WEB TECHNOLOGIES LABORATORY

MINI PROJECT REPORT

ONLINE MARKET PLACE

Submitted by

VIJAYALAKSHMI T (953622104118)

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAMCO INSTITUTE OF TECHNOLOGY,

RAJAPALAYAM

DECEMBER 2024

BONAFIDE CERTIFICATE

Certified that this miniproject report **"ONLINE MARKET PLACE"** is the bonafide work of **"VIJAYALAKSHMI T (953622104118)"** who carried out the miniproject work under my supervision.

SIGNATURE
Dr.K.Vijayalakshmi M.E., Ph.D.

HEAD OF THE DEPARTMENT
Department of Computer Science and Engineering
Ramco Institute of Technology,
North Venganallur Village, Rajapalayam – 626117.

SIGNATURE
Mrs.B.Vijayalakshmi M.E., Ph.D.

FACULTY IN-CHARGE
Department of Computer Science and Engineering
Ramco Institute of Technology,
North Venganallur Village,
Rajapalayam– 626117.

Submitted for the Practical Examination held at Ramco Institute of Technology, Rajapalayam on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

TABLE OF CONTENT

1.ABSTRACT

The **Online Marketplace** is a dynamic e-commerce platform developed using the **MERN** stack (MongoDB, Express.js, React.js, and Node.js) that enables users and sellers to interact seamlessly for buying and selling products. The application simplifies online shopping by allowing sellers to list their products and buyers to explore items across various categories, such as electronics, cosmetics, and more, based on personalized preferences.

The frontend, built with React.js, ensures a responsive and engaging user experience across various devices. It communicates with the backend, powered by Node.js and Express.js, which handles essential functions such as user authentication, seller management, product addition, and category-based product storage.The application leverages MongoDB as its database to store seller information, user accounts, and product details. MongoDB's flexible document-based model allows for efficient management and scaling of product attributes, including names, prices, categories, images, and discounts, ensuring the system is adaptable to future requirements.

Session-based user authentication and **JSON Web Tokens (JWT)** ensure that users and sellers can securely create accounts, log in, and manage their activities. Sellers can add products dynamically while maintaining a personalized dashboard, and buyers can browse categorized products and add items to their cart. The system is designed for scalability and modularity, enabling future enhancements such as payment gateways, advanced search functionality, and AI-powered recommendations. These features aim to enhance user experience, providing personalized shopping suggestions and efficient product discovery.

2.INTRODUCTION

The **Online Marketplace** is designed to address the challenges faced by users when searching for and purchasing products online. While existing e-commerce platforms provide a broad selection of items, they often lack user-centric features such as seamless seller integration, personalized recommendations, and efficient product categorization. This project offers an innovative solution by providing a streamlined platform for both sellers and buyers, ensuring a robust and user-friendly shopping experience.Built using the **MERN** stack (MongoDB, Express.js, React.js, and Node.js), the application offers a responsive and scalable solution for managing online shopping activities. The React.js-based frontend ensures an interactive user interface, enabling buyers to search for products efficiently and sellers to add and manage their inventory with ease. Buyers can browse products by categories such as electronics, cosmetics, and more, while sellers can add product details dynamically.

The backend, powered by Node.js and Express.js, facilitates secure user and seller management, including authentication, product handling, and database operations. MongoDB serves as the database, efficiently storing user, seller, and product data in a flexible and scalable document-based model. It supports features like product images, prices, categories, and discounts.

Key features include user authentication using **JSON Web Tokens (JWT)**, session management for seamless transitions, and seller dashboards for managing product inventory. The modular and scalable architecture of the system ensures easy maintenance and the potential for future enhancements, such as payment gateways, machine learning-powered recommendations, and customer reviews.Overall, this Online Marketplace aims to bridge the gap between buyers and sellers, offering a platform that is not only efficient and scalable but also adaptable to evolving e-commerce trends.

3.PROBLEM STATEMENT

In the era of digital shopping, the overwhelming number of options and poorly designed platforms make the shopping experience challenging for both buyers and sellers. Existing e-commerce systems often lack user-centric features, leading to inefficiencies and dissatisfaction.

Challenges in Current Platforms:

1. **Information Overload:** Users face long lists of items without advanced filtering options, making it difficult to find desired products.
2. **Lack of Seller-Friendly Features:** Sellers often struggle with unintuitive interfaces and limited tools for managing their products.
3. **Limited Personalization:** Platforms fail to offer personalized recommendations based on user preferences.
4. **Fragmented User Experience:** Features like seamless seller integration and session persistence are often absent, hindering the overall experience.

This project seeks to overcome these challenges by creating an efficient, personalized, and user-friendly online marketplace. With features like dynamic product categorization, robust seller tools, and secure authentication, the platform ensures a seamless shopping experience for all users.

4.OBJECTIVES

The **Online Marketplace** project focuses on developing a scalable, user-friendly platform that bridges the gap between buyers and sellers. The following objectives define the project scope:

Core Objectives:

1. **User-Friendly Interface:**
    - Design an intuitive and responsive web interface using React.js.
    - Ensure smooth navigation for buyers and sellers across devices.
2. **Seller Management:**
    - Allow sellers to register, log in, and manage their product inventory.
    - Provide secure tools for sellers to upload product images, set prices, and categorize items.
3. **Personalized Product Discovery:**
    - Enable buyers to browse and filter products by category, price, and offers.
    - Display dynamic and real-time product results.
4. **Secure User and Seller Management:**
    - Implement JWT-based authentication for secure logins.
    - Ensure session persistence for seamless user and seller experiences.
5. **Scalable System Design:**
    - Develop a modular architecture using the MERN stack.
    - Follow coding best practices to allow easy future enhancements.
6. **Category-Based Product Management:**
    - Store products dynamically in collections (e.g., Electronics, Beauty) based on their categories.

5.SYSTEM DESIGN

The system design encompasses the architecture diagram, use case diagram, entity-relationship diagram (ERD), database design and module descriptions. The application follows a modular structure that integrates frontend and backend systems seamlessly using the MERN stack.

5.1. Architecture Diagram - High-Level System Architecture & Data Flow

High-Level System Architecture:

- **Frontend (React.js)**: User interface where users interact with the application. React components handle user actions and display restaurant data fetched from the backend.
- **Backend (Node.js & Express.js)**: RESTful API that handles business logic, user authentication, and communication with the database. The backend processes requests such as fetching restaurant data, saving favourites, and managing user accounts.
- **Database (MongoDB)**: NoSQL database used to store restaurant details, user information, and saved restaurants. MongoDB's flexible document structure supports various restaurant attributes such as location, cuisine, ratings, etc.
- **JWT Authentication**: Secure token-based user authentication is handled through JSON Web Tokens (JWT), allowing users to register, log in, and securely access their saved restaurant data.

Data Flow:

1. **User Request**: Buyers and sellers interact with the React.js frontend.
2. **API Request**: Frontend sends requests to the backend API to fetch or store data.
3. **Backend Processing**: Backend validates requests, interacts with the MongoDB database, and returns responses.
4. **Database Operations**: MongoDB stores and retrieves data such as products, user details, and seller inventories.
5. **Response**: Backend returns processed data to the frontend for rendering.

5.2. Use Case Diagram

The use case diagram illustrates the major functions and interactions between users and the system.

Actor:

- **Buyer:** Browses products, adds items to the cart, and manages purchases.
- **Seller:** Adds, updates, and manages product listings.

Use Cases:

1. **Buyer Use Cases:**
    - Search products.
    - View product details.
    - Add items to the cart.
2. **Seller Use Cases:**
    - Register and log in.
    - Add or update product listings.
    - View dashboard for product management.

