduct" element={<AddProductForm />} />
        <Route path="addcategory" element={<AddCategoryForm />} />
        <Route path="/product" element={<Product />} />
        <Route path="/user/register" element={<AddUserForm />} />
        <Route path="/user/login" element={<UserLoginForm />} />
        <Route path="/user/admin/register" element={<RegisterAdminForm />} />
        <Route path="/user/admin/login" element={<AdminLoginPage />} />
```

```
            <Route path="/user/admin/login" element={<AdminLoginPage />} />
            <Route
              path="/user/deliveryperson/register"
              element={<AddDeliveryPerson />}
            />
            <Route
              path="/user/deliveryperson/login"
              element={<DeliveryPersonLogin />}
            />
            <Route
              path="/home/product/category/:categoryId/:categoryName"
              element={<HomePage />}
            />
            <Route
              path="/product/:productId/category/:categoryId"
              element={<Product />}
            />
            <Route path="/user/mycart" element={<MyCart />} />
            <Route path="/user/order/payment" element={<AddCardDetails />} />
            <Route path="/user/myorder" element={<MyOrder />} />
            <Route path="/user/admin/allorder" element={<AllOrders />} />
            <Route path="/user/admin/searchOrder" element={<SearchOrder />} />
            <Route
              path="/user/admin/assigndelivery"
              element={<AssignDeliveryToOrders />}
            />
            <Route path="/user/delivery/myorders" element={<MyDeliveries />} />
          </Routes>
        </div>
    );
  }

  export default App;

```

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   code {
10    font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
11      monospace;
12  }
13
14  .custom-bg {
15    background-color: #FDF1E7;
16  }
17
18  .text-color {
19    color: #25493F;
20  }
21
22  .border-color {
23    border-color: #25493F;
24  }
25
26  .custom-bg-text {
27    color: #FDF1E7;
28  }
29
30  .bg-color {
31    background-color: #25493F;
32  }
33
34  nt, 'Segoe UI', 'Roboto', 'Oxygen',
35   Sans', 'Helvetica Neue',
```

```
35
36    .rounded-card {
37      padding: 1.5em .5em .5em;
38      border-radius: 2em;
39      box-shadow: 0 5px 10px rgba(0,0,0,.2);
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 maintaind 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

B**rand 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:

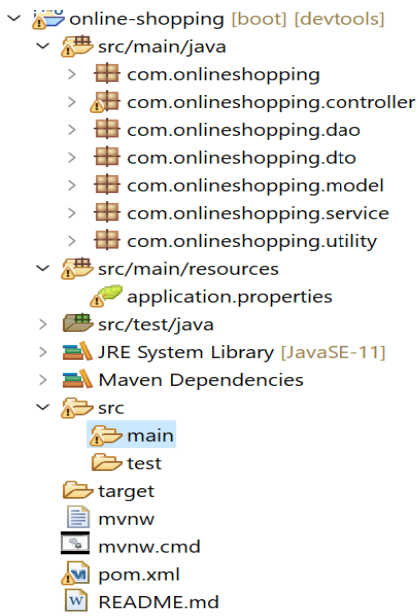


Figure:43

- **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.

```java
package com.onlineshopping;

import org.springframework.boot.SpringApplication;

@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.

```java
package com.onlineshopping.controller;

import java.util.ArrayList;

@RestController
@RequestMapping("api/user/")
@CrossOrigin(origins = "http://localhost:3000")
public class CartController {

    @Autowired
    private CartDao cartDao;

    @Autowired
    private UserDao userDao;

    @Autowired
    private ProductDao productDao;

    ObjectMapper objectMapper = new ObjectMapper();

    @PostMapping("cart/add")
    public ResponseEntity add(@RequestBody AddToCartRequest addToCartRequest) {

        System.out.println("request came for ADD PRODUCT TO CART");
        System.out.println(addToCartRequest);
        Optional<User> optionalUser = userDao.findById(addToCartRequest.getUserId());
        User user = null;
        if(optionalUser.isPresent()) {
            user = optionalUser.get();
        }

        Optional<Product> optionalProduct = productDao.findById(addToCartRequest.getProductId());
        Product product = null;
        if(optionalProduct.isPresent()) {
            product = optionalProduct.get();
        }

        Cart cart = new Cart();
        cart.setProduct(product);
        cart.setQuantity(addToCartRequest.getQuantity());
        cart.setUser(user);

        cartDao.save(cart);


        return new ResponseEntity(HttpStatus.OK);

    }

    @GetMapping("mycart")
    public ResponseEntity getMyCart(@RequestParam("userId") int userId) throws JsonProcessingException {

        System.out.println("request came for MY CART for USER ID : "+userId);

        List<CartDataResponse> cartDatas = new ArrayList<>();

        List<Cart> userCarts = cartDao.findByUser_id(userId);

        double totalCartPrice = 0;

        for (Cart cart : userCarts) {
            CartDataResponse cartData = new CartDataResponse();
            cartData.setCartId(cart.getId());
            cartData.setProductDescription(cart.getProduct().getDescription());
            cartData.setProductName(cart.getProduct().getTitle());
            cartData.setProductImage(cart.getProduct().getImageName());
            cartData.setQuantity(cart.getQuantity());
            cartData.setProductId(cart.getProduct().getId());

            cartDatas.add(cartData);

            double productPrice = Double.parseDouble(cart.getProduct().getPrice().toString());

            totalCartPrice =  totalCartPrice + (cart.getQuantity() * productPrice);

        }
```

```
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.

```java
package com.onlineshopping.controller;

import java.util.List;

@RestController
@RequestMapping("api/category")
@CrossOrigin(origins = "http://localhost:3000")
public class CategoryController {

    @Autowired
    private CategoryDao categoryDao;

    @GetMapping("all")
    public ResponseEntity<List<Category>> getAllCategories() {

        System.out.println("request came for getting all categories");

        List<Category> categories = this.categoryDao.findAll();

        ResponseEntity<List<Category>> response = new ResponseEntity<>(categories, HttpStatus.OK);

        System.out.println("response sent");

        return response;

    }

    @PostMapping("add")
    public ResponseEntity add(@RequestBody Category category) {

        System.out.println("request came for add category");

        Category c = categoryDao.save(category);

        if(c != null) {
            System.out.println("response sent");
            return new ResponseEntity( c ,HttpStatus.OK);
        }

        else {
            System.out.println("response sent");
            return new ResponseEntity("Failed to add category!",HttpStatus.INTERNAL_SERVER_ERROR);
        }

    }
```

Figure:47

Order Controller:

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

```java
package com.onlineshopping.controller;

import java.time.LocalDateTime;

@RestController
@RequestMapping("api/user/")
@CrossOrigin(origins = "*", allowedHeaders = "*")
public class OrderController {

    @Autowired
    private OrderDao orderDao;

    @Autowired
    private CartDao cartDao;

    @Autowired
    private UserDao userDao;

    @Autowired
    private ProductDao productDao;

    private ObjectMapper objectMapper = new ObjectMapper();

    @PostMapping("order")
    public ResponseEntity customerOrder(@RequestParam("userId") int userId) throws JsonProcessingException {

        System.out.println("request came for ORDER FOR CUSTOMER ID : " + userId);

        String orderId = Helper.getAlphaNumericOrderId();

        List<Cart> userCarts = cartDao.findByUser_id(userId);

        LocalDateTime currentDateTime = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
        String formatDateTime = currentDateTime.format(formatter);

        for (Cart cart : userCarts) {

            Orders order = new Orders();
            order.setOrderId(orderId);
            order.setUser(cart.getUser());
            order.setProduct(cart.getProduct());
            order.setQuantity(cart.getQuantity());
            order.setOrderDate(formatDateTime);
            order.setDeliveryDate(DeliveryStatus.PENDING.value());
            order.setDeliveryStatus(DeliveryStatus.PENDING.value());
            order.setDeliveryTime(DeliveryTime.DEFAULT.value());
            order.setDeliveryAssigned(IsDeliveryAssigned.NO.value());

            orderDao.save(order);
            cartDao.delete(cart);
        }

        System.out.println("response sent!!!");

        return new ResponseEntity("ORDER SUCCESS", HttpStatus.OK);

    }

    @GetMapping("myorder")
    public ResponseEntity getMyOrder(@RequestParam("userId") int userId) throws JsonProcessingException {

        System.out.println("request came for MY ORDER for USER ID : " + userId);

        List<Orders> userOrder = orderDao.findByUser_id(userId);

        OrderDataResponse orderResponse = new OrderDataResponse();

        List<MyOrderResponse> orderDatas = new ArrayList<>();

        for (Orders order : userOrder) {
            MyOrderResponse orderData = new MyOrderResponse();
            orderData.setOrderId(order.getOrderId());
            orderData.setProductDescription(order.getProduct().getDescription());
            orderData.setProductName(order.getProduct().getTitle());
            orderData.setProductImage(order.getProduct().getImageName());
            orderData.setQuantity(order.getQuantity());
            orderData.setOrderDate(order.getOrderDate());
            orderData.setProductId(order.getProduct().getId());
            orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
            orderData.setDeliveryStatus(order.getDeliveryStatus());
            orderData.setTotalPrice(
            String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toString())));
            if (order.getDeliveryPersonId() == 0) {
                orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
                orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
            }
```

```
            else {

                User deliveryPerson = null;

                Optional<User> optionalDeliveryPerson = this.userDao.findById(order.getDeliveryPersonId());

                deliveryPerson = optionalDeliveryPerson.get();

                orderData.setDeliveryPersonContact(deliveryPerson.getPhoneNo());
                orderData.setDeliveryPersonName(deliveryPerson.getFirstName());
            }
            orderDatas.add(orderData);
        }

        String json = objectMapper.writeValueAsString(orderDatas);

        System.out.println(json);

        return new ResponseEntity(orderDatas, HttpStatus.OK);

    }

    @GetMapping("admin/allorder")
    public ResponseEntity getAllOrder() throws JsonProcessingException {

        System.out.println("request came for FETCH ALL ORDERS");

        List<Orders> userOrder = orderDao.findAll();

        OrderDataResponse orderResponse = new OrderDataResponse();

        List<MyOrderResponse> orderDatas = new ArrayList<>();

        for (Orders order : userOrder) {
            MyOrderResponse orderData = new MyOrderResponse();
            orderData.setOrderId(order.getOrderId());
            orderData.setProductDescription(order.getProduct().getDescription());
            orderData.setProductName(order.getProduct().getTitle());
            orderData.setProductImage(order.getProduct().getImageName());
            orderData.setQuantity(order.getQuantity());
            orderData.setOrderDate(order.getOrderDate());
            orderData.setProductId(order.getProduct().getId());
            orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
            orderData.setDeliveryStatus(order.getDeliveryStatus());
            orderData.setTotalPrice(
                    String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toStr
            orderData.setUserId(order.getUser().getId());
            orderData.setUserName(order.getUser().getFirstName() + " " + order.getUser().getLastName());
            orderData.setUserPhone(order.getUser().getPhoneNo());
            orderData.setAddress(order.getUser().getAddress());
            if (order.getDeliveryPersonId() == 0) {
                orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
                orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
            }

            else {
                User deliveryPerson = null;

                Optional<User> optionalDeliveryPerson = this.userDao.findById(order.getDeliveryPersonId());

                deliveryPerson = optionalDeliveryPerson.get();

                orderData.setDeliveryPersonContact(deliveryPerson.getPhoneNo());
                orderData.setDeliveryPersonName(deliveryPerson.getFirstName());
            }
            orderDatas.add(orderData);

        }

        String json = objectMapper.writeValueAsString(orderDatas);

        System.out.println(json);

        System.out.println("response sent !!!");

        return new ResponseEntity(orderDatas, HttpStatus.OK);

    }

    @GetMapping("admin/showorder")
    public ResponseEntity getOrdersByOrderId(@RequestParam("orderId") String orderId) throws JsonProcessingExce

        System.out.println("request came for FETCH ORDERS BY ORDER ID : " + orderId);

        List<Orders> userOrder = orderDao.findByOrderId(orderId);

        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
232                 Optional<User> optionalDeliveryPerson = this.userDao.findById(order.getDeliveryPersonId());
233
234                 deliveryPerson = optionalDeliveryPerson.get();
235
236                 orderData.setDeliveryPersonContact(deliveryPerson.getPhoneNo());
237                 orderData.setDeliveryPersonName(deliveryPerson.getFirstName());
238             }
239             orderDatas.add(orderData);
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.findByOrderId(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.findByOrderId(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
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 FPOR 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             }
```

```
            orderDatas.add(orderData);

        }

        String orderJson = objectMapper.writeValueAsString(orderDatas);

        System.out.println(orderJson);

        System.out.println("response sent !!!");

        return new ResponseEntity(orderDatas, HttpStatus.OK);
    }

    @GetMapping("delivery/myorder")
    public ResponseEntity getMyDeliveryOrders(@RequestParam("deliveryPersonId") int deliveryPersonId)
            throws JsonProcessingException {

        System.out.println("request came for MY DELIVERY ORDER for USER ID : " + deliveryPersonId);

        User person = null;

        Optional<User> oD = this.userDao.findById(deliveryPersonId);

        if (oD.isPresent()) {
            person = oD.get();
        }

        List<Orders> userOrder = orderDao.findByDeliveryPersonId(deliveryPersonId);

        List<MyOrderResponse> orderDatas = new ArrayList<>();

        for (Orders order : userOrder) {
            MyOrderResponse orderData = new MyOrderResponse();
            orderData.setOrderId(order.getOrderId());
            orderData.setProductDescription(order.getProduct().getDescription());
            orderData.setProductName(order.getProduct().getTitle());
            orderData.setProductImage(order.getProduct().getImageName());
            orderData.setQuantity(order.getQuantity());
            orderData.setOrderDate(order.getOrderDate());
            orderData.setProductId(order.getProduct().getId());
            orderData.setDeliveryDate(order.getDeliveryDate() + " " + order.getDeliveryTime());
            orderData.setDeliveryStatus(order.getDeliveryStatus());
            orderData.setTotalPrice(
                    String.valueOf(order.getQuantity() * Double.parseDouble(order.getProduct().getPrice().toString())));
            orderData.setUserId(order.getUser().getId());
            orderData.setUserName(order.getUser().getFirstName() + " " + order.getUser().getLastName());
            orderData.setUserPhone(order.getUser().getPhoneNo());
            orderData.setAddress(order.getUser().getAddress());

            if (order.getDeliveryPersonId() == 0) {
                orderData.setDeliveryPersonContact(DeliveryStatus.PENDING.value());
                orderData.setDeliveryPersonName(DeliveryStatus.PENDING.value());
            }

            else {
                orderData.setDeliveryPersonContact(person.getPhoneNo());
                orderData.setDeliveryPersonName(person.getFirstName());
            }

            orderDatas.add(orderData);