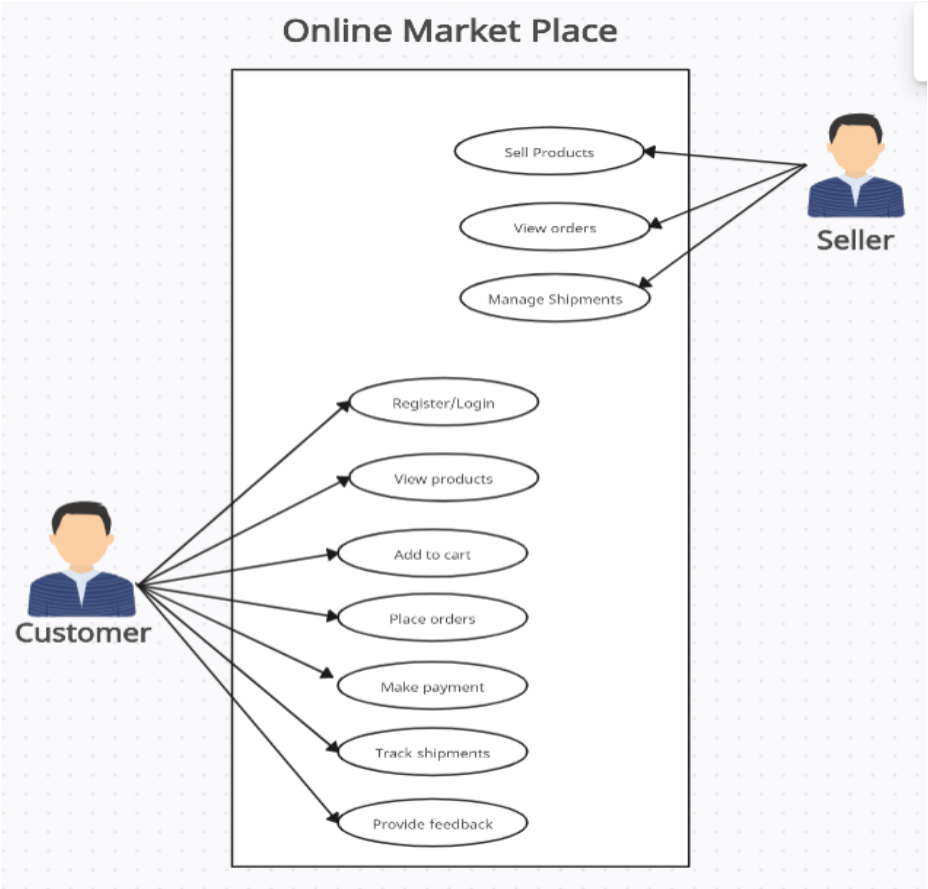

Fig:5.2.1 Use Case diagram

5.3. ER Diagram (Entity-Relationship Diagram)

The ER diagram represents the relationships between the entities in the database.

Entities:

1. **Users**:
    - **name :** User name for login management
    - **email :** User email
    - **mobileNumber :** User mobile number
    - **password :** User details for login management
2. **sellers:**
    - **sellerId :** Unique identifier.
    - **username :** Seller name for seller authentication.
    - **password :** Details for seller authentication.
3. **sellproduct :**
    - **name :** The name of the product.
    - **price :** The product price.
    - **category :** Split the product by category-wise.
    - **imageUrl :** The image link for the product.
    - **discount :** The discount offered by the seller while selling the product.
4. **products:**
    - **name :** The name of the product.
    - **imageUrl :** The image link for the product
    - **price :** Displays the price of the product.
    - **category :** Separates product by using name of the category.
5. **cosmetics:**
    - **Name** The name of the product.
    - **imageUrl** The image link for the product
    - **price :** Displays the price of the product.
    - **category :** Separates product as category-wise.

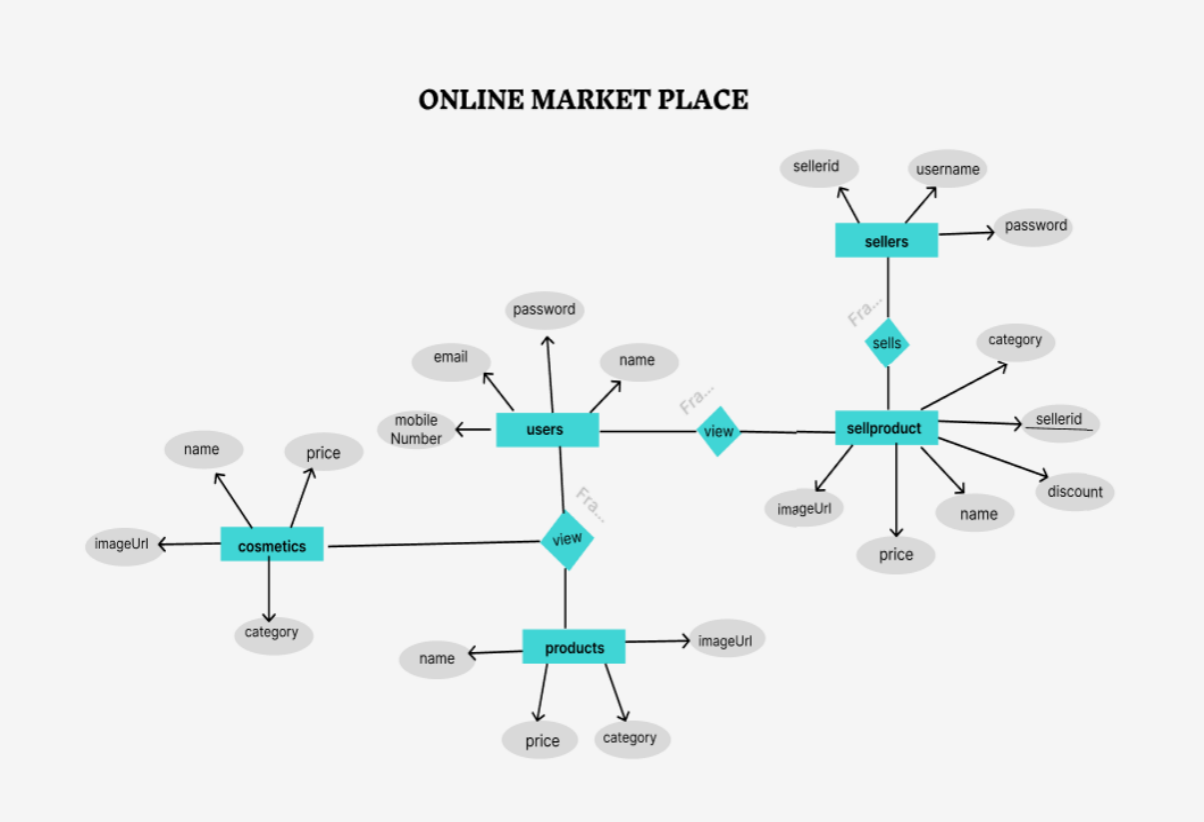

Fig:5.3.1 ER Diagram

Explanation:

1. **users** :
    - Each user has personal details like **name**, **email**, and **password**.
    - They can sign in using the correct credentials such as username and the correct password
2. **sellers** :
    - Each seller was uniquely identified by **sellerId(_id)**.
    - Each seller has details like **name** and **password**.
6. **sellproduct :**
    - Each product was added with their respective seller like **sellerId**.
    - The sellproduct collection contains following details about that product.
        - name
        - price
        - category,
        - imageUrl
        - discount

7. **products:**
    - The **products** collection was used to store the product that was categorized under the Electronics category.
        - **name**
        - **imageUrl**
        - **price**
        - **category**
8. **cosmetics:**
    - The **cosmetics** collection was used to store the product that was categorized under the Beauty category.
        - **name**
        - **imageUrl**
        - **price :** displays the price of the product.
        - **category :** separates product as category-wise.
9. **Relationship**:
    - Sellers manage multiple products (One-to-Many).
    - Users can interact with multiple products (Many-to-Many).

5.4. DB Design

MongoDB Schema Design:

The database design will consist of five collections. Each collection will contain documents (records) that store various attributes related to users and restaurants. These collections are designed to efficiently store data, allow for easy retrieval, and support scalability.

- **users**
- **sellers**
- **products**
- **sellproducts**
- **cosmetics**.

5.5. Modules Description

The Online Market Place is divided into several modules, each responsible for a specific functionality. The system is built using the MERN stack (MongoDB, Express.js, React.js, Node.js), ensuring a seamless flow between the frontend, backend, and database.

1. Authentication Module

Functionality: Handles user registration, login, and authentication. Ensures secure access to protected features through JWT-based authentication.

Operations:

- **registerUser:**
  - Takes user details (name, email, mobileNumber, and password) for user registration.
  - Stores the **users** collection in the database.
- **loginUser:**
  - Validates the user's name and password.
  - Generates a JWT token upon successful login.
  - Sends the token to the frontend for authenticated API requests.
- **registerSeller:**
  - Takes seller details (sellerId, username, and password) for seller registration.
  - Stores the seller in the **sellers** collection.
- **loginSeller:**
  - Validates the seller's username and password from the **Sellers** collection.
  - Generates a JWT token upon successful login.
  - Sends the token to the frontend for authenticated API requests.
- **validateToken:**
  - Middleware function that verifies JWT tokens for protected routes.
  - Ensures that the request is coming from an authenticated user or seller.

Technologies: Node.js, Express.js, MongoDB, JWT.

2. Product Management Module

Functionality: Manages product-related operations such as retrieving and filtering product data from the database.

Operations:

- **getAllProducts:**
  - **Retrieves a list of all available products from the Products collection.**
- **filterProducts:**
  - **Filters products based on category, price, or other relevant parameters.**
  - **Returns a list of products filtered based on the selected criteria.**
- **getProductById:**
  - **Fetches specific product details from the Products collection by its unique productId.**
- **addProduct:**
  - **Allows sellers to add products to the SellProduct collection, including details such as name, price, category, imageUrl, and discount.**
- **updateProduct:**
  - **Allows sellers to update their product details in the SellProduct collection.**

Technologies: Node.js, Express.js, MongoDB queries, Mongoose ORM

3. Saved Products Module(User)

Functionality: Manages the user's profile and their saved products (favorites).