        }

        String json = objectMapper.writeValueAsString(orderDatas);

        System.out.println(json);

        return new ResponseEntity(orderDatas, HttpStatus.OK);

    }

}
```

Figure:48

Product Controller

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

```java
package com.onlineshopping.controller;

import java.io.IOException;

@RestController
@RequestMapping("api/product")
@CrossOrigin(origins = "http://localhost:3000")
public class ProductController {

    @Autowired
    private ProductService productService;

    @Autowired
    private ProductDao productDao;

    @Autowired
    private CategoryDao categoryDao;

    @Autowired
    private StorageService storageService;

    @Autowired
    private UserDao userDao;

    @PostMapping("add")
    public ResponseEntity<?> addProduct(ProductAddRequest productDto) {
        System.out.println("recieved request for ADD PRODUCT");
        System.out.println(productDto);
        Product product=ProductAddRequest.toEntity(productDto);

        Optional<Category> optional = categoryDao.findById(productDto.getCategoryId());
        Category category = null;
        if(optional.isPresent()) {
            category = optional.get();
        }

        product.setCategory(category);
        productService.addProduct(product, productDto.getImage());

        System.out.println("response sent!!!");
        return ResponseEntity.ok(product);

    }

    @GetMapping("all")
    public ResponseEntity<?> getAllProducts() {

        System.out.println("request came for getting all products");

        List<Product> products = new ArrayList<Product>();

        products = productDao.findAll();

        System.out.println("response sent!!!");

        return ResponseEntity.ok(products);

    }

    @GetMapping("id")
    public ResponseEntity<?> getProductById(@RequestParam("productId") int productId) {

        System.out.println("request came for getting Product by Product Id");

        Product product = new Product();

        Optional<Product> optional = productDao.findById(productId);

        if(optional.isPresent()) {
            product = optional.get();
        }
        System.out.println("response sent!!!");

        return ResponseEntity.ok(product);

    }

    @GetMapping("category")
    public ResponseEntity<?> getProductsByCategories(@RequestParam("categoryId") int categoryId) {

        System.out.println("request came for getting all products by category");

        List<Product> products = new ArrayList<Product>();

        products = productDao.findByCategoryId(categoryId);

        System.out.println("response sent!!!");

```

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

```
    @PostMapping("login")
    public ResponseEntity<?> loginUser(@RequestBody UserLoginRequest loginRequest) {
        System.out.println("recieved request for LOGIN USER");
        System.out.println(loginRequest);

        User user = new User();
        user = userDao.findByEmailIdAndPasswordAndRole(loginRequest.getEmailId(), loginRequest.getPassword(), loginRequest.getRole());

        System.out.println("response sent!!!");
        return ResponseEntity.ok(user);
    }

    @GetMapping("deliveryperson/all")
    public ResponseEntity<?> getAllDeliveryPersons() {
        System.out.println("recieved request for getting ALL Delivery Persons!!!");

        List<User> deliveryPersons = this.userDao.findByRole("Delivery");

        System.out.println("response sent!!!");
        return ResponseEntity.ok(deliveryPersons);
    }

}
```

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:

```
package com.onlineshopping.dao;

import org.springframework.data.jpa.repository.JpaRepository;

@Repository
public interface AddressDao extends JpaRepository<Address, Integer> {

}
```

Figure:51

Cartdao:

```
package com.onlineshopping.dao;

import java.util.List;

@Repository
public interface CartDao extends JpaRepository<Cart, Integer> {

    List<Cart> findByUser_id(int userId);

}
```

Figure:52

Categorydao:

```
package com.onlineshopping.dao;

import org.springframework.data.jpa.repository.JpaRepository;

public interface CategoryDao extends JpaRepository<Category, Integer> {


}
```

Figure:53

Orderdao:

```java
package com.onlineshopping.dao;

import java.util.List;

@Repository
public interface OrderDao extends JpaRepository<Orders, Integer> {

    List<Orders> findByUser_id(int userId);
    List<Orders> findByOrderId(String orderId);
    List<Orders> findByUser_idAndProduct_id(int userId, int productId);
    List<Orders> findByUser(User user);
    List<Orders> findByDeliveryPersonId(int deliveryPersonId);

}
```

Figure:54

Productdao:

```java
package com.onlineshopping.dao;

import java.util.List;

@Repository
public interface ProductDao extends JpaRepository<Product, Integer> {

    List<Product> findByCategoryId(int category);

}
```

Figure:55

Userdao:

```java
package com.onlineshopping.dao;

import java.util.List;

@Repository
public interface UserDao extends JpaRepository<User, Integer> {

    User findByEmailIdAndPasswordAndRole(String emailId, String password, String role);
    List<User> findByRole(String role);

}
```

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:

```java
package com.onlineshopping.dto;

public class AddToCartRequest {

    private int productId;

    private int quantity;

    private int userId;

    public int getProductId() {
        return productId;
    }

    public void setProductId(int productId) {
        this.productId = productId;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    @Override
    public String toString() {
        return "AddToCartRequest [productId=" + productId + ", quantity=" + quantity + ", userId=" + userId + "]";
    }


}
```

Figure:57

AddUserRequest:

```java
package com.onlineshopping.dto;

public class AddUserRequest {

    private String firstName;

    private String lastName;

    private String emailId;

    private String password;

    private String phoneNo;

    private String street;

    private String city;

    private int pincode;

    private String role;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
```

```
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:

```java
package com.onlineshopping.dto;

public class CartDataResponse {

private int cartId;

    private int productId;

    private String productName;

    private String productDescription;

    private String productImage;

    private int quantity;

    public int getCartId() {
        return cartId;
    }

    public void setCartId(int cartId) {
        this.cartId = cartId;
    }

    public int getProductId() {
        return productId;
    }

    public void setProductId(int productId) {
        this.productId = productId;
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public String getProductDescription() {
        return productDescription;
    }

    public void setProductDescription(String productDescription) {
        this.productDescription = productDescription;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public String getProductImage() {
        return productImage;
    }

    public void setProductImage(String productImage) {
        this.productImage = productImage;
    }

    @Override
    public String toString() {
        return "CartDataResponse [cartId=" + cartId + ", productId=" + productId + ", productName=" + productName
                + ", productDescription=" + productDescription + ", productImage=" + productImage + ", quantity="
                + quantity + "]";
    }

}
```

Figure:59

Cart Response:

```java
package com.onlineshopping.dto;

import java.util.List;

public class CartResponse {

    private String totalCartPrice;

    private List<CartDataResponse> cartData;

    public String getTotalCartPrice() {
        return totalCartPrice;
    }

    public void setTotalCartPrice(String totalCartPrice) {
        this.totalCartPrice = totalCartPrice;
    }

    public List<CartDataResponse> getCartData() {
        return cartData;
    }

    public void setCartData(List<CartDataResponse> cartData) {
        this.cartData = cartData;
    }

    @Override
    public String toString() {
        return "CartResponse [totalCartPrice=" + totalCartPrice + ", cartData=" + cartData + "]";
    }

}
```