Operation:

- **saveProduct**:
  - Allows users to save products to their list of favorites (`savedProducts` array in the **Users** collection).
- **getSavedProducts**:
  - Retrieves a list of products saved by the user.

Technologies: Node.js, Express.js, MongoDB, Mongoose

4. Frontend UI Module

Functionality: Provides an intuitive interface for users and sellers to explore products, manage their profiles, and perform search operations..

Operations:

- **Product List View:**
  - Displays a list of products based on filters like category, price, etc.
- **Product Details View:**

    - Displays detailed product information, including name, price, category, imageUrl, and discount.
- **Save and Manage Favorites:**
    - Allows users to save products to their favorites list, retrieved from the Users collection.
- **User Authentication:**
    - Includes login and registration forms for both users and sellers.

Technologies: React.js, React Router, Axios for API calls, CSS Modules for styling

5. API Routes Module

Functionality: Defines RESTful API routes for the frontend to communicate with the backend.

Operations:

- **POST /register:** Registers a new user.
- **POST /login:** Authenticates a user and returns a JWT token.
- **POST /seller/register**: Registers a new seller.
- **POST /seller/login:** Authenticates a seller and returns a JWT token.
- **GET /products**: Retrieves products based on filters (e.g., category, price).
- **GET /sell-products**: Retrieves products listed by a specific seller.
- **POST /user/savedProducts:** Saves a product to the user's list of favorites.

Technologies: Node.js, Express.js, MongoDB, Mongoose

6. Database Interaction Module

Functionality: Manages all database operations, including CRUD functions for users, sellers, and products.

Operations:

- **createUser:** Inserts a new user document into the Users collection.
- **findUserByEmail:** Searches the database for a user by their email.
- **createSeller:** Inserts a new seller document into the Sellers collection.
- **findSellerByUsername:** Searches the database for a seller by their username.
- **addProduct:** Inserts a new product document into the SellProduct collection.
- **getProductById:** Retrieves a product document by its unique productId.
- **saveUserProduct:** Adds a product to the user's savedProducts array in the Users collection.

Technologies: MongoDB, Mongoose ORM

6.IMPLEMENTATION

1.Frontend Implementation (React.js)

The frontend of the **Online Marketplace** is developed using **React.js**, which is a popular JavaScript library for building dynamic user interfaces. React allows for seamless interaction with the backend through HTTP requests, dynamic rendering of data, and management of state.

Key Features:

- **User Authentication Pages**:
  - **Registration Page**:
    - Allows users to create new accounts with fields for name, email, mobileNumber, and password.
    - Implements validation for password strength and email format.
    - Sends a POST request to /register with the user's email, password, and mobileNumber.
    - Redirects to the login page upon successful registration.
  - **Login Page**:
    - Allows users to log in with their email and password.
    - Sends a POST request to /login to authenticate the user.
    - Receives a **JWT token** from the backend for session management and stores it in **localStorage** or **sessionStorage**.
- **Product Listing and Details View**:
  - **Product Search**:
    - Users can search for products based on filters like category, price, and rating.
    - Sends a GET request with filter parameters to the backend.
    - Dynamically renders a list of products returned by the backend.
  - **Product Details**:
    - Displays detailed information about a specific product when clicked on a product card.
- **Saved Management**:
  - Allows users to mark products as saved.
  - Sends a POST request to /user/savedProducts to save a product to the user's savedProducts.

- **Navigation and Routing (React Router)**:
    - **React Router** is used to handle page navigation within the app.
    - Enables Single-Page Application (SPA) behavior for smooth transitions between pages without reloading the entire page.
- **State Management (React Hooks)**:
    - Uses useState and useEffect hooks to manage states and side-effects.
    - useState is used for managing the products list, user data, and other component-specific states.
    - useEffect is used to fetch data from the backend on initial load and after performing searches.
- **Error Handling and Validation**:
    - Implements real-time form validation for the registration and login forms.
    - Displays user-friendly error messages in case of failed requests (e.g., invalid credentials).

Technologies and Libraries:

- **React.js** for building UI components.
- **React Router** for page navigation.
- **Axios** for making HTTP requests.
- **CSS Modules** for scoped styling.
- **LocalStorage/SessionStorage** for storing JWT tokens.

2. Backend Implementation (Node.js and Express.js)

The backend of the **Online Marketplace** is built using **Node.js** and **Express.js**. It handles user authentication, product management, and API routes to interact with the MongoDB database.

Key Features:

- **User Authentication and Session Management**:
    - **Registration**:
        - Accepts user data (name, email, mobileNumber, and password), hashes the password using **bcrypt**, and stores the user in the **Users** collection.
    - **Login**:
        - Verifies the user's email and password, and if valid, generates a **JWT token** using **jsonwebtoken** for session management.

- **Seller Authentication**:
    - **registerSeller**:
        - Accepts username and password, hashes the password using **bcrypt**, and stores the seller in the **Sellers** collection.
    - **loginSeller**:
        - Verifies seller's username and password, and generates a **JWT token** upon successful login.
- **Product Management**:
    - **GET /products**: Retrieves products from the **Products** collection, with optional filters (e.g., by category, price).
    - **GET /sell-products**: Retrieves products added by a specific seller.
    - **POST /sell-products**: Allows sellers to add new products to the **SellProduct** collection.
- **Saved Products**:
    - **GET /user/saved**: Retrieves a list of saved products for the logged-in user.
    - **POST /user/savedProducts**: Allows users to save a product to their favorites.
- **API Security and Middleware**:
    - **JWT Middleware**: Verifies JWT tokens for incoming requests to ensure the user or seller is authenticated.
    - **Error Handling**: Manages invalid requests or server issues (404, 500 errors).

Technologies and Libraries:

- **Node.js** for server-side logic.
- **Express.js** for routing and middleware.
- **jsonwebtoken** for JWT-based authentication.
- **MongoDB** for database management.
- **Mongoose** for ODM (Object Data Modeling).

3. Database Implementation (MongoDB and Mongoose)

The database stores user, seller, product, and favorites data. **MongoDB** is chosen due to its flexibility with unstructured data and scalability. **Mongoose** is used for modeling the data and interacting with MongoDB.

Key Features:

- **User Data Storage**:
  - **Users** collection stores name, email, mobileNumber, password, and savedProducts (an array of product IDs referencing the **SellProduct** collection).
- **Seller Data Storage**:
  - **Sellers** collection stores sellerId, username, and password.
- **Product Data Storage**:
  - **SellProduct** collection stores product details like name, price, category, imageUrl, and discount.
  - **Products** collection stores products displayed in the marketplace, including name, price, category, and imageUrl.
- **Cosmetics Data Storage**:
  - **Cosmetics** collection stores products related to beauty and cosmetics, similar to the **Products** collection.

Technologies and Libraries:

- MongoDB as the database.
- Mongoose for data modeling and querying.