Figure:60

MyOrderResponse:

```java
package com.onlineshopping.dto;

import com.onlineshopping.model.Address;

public class MyOrderResponse {

    private String orderId;

    private int productId;

    private int userId;

    private String userName;  // first name + last name

    private Address address;

    private String userPhone;

    private String productName;

    private String productDescription;

    private String productImage;

    private int quantity;

    private String totalPrice;

    private String orderDate;

    private String deliveryDate;

    private String deliveryStatus;

    private String deliveryPersonName;

    private String deliveryPersonContact;

    public String getOrderId() {
        return orderId;
    }
```

```
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
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
```

```
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    public String getUserPhone() {
        return userPhone;
    }

    public void setUserPhone(String userPhone) {
        this.userPhone = userPhone;
    }

    public String getDeliveryPersonName() {
        return deliveryPersonName;
    }

    public void setDeliveryPersonName(String deliveryPersonName) {
        this.deliveryPersonName = deliveryPersonName;
    }
    public String getDeliveryPersonContact() {
        return deliveryPersonContact;
    }
    public void setDeliveryPersonContact(String deliveryPersonContact) {
        this.deliveryPersonContact = deliveryPersonContact;
    }

}
```

Figure:61

OrderDataResponse:

```
package com.onlineshopping.dto;

import java.util.List;

public class OrderDataResponse {

    private List<MyOrderResponse> orderResponse;

    private String totalCartPrice;

    public List<MyOrderResponse> getOrderResponse() {
        return orderResponse;
    }

    public void setOrderResponse(List<MyOrderResponse> orderResponse) {
        this.orderResponse = orderResponse;
    }

    public String getTotalCartPrice() {
        return totalCartPrice;
    }

    public void setTotalCartPrice(String totalCartPrice) {
        this.totalCartPrice = totalCartPrice;
    }

    @Override
    public String toString() {
        return "OrderDataResponse [orderResponse=" + orderResponse + ", totalCartPrice=" + totalCartPrice + "]";
    }

}
```

Figure:62

ProductAddRequest:

```java
package com.onlineshopping.dto;

import java.math.BigDecimal;

public class ProductAddRequest {

    private int id;
    private String title;
    private String description;
    private int quantity;
    private BigDecimal price;
    private int categoryId;
    private MultipartFile image;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public BigDecimal getPrice() {
        return price;
    }
    public void setPrice(BigDecimal price) {
        this.price = price;
    }
    public MultipartFile getImage() {
        return image;
    }
    public void setImage(MultipartFile image) {
        this.image = image;
    }
    public int getCategoryId() {
        return categoryId;
    }
    public void setCategoryId(int categoryId) {
        this.categoryId = categoryId;
    }
    public static Product toEntity(ProductAddRequest dto) {
        Product entity=new Product();
        BeanUtils.copyProperties(dto, entity, "image", "categoryId");
        return entity;
    }
    @Override
    public String toString() {
        return "ProductAddRequest [id=" + id + ", title=" + title + ", description=" + description + ", quantity="
                + quantity + ", price=" + price + ", categoryId=" + categoryId + "]";
    }

}
```

Figure:63

UserLoginRequest:

```java
package com.onlineshopping.dto;

public class UserLoginRequest {

    private String emailId;
    private String password;
    private String role;

    public String getEmailId() {
        return emailId;
    }
    public void setEmailId(String emailId) {
        this.emailId = emailId;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }
    public void setRole(String role) {
        this.role = role;
    }
    @Override
    public String toString() {
        return "UserLoginRequest [emailId=" + emailId + ", password=" + password + "]";
    }

}
```

Figure:64

UserResponse:

```java
package com.onlineshopping.dto;

public class UserResponse {

}
```

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.

```java
package com.onlineshopping.model;

import javax.persistence.Entity;

@Entity
public class Address {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String street;

    private String city;

    private int pincode;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public int getPincode() {
        return pincode;
    }

    public void setPincode(int pincode) {
        this.pincode = pincode;
    }

    @Override
    public String toString() {
        return "Address [id=" + id + ", street=" + street + ", city=" + city + ", pincode=" + pincode + "]";
    }

}
```

Figure:66

Cart Model:

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

```java
package com.onlineshopping.model;

import javax.persistence.Entity;

@Entity
public class Cart {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    @OneToOne
    @JoinColumn(name = "product_id")
    private Product product;

    @OneToOne
    @JoinColumn(name = "user_id")
    private User user;

    private int quantity;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Product getProduct() {
```

```
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;
 7
 8  @Entity
 9  public class Category {
10
11      @Id
12      @GeneratedValue(strategy=GenerationType.IDENTITY)
13      private int id;
14
15      private String title;
16
17      private String description;
18
19      public int getId() {
20          return id;
21      }
22
23      public void setId(int id) {
24          this.id = id;
25      }
26
27      public String getTitle() {
28          return title;
29      }
30
31      public void setTitle(String title) {
32          this.title = title;
33      }
34
35      public String getDescription() {
36          return description;
37      }
38
39      public void setDescription(String description) {
40          this.description = description;
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.

```java
package com.onlineshopping.model;

import javax.persistence.Entity;

@Entity
public class Orders {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String orderId;

    @OneToOne
    @JoinColumn(name = "product_id")
    private Product product;

    @OneToOne
    @JoinColumn(name = "user_id")
    private User user;

    private int quantity;

    private String orderDate;  // 13-01-2022 10:00 PM

    private String deliveryStatus;

    private String deliveryDate;

    private String deliveryTime; // evening, afternoon....

    private String deliveryAssigned;

    private int deliveryPersonId;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getOrderId() {
        return orderId;
    }

    public void setOrderId(String orderId) {
        this.orderId = orderId;
    }

    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public String getOrderDate() {
        return orderDate;
    }
```

```
    public void setOrderDate(String orderDate) {
        this.orderDate = orderDate;
    }
    public String getDeliveryStatus() {
        return deliveryStatus;
    }

    public void setDeliveryStatus(String deliveryStatus) {
        this.deliveryStatus = deliveryStatus;
    }
    public String getDeliveryDate() {
        return deliveryDate;
    }

    public void setDeliveryDate(String deliveryDate) {
        this.deliveryDate = deliveryDate;
    }

    public String getDeliveryTime() {
        return deliveryTime;
    }

    public void setDeliveryTime(String deliveryTime) {
        this.deliveryTime = deliveryTime;
    }

    public int getDeliveryPersonId() {
        return deliveryPersonId;
    }
    public void setDeliveryPersonId(int deliveryPersonId) {
        this.deliveryPersonId = deliveryPersonId;
    }
    public String getDeliveryAssigned() {
        return deliveryAssigned;
    }
    public void setDeliveryAssigned(String deliveryAssigned) {
        this.deliveryAssigned = deliveryAssigned;
    }

}
```

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.