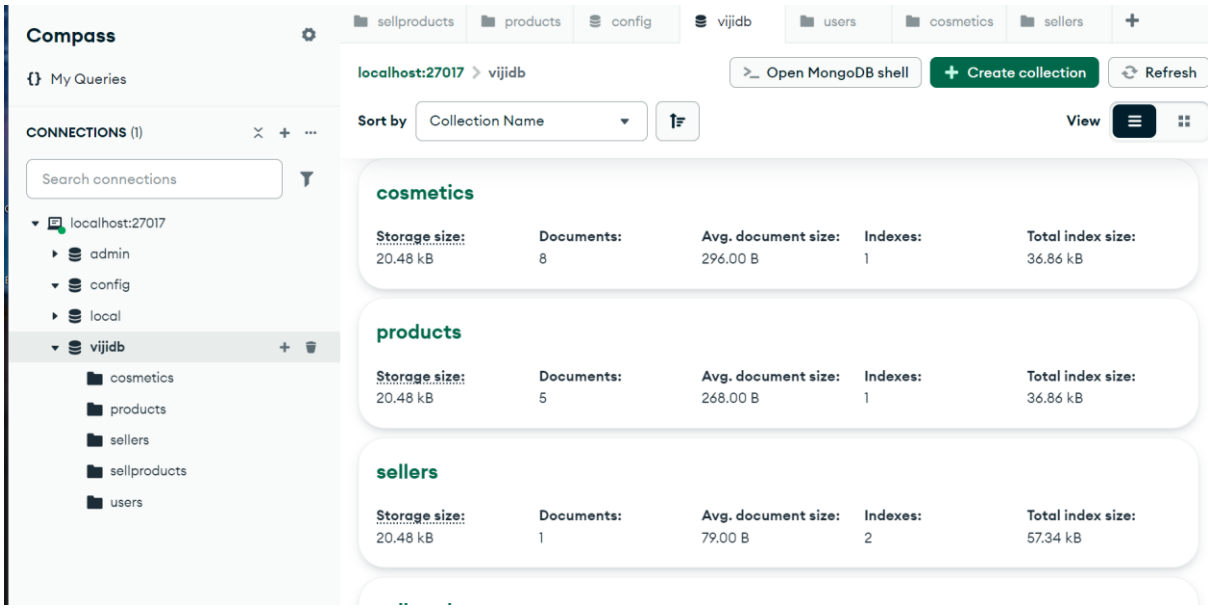

Fig:6.1 MongoDB Collections

7.RESULTS

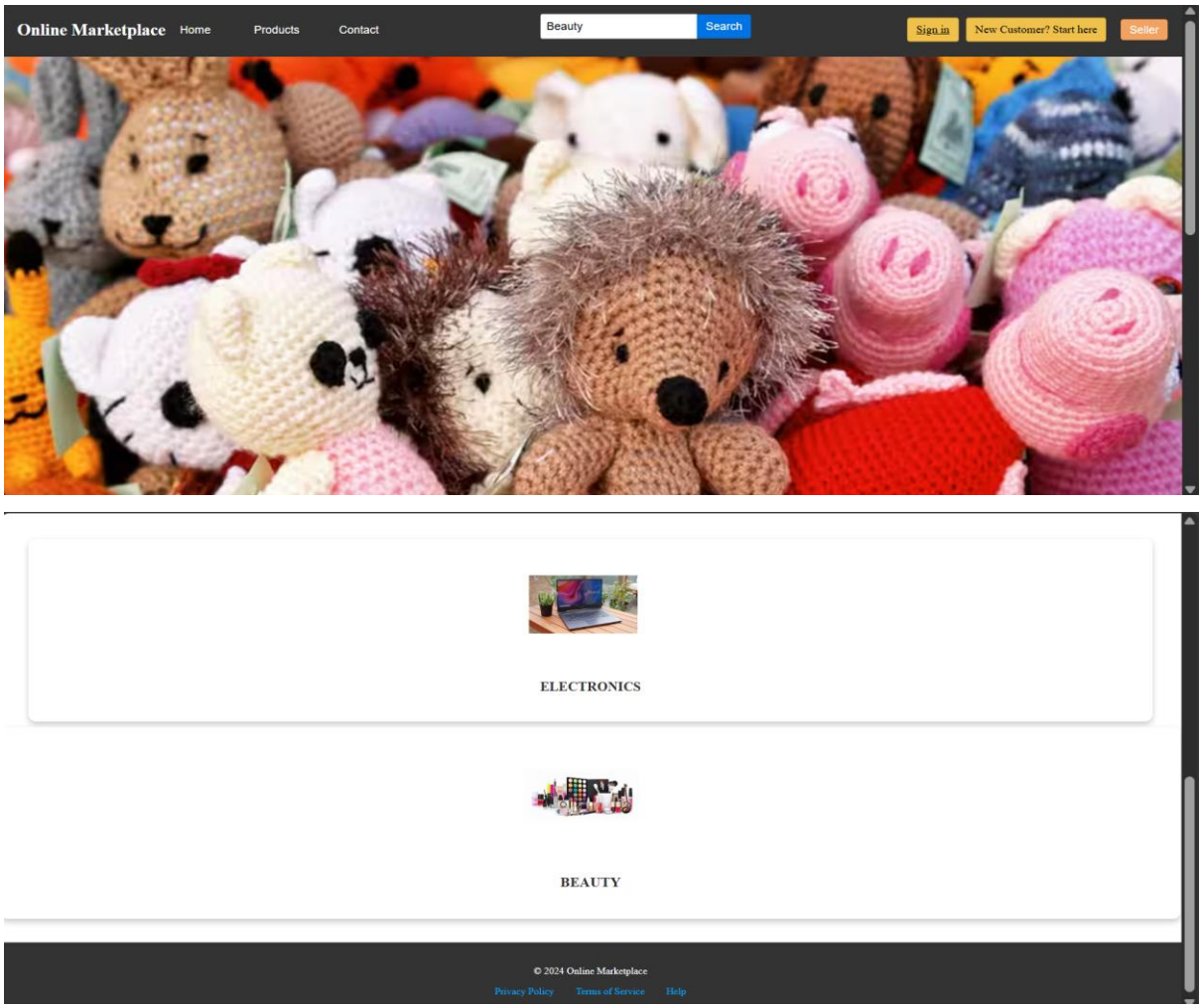

Fig:7.1 Home page

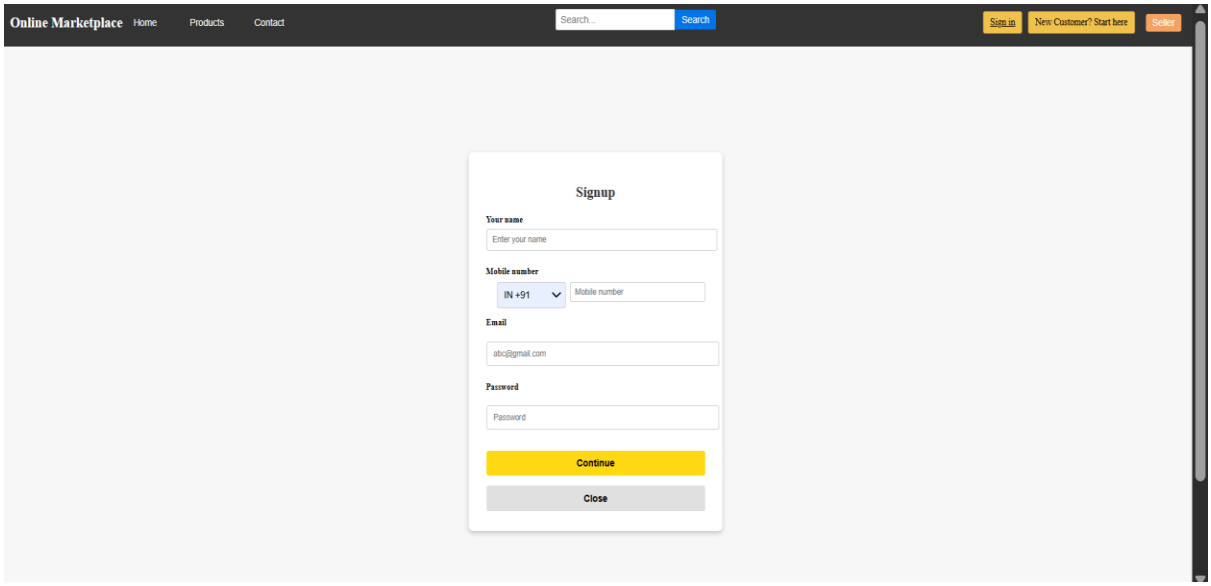

Fig:7.2 Sign up page

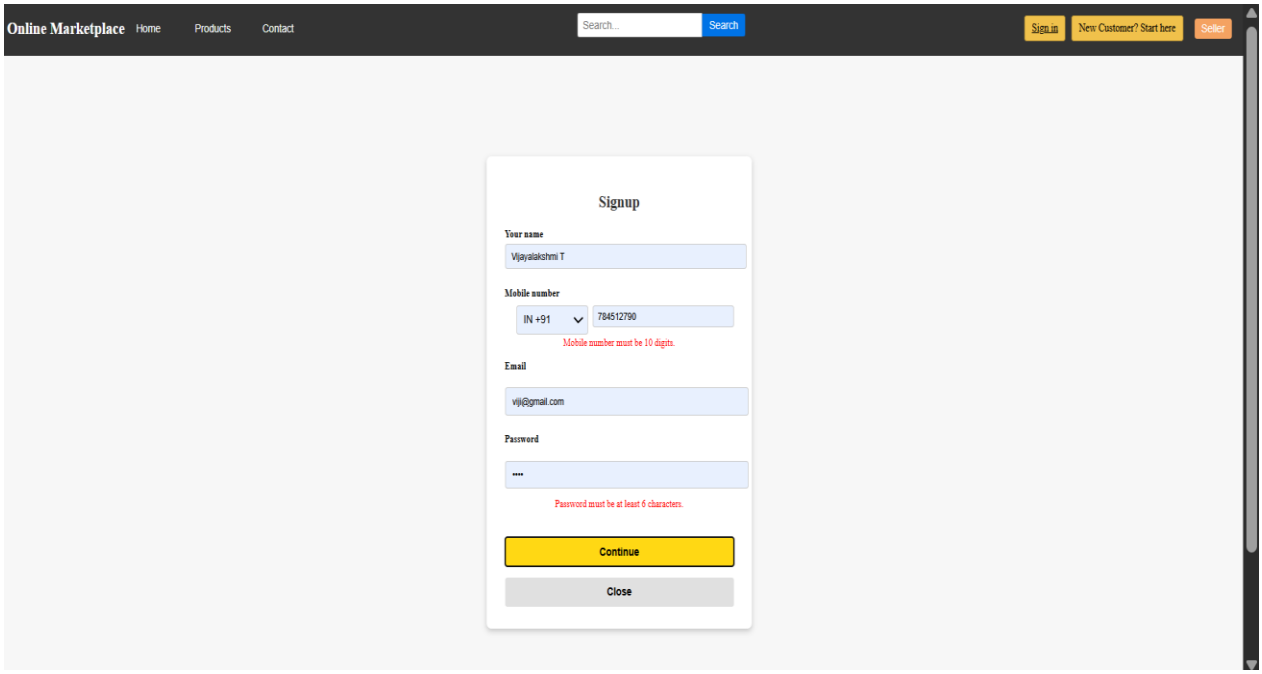

Fig:7.3 Sign up page validity

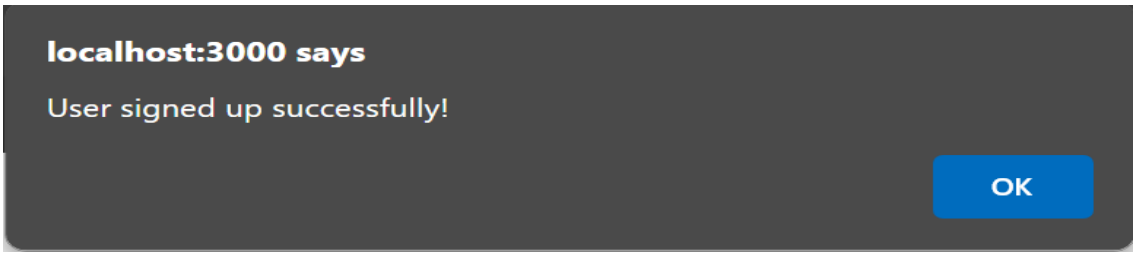

Fig:7.4 Successful user registration

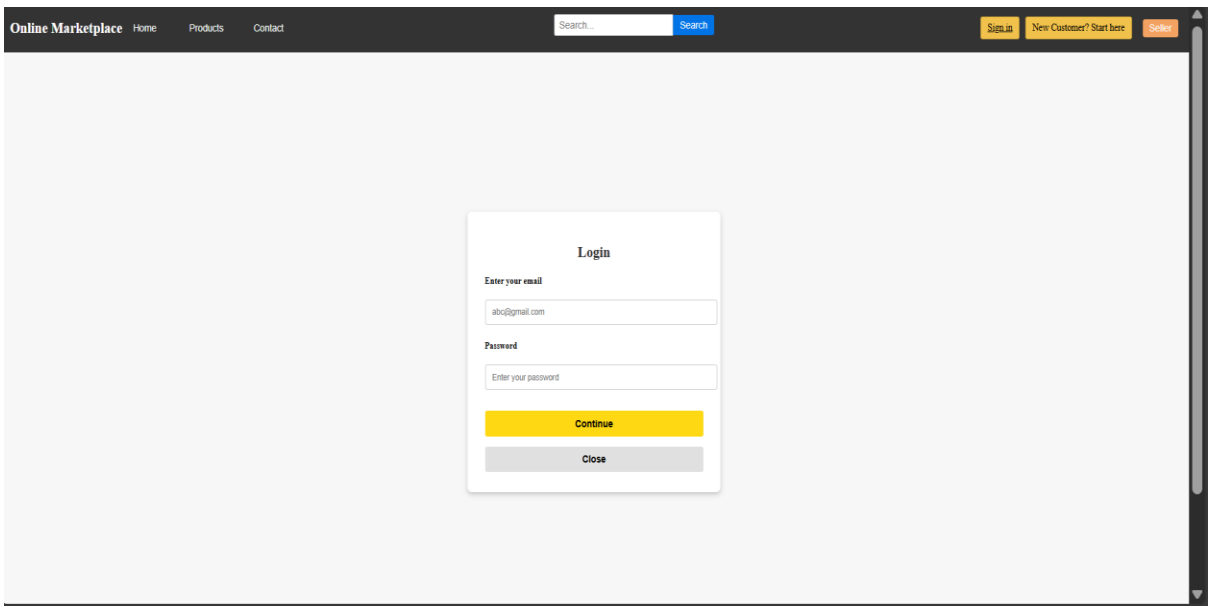

Fig:7.5 Login page

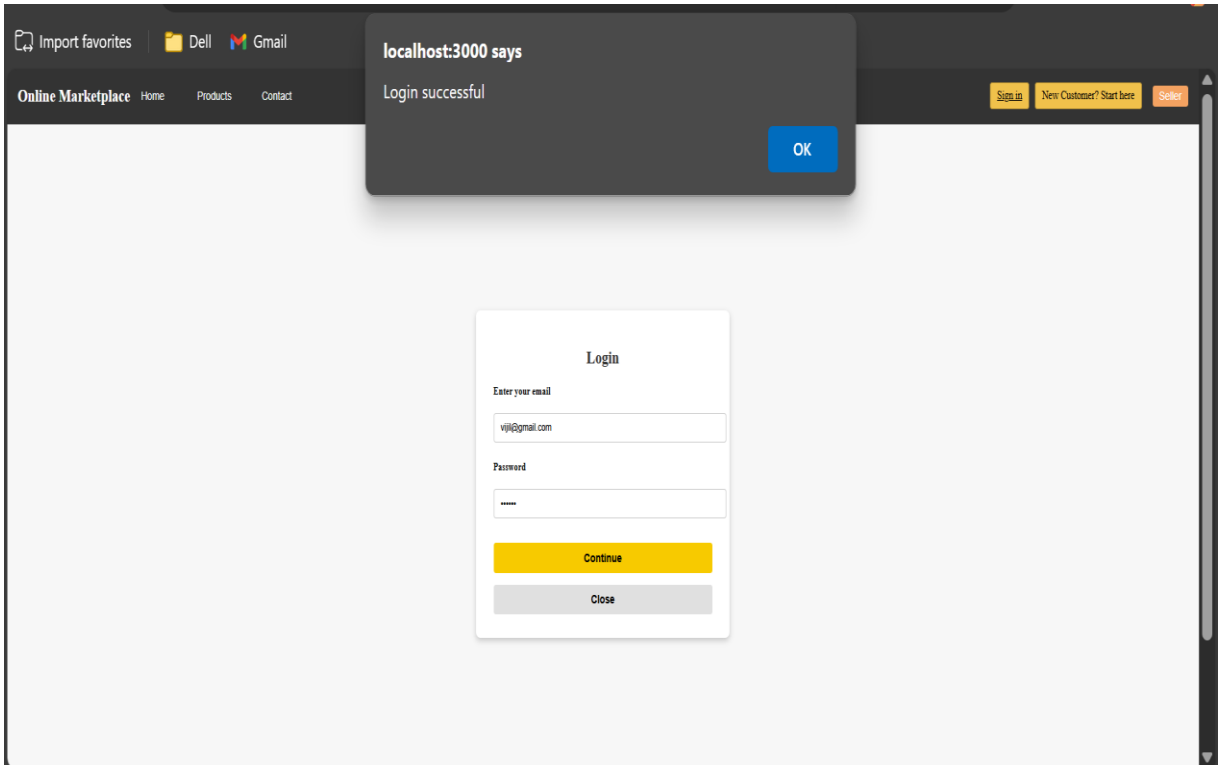

Fig:7.6 Successful Login