```java
package com.onlineshopping.model;

import java.math.BigDecimal;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String title;
    private String description;
    private int quantity;
    private BigDecimal price;
    private String imageName;

    @ManyToOne
    @JoinColumn(name ="categoryId")
    private Category category;


    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public int getQuantity() {
        return quantity;
    }
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public BigDecimal getPrice() {
        return price;
    }
    public void setPrice(BigDecimal price) {
        this.price = price;
    }
    public String getImageName() {
        return imageName;
    }
    public void setImageName(String imageName) {
        this.imageName = imageName;
    }
    public Category getCategory() {
        return category;
    }
    public void setCategory(Category category) {
        this.category = category;
    }

}
```

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.

```java
package com.onlineshopping.model;

import javax.persistence.Entity;

@Entity
public class User {

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;

	private String firstName;

	private String lastName;

	private String emailId;

	@JsonIgnore
	private String password;

	private String phoneNo;

	private String role;

	@OneToOne
	@JoinColumn(name = "address_id")
	private Address address;

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}


	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public String getEmailId() {
		return emailId;
	}

	public void setEmailId(String emailId) {
		this.emailId = emailId;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public String getPhoneNo() {
		return phoneNo;
	}

	public void setPhoneNo(String phoneNo) {
		this.phoneNo = phoneNo;
	}

```

```
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;
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
```

```
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
```

```
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:

```java
package com.onlineshopping.service;

import org.springframework.beans.factory.annotation.Autowired;


@Service
public class ProductServiceImpl implements ProductService {

	@Autowired
	private ProductDao productDao;

	@Autowired
	private StorageService storageService;

	@Override
	public void addProduct(Product product, MultipartFile productImmage) {

		String productImageName = storageService.store(productImmage);

		product.setImageName(productImageName);

		this.productDao.save(product);
	}

}
```

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.

```java
package com.onlineshopping.utility;

public class HeaderCode {

}
```

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.

```java
package com.onlineshopping.utility;

public class Helper {

	public static String getAlphaNumericOrderId()
	{

		String AlphaNumericString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
				+ "0123456789"
				+ "abcdefghijklmnopqrstuvxyz";

		StringBuilder sb = new StringBuilder(10);

		for (int i = 0; i < 10; i++) {

			int index
				= (int)(AlphaNumericString.length()
					* Math.random());

			sb.append(AlphaNumericString
					.charAt(index));
		}

		return sb.toString().toUpperCase();
	}

}
```

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.

```java
package com.onlineshopping.utility;

import java.util.List;

public interface StorageService {

	List<String> loadAll();
	String store(MultipartFile file);
	Resource load(String fileName);
	void delete(String fileName);

}

```

Figure:76

Storage Serviceimpl:

```java
package com.onlineshopping.utility;
import java.io.File;
import java.io.FileOutputStream;
import java.util.Arrays;
import java.util.List;
import java.util.UUID;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Component;
import org.springframework.util.FileCopyUtils;
import org.springframework.web.multipart.MultipartFile;
@Component
public class StorageServiceImpl implements StorageService {
    @Value("${disk.upload.basepath}")
    private String BASEPATH;
    @Override
    public List<String> loadAll() {
        File dirPath = new File(BASEPATH);
        return Arrays.asList(dirPath.list());}
    @Override
    public String store(MultipartFile file) {
        System.out.println(file.getOriginalFilename());
        String ext=file.getOriginalFilename().substring(file.getOriginalFilename().lastIndexOf("."));
        System.out.println(ext);
        String fileName = UUID.randomUUID().toString().replaceAll("-", "")+ext;
        File filePath = new File(BASEPATH, fileName);
        try(FileOutputStream out = new FileOutputStream(filePath)) {
            FileCopyUtils.copy(file.getInputStream(), out);
            return fileName;
        } catch (Exception e) {
            e.printStackTrace();}
        return null;}
    @Override
    public Resource load(String fileName) {
        File filePath = new File(BASEPATH, fileName);
        if(filePath.exists())
            return new FileSystemResource(filePath);
        return null;}
    @Override
    public void delete(String fileName) {
        File filePath = new File(BASEPATH, fileName);
        if(filePath.exists())
            filePath.delete();}}