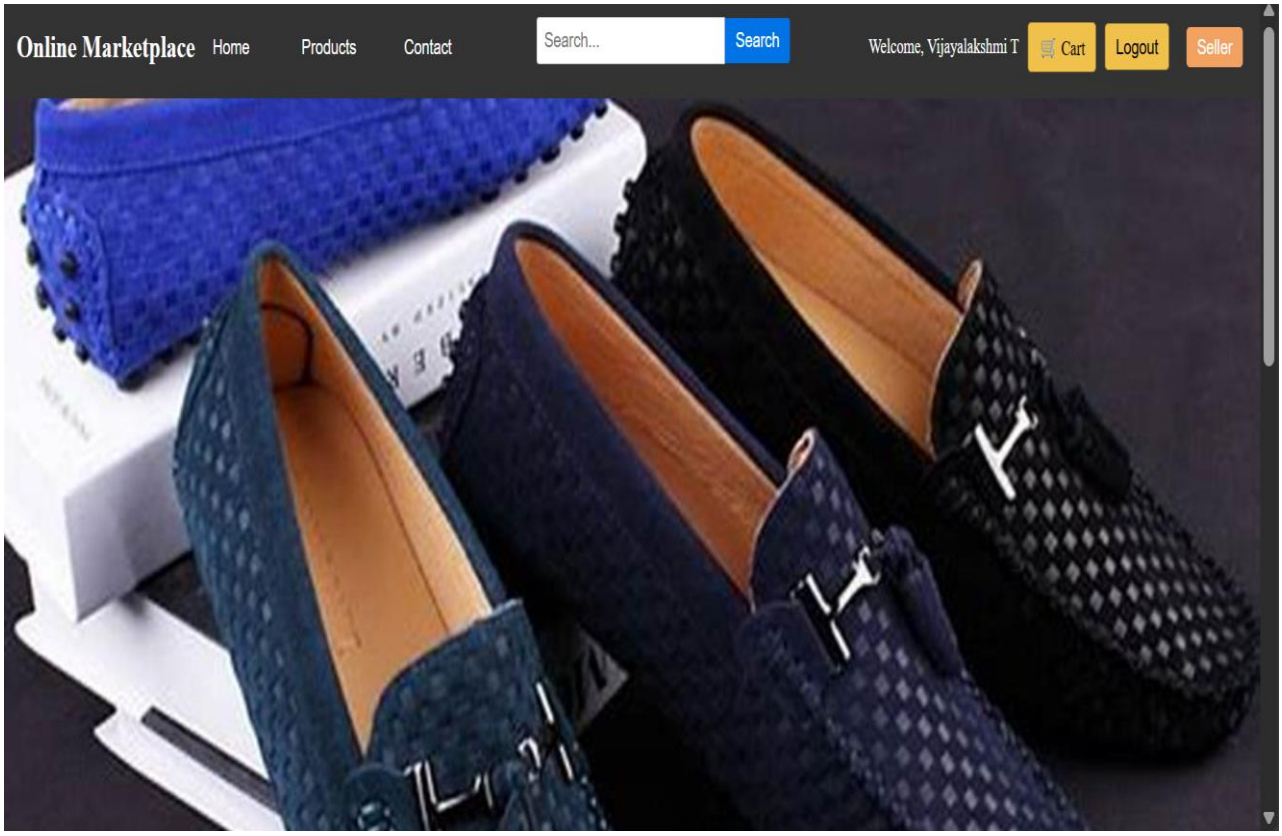

Fig:7.7 Home page after Successful Login

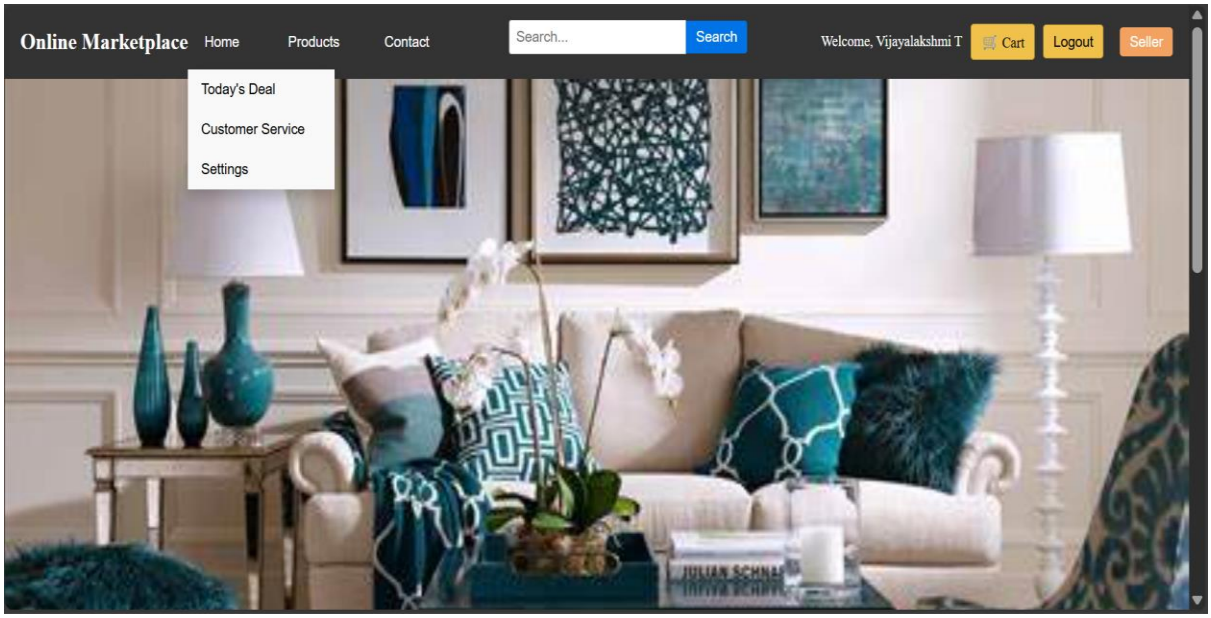

Fig:7.8 Home's Menu

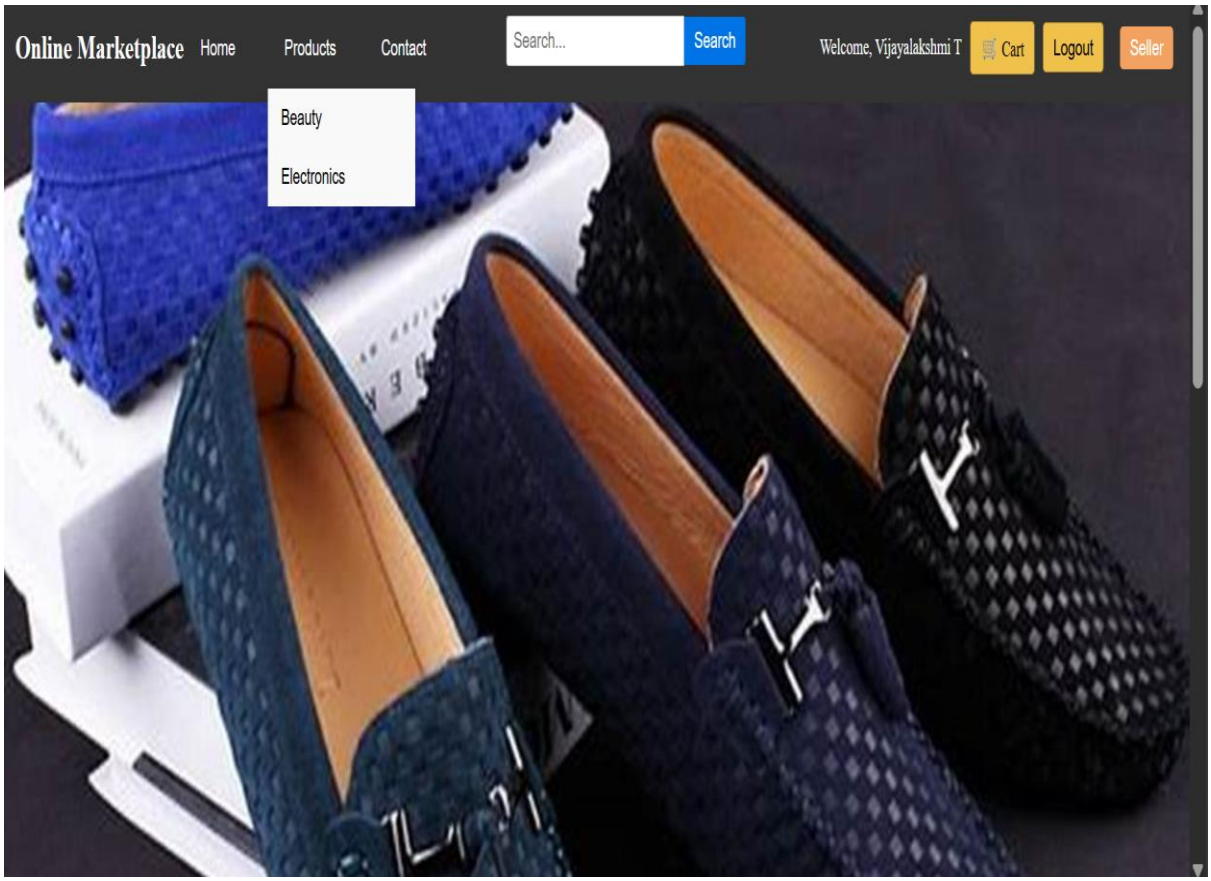

Fig:7.9 Products's Menu

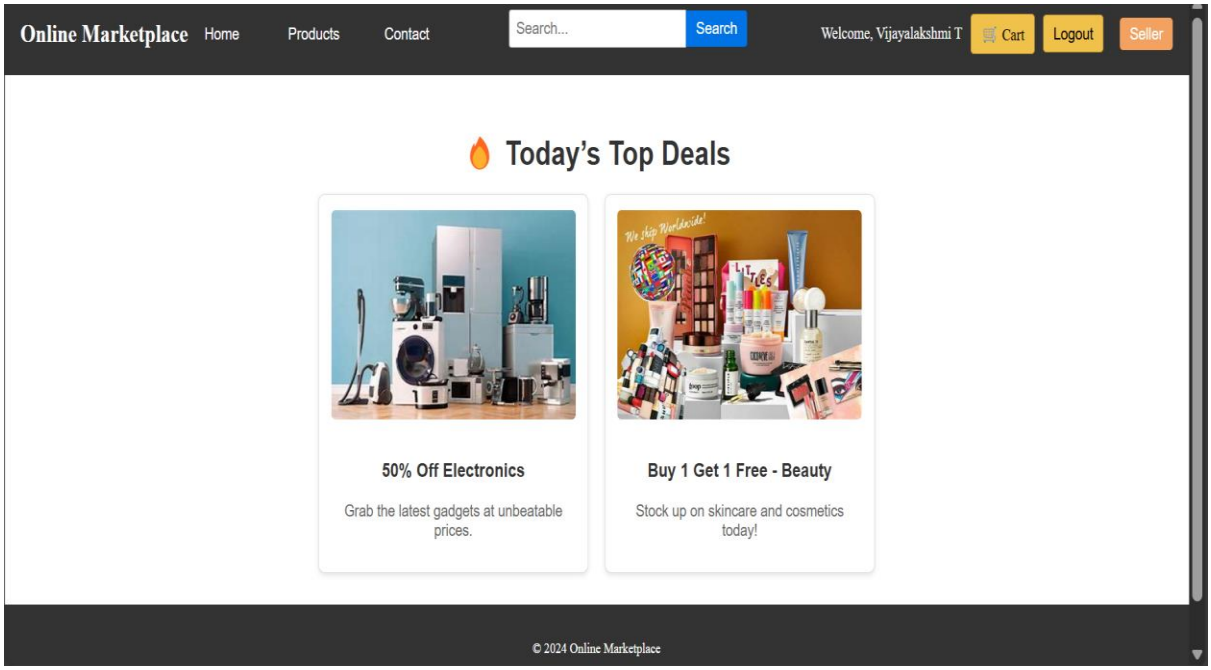

Fig:7.10 Today's Deal page

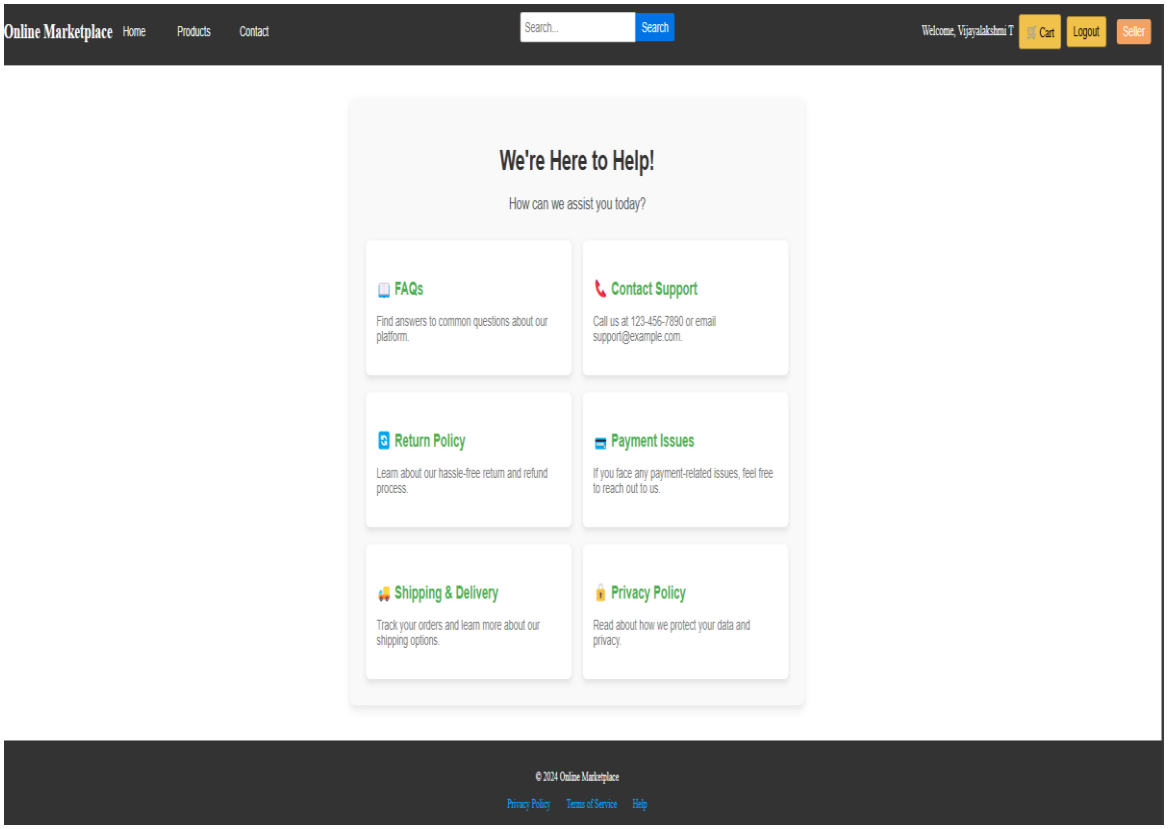

Fig:7.11 Customer Service Page

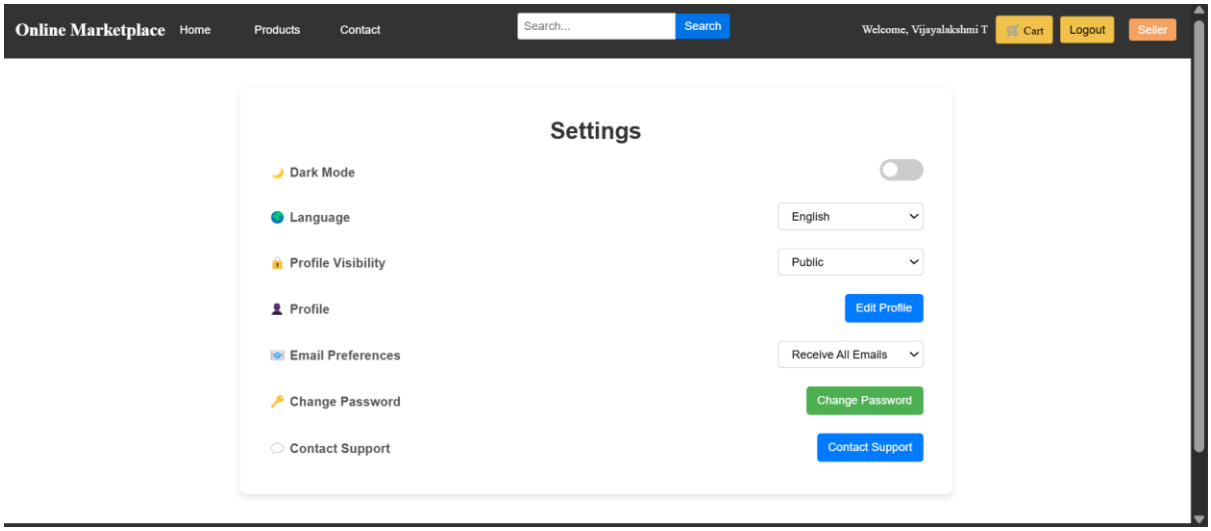

Fig:7.12 Settings Page

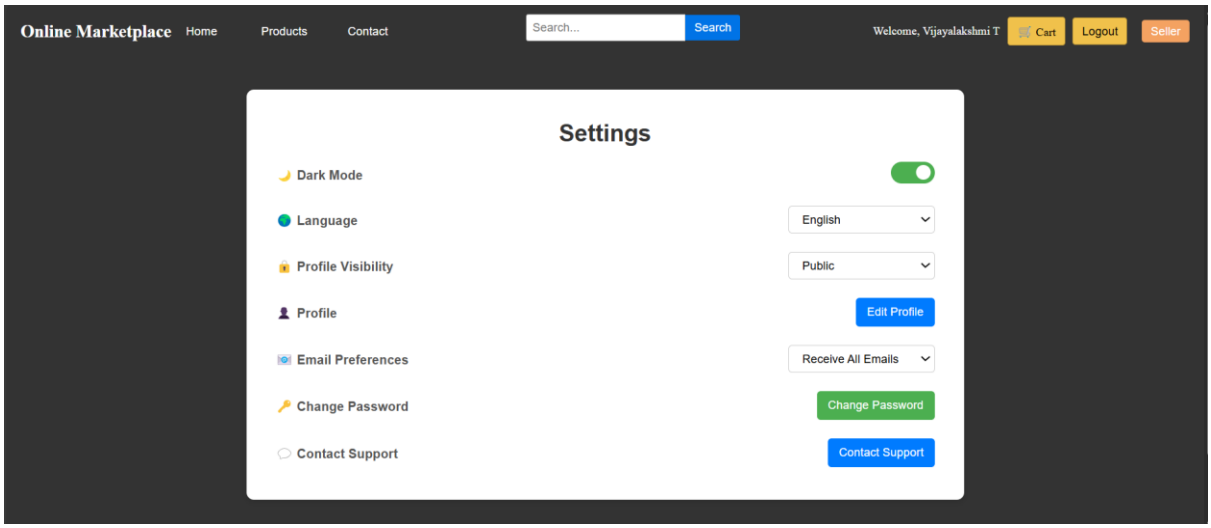

Fig:7.13 After enabling dark theme icon