```

Figure:77

5.2.8 Resources:

```
# MySQL Properties
spring.datasource.url=jdbc:mysql://localhost:3306/online_shopping?createDatabaseIfNotExist=true&useUnicode=true
spring.datasource.username=root
spring.datasource.password=9666516041@vV
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
disk.upload.basepath=C:/Users/91950/Downloads/Ecommerce_Project_Fullstack/images

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

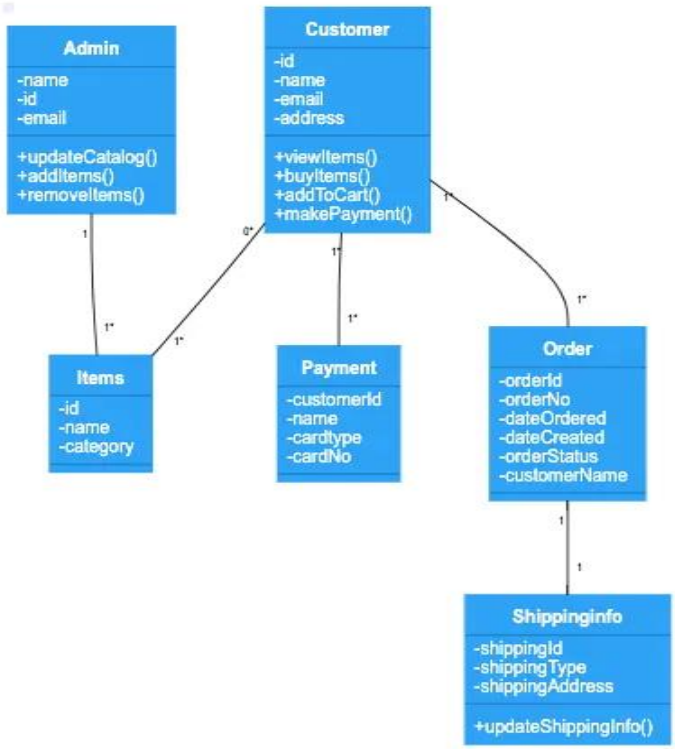


Figure:79 Class Diagram

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 27 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.

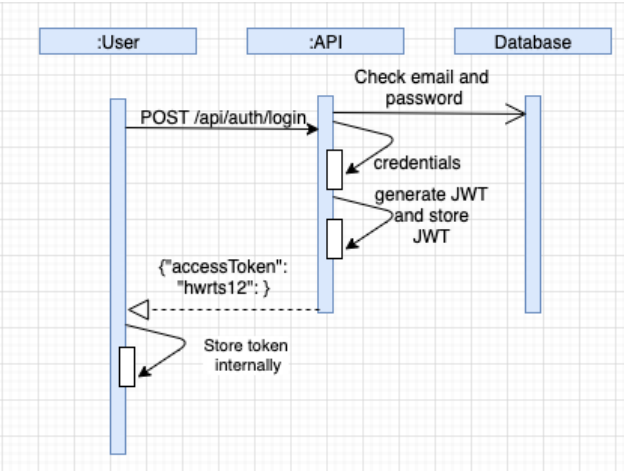


Figure:80 Login

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**.**

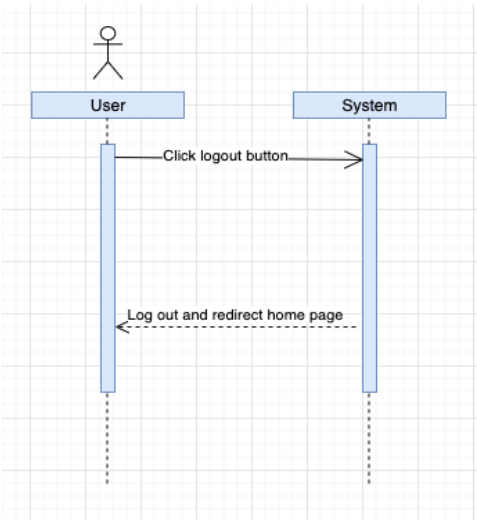


Figure:81 Logout

Add Product

The sequence diagram in Figure82 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 in-formation. 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.

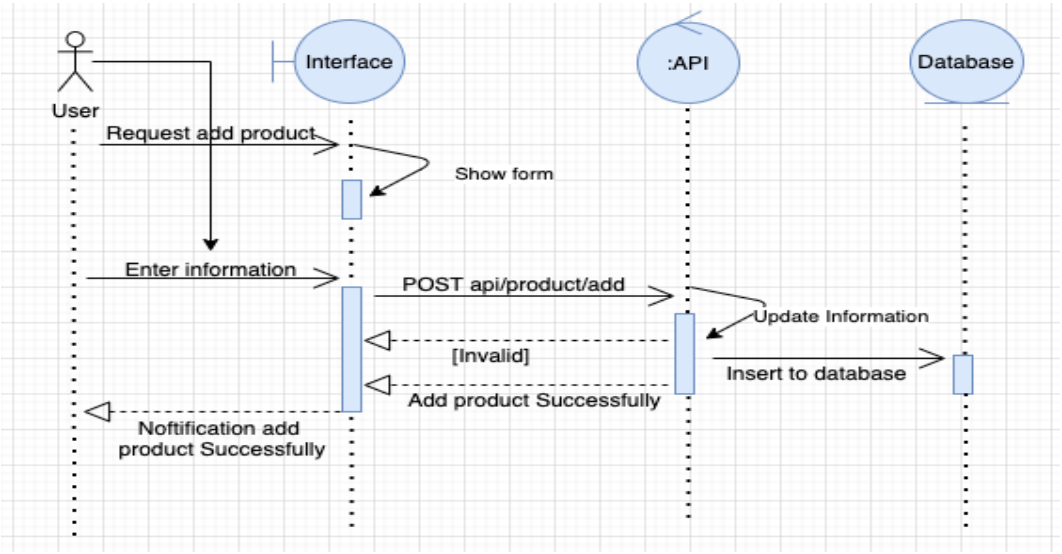


Figure:82 Add Product

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 is information correct.

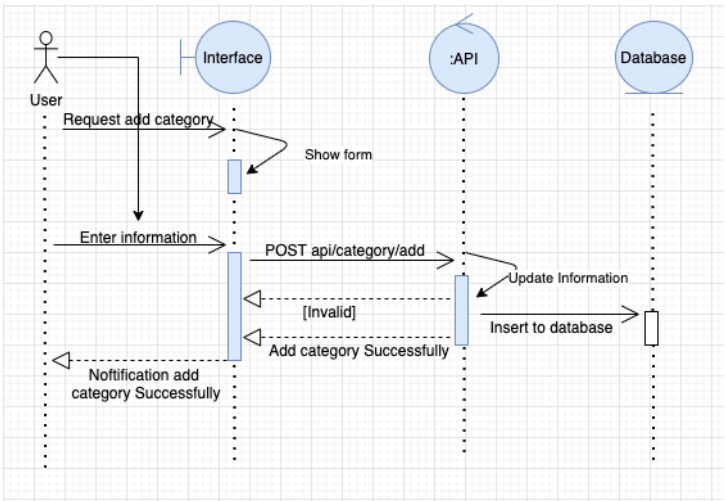


Figure:83 Categories

Add User

The sequence diagram in Figure84 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.

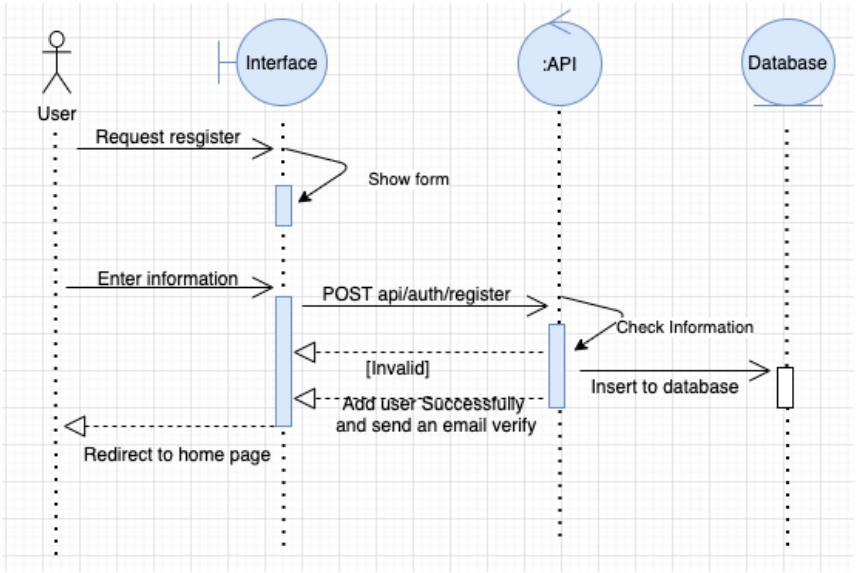


Figure:84 Add User

Assign Role

The sequence diagram in Figure 85 shows the steps in assigning roles. The Admin logins 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.

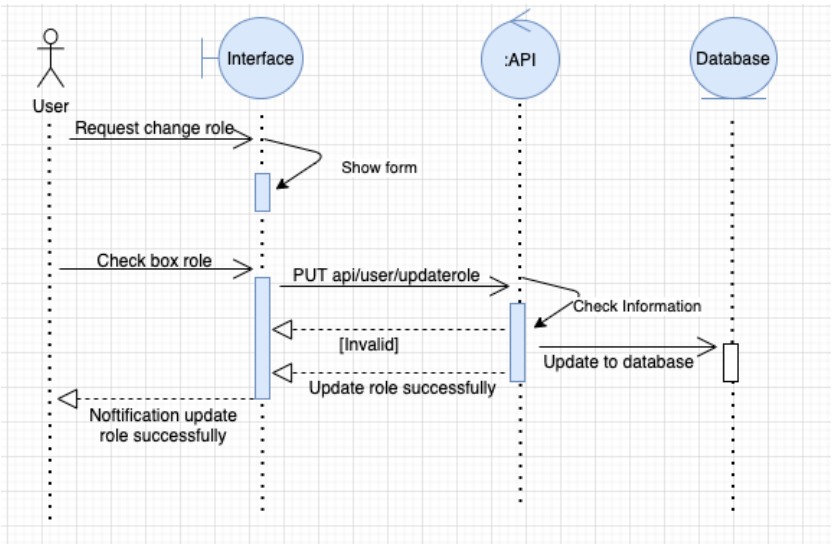


Figure:85 Assign Role

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.

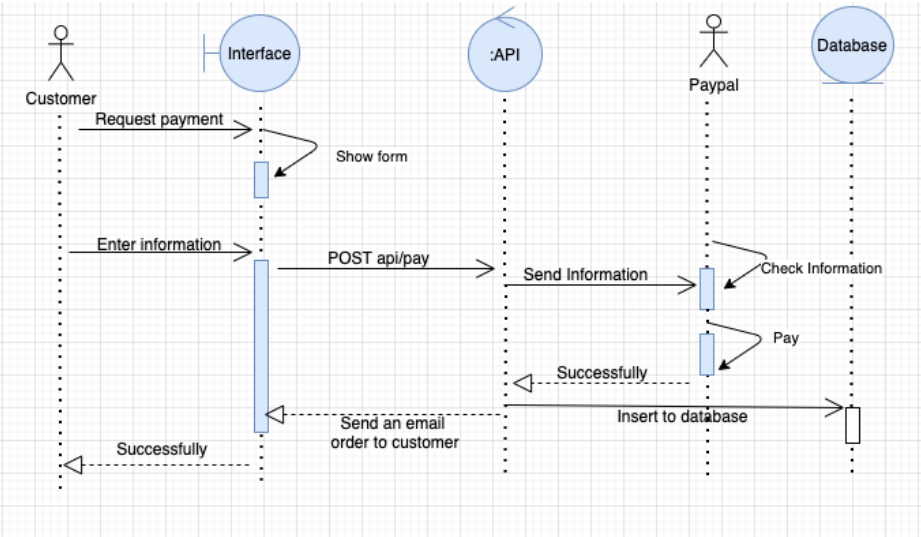


Figure:86 Order

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	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”.

	2. The user enters the necessary information (including some required information) and presses "Save". 3. The system checks the information, if the information is valid, the next step will be taken. (Other event stream: Invalid information). 4. The system saves the data and reports it successfully. (Other event: Cannot add product to database).
Other Event	1. Invalid information: The system displays a red message on the spot of an error and asks to re-enter information. 2. Cannot save to database: Error while adding => Ask user to re-enter information, if still error contact the development team.
Special Required	Not available
System state before use case execution	Actor: Admin, Staff 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 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	1. Go to the product category manager, click the button "Add categories."' 2. The user enters the necessary information (including some required information) and presses "Save". 3. The system checks the information, if the information is valid, the next step will be taken. (Other event: Invalid information). 4. The system saves the data and notify it of success. (Other event stream: Cannot update to the database)
Other Event	1. Invalid information: The system displays a message asking for information re-input. 2. Cannot update the database: Error while updating => Request user to re-enter information; if still error, contact development team.
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.

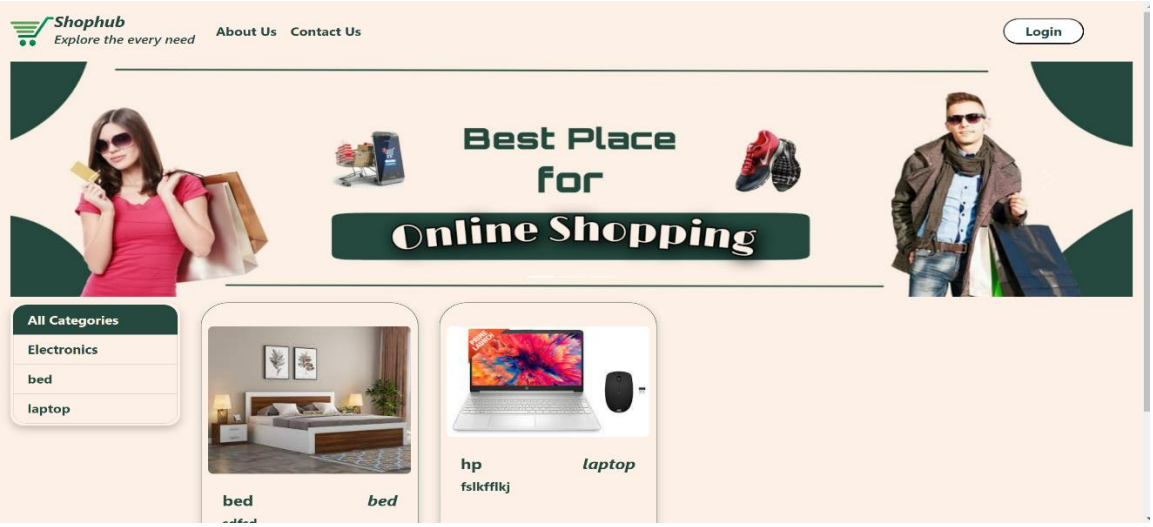


Figure:87 HomePage

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.

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.

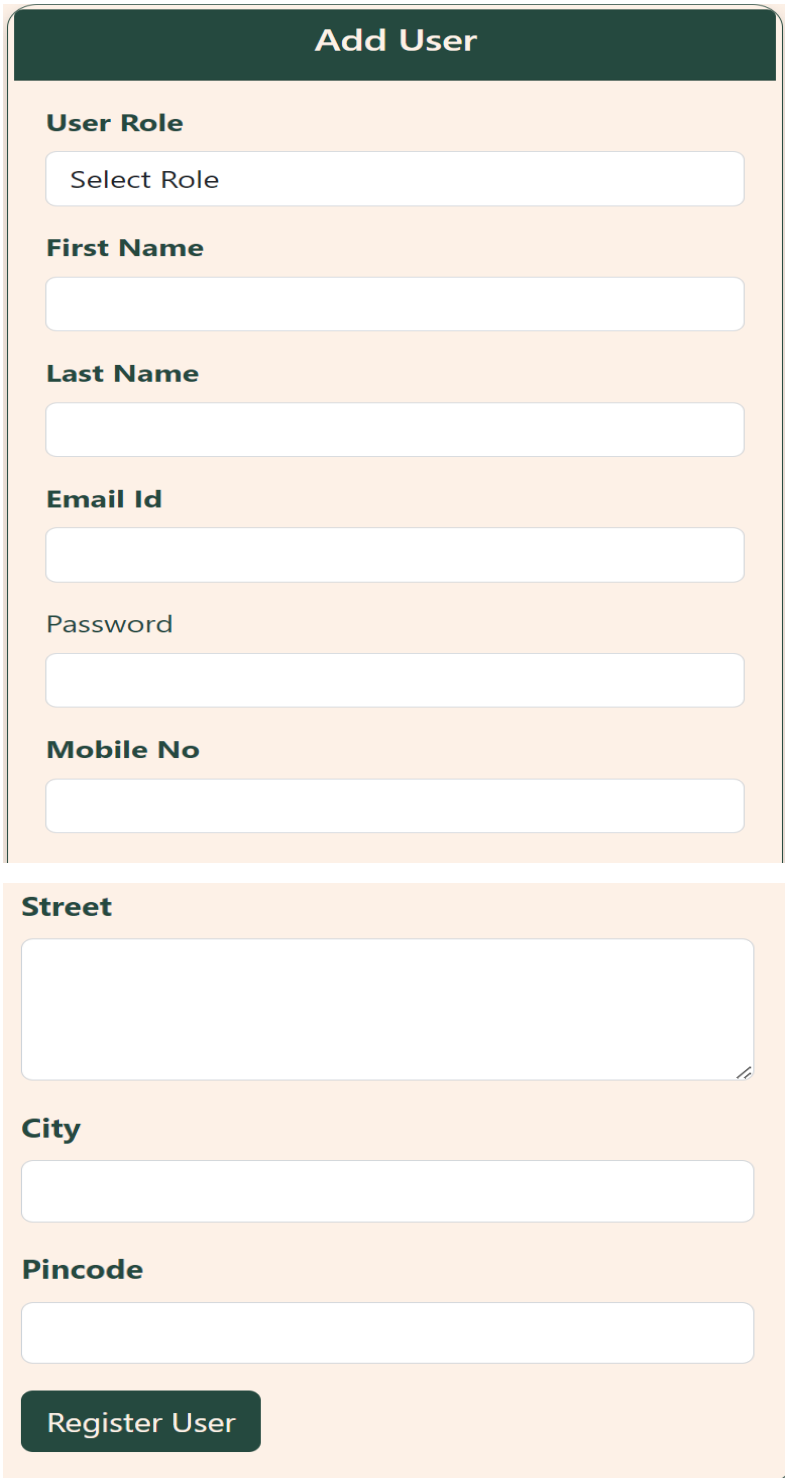


Figure:89 Register

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.

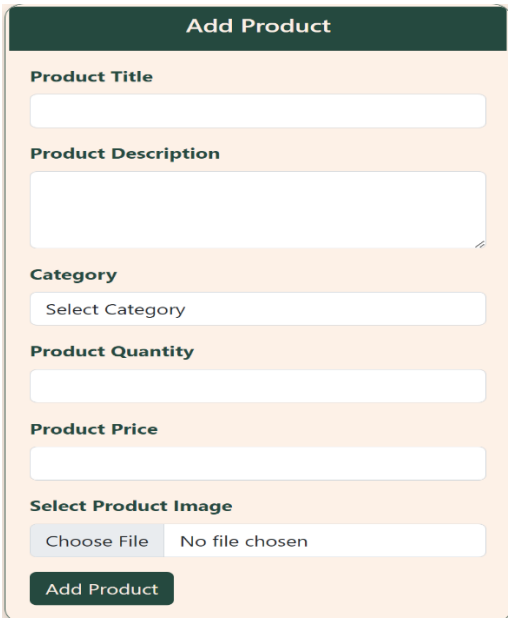


Figure:91 Add Product

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.

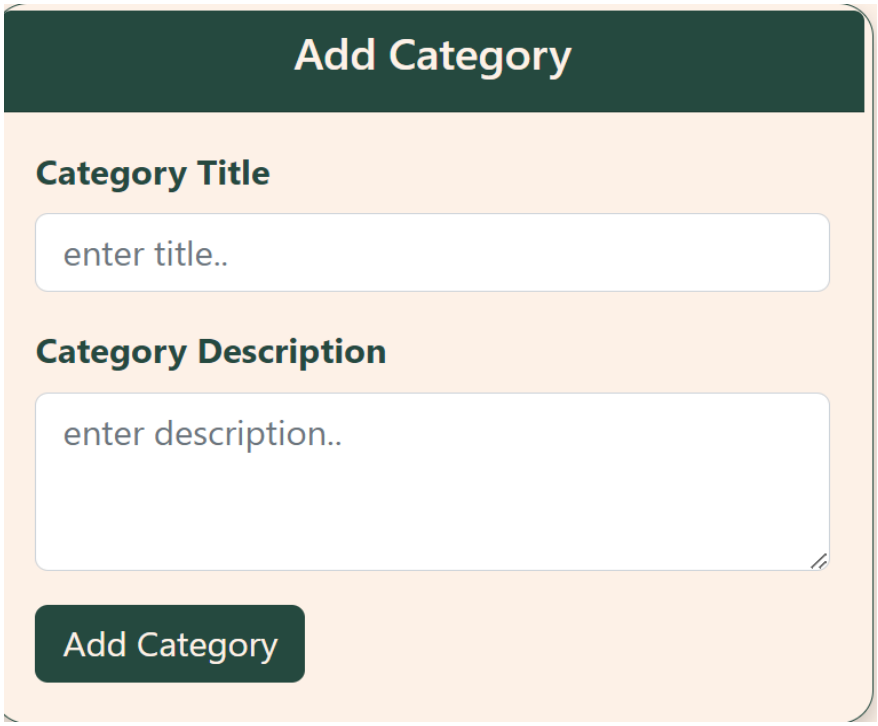


Figure:92 Add Category

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.

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.

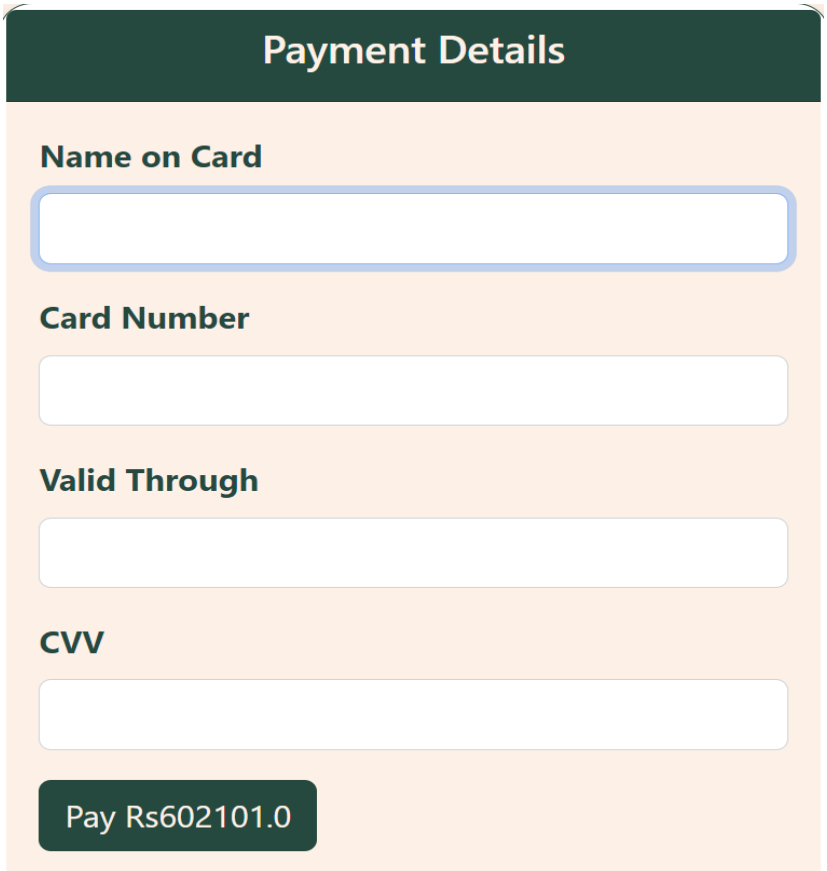


Figure:94 Checkout Page

My Orders

Which we can see All Orders in the My Orders pages

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.