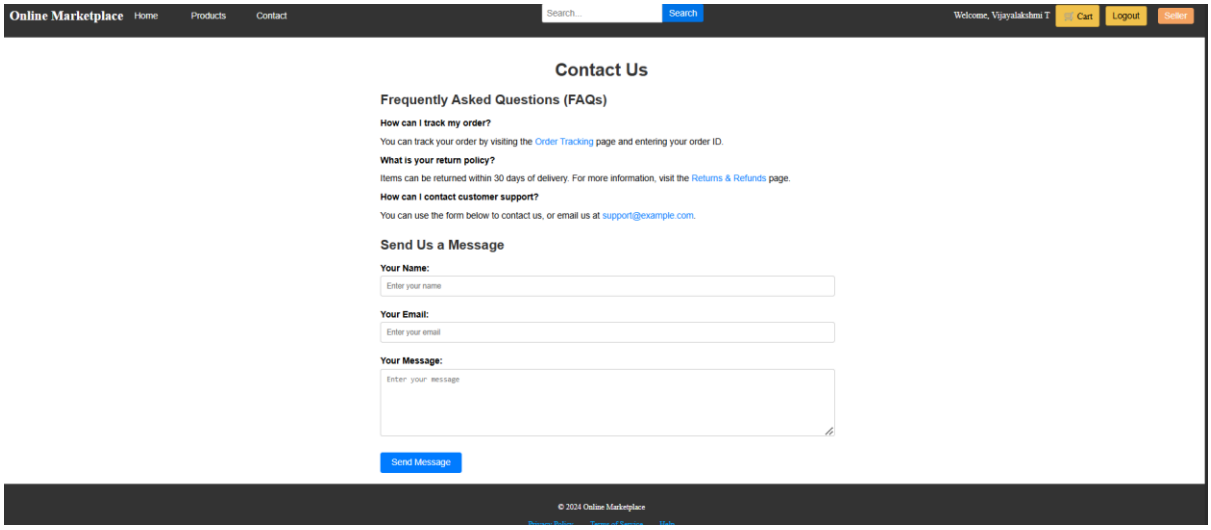

Fig:7.14 Contact page

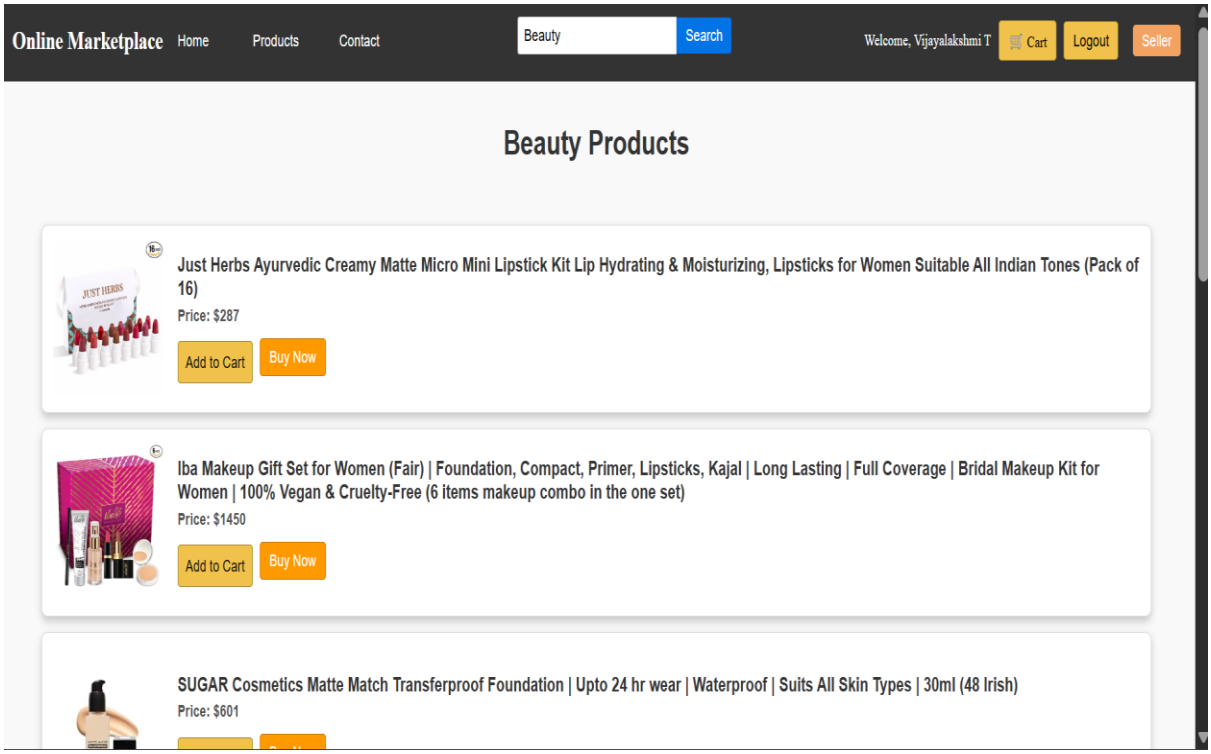

Fig:7.15 Beauty Products Page

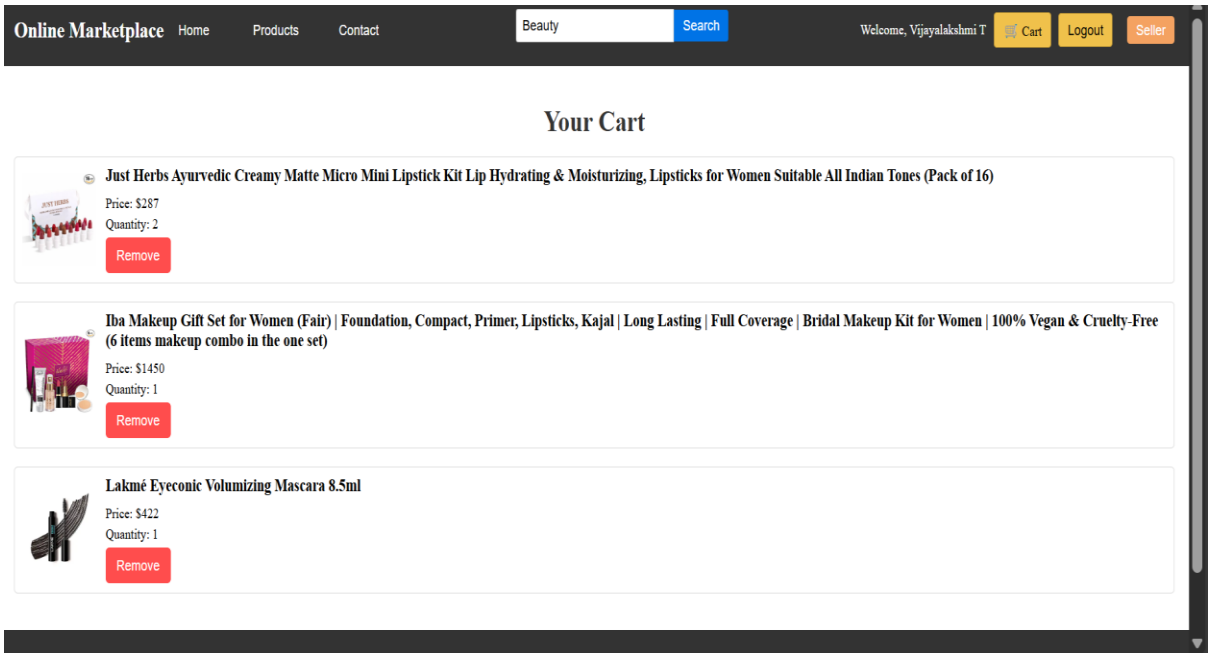

Fig:7.16 After adding beauty products to the cart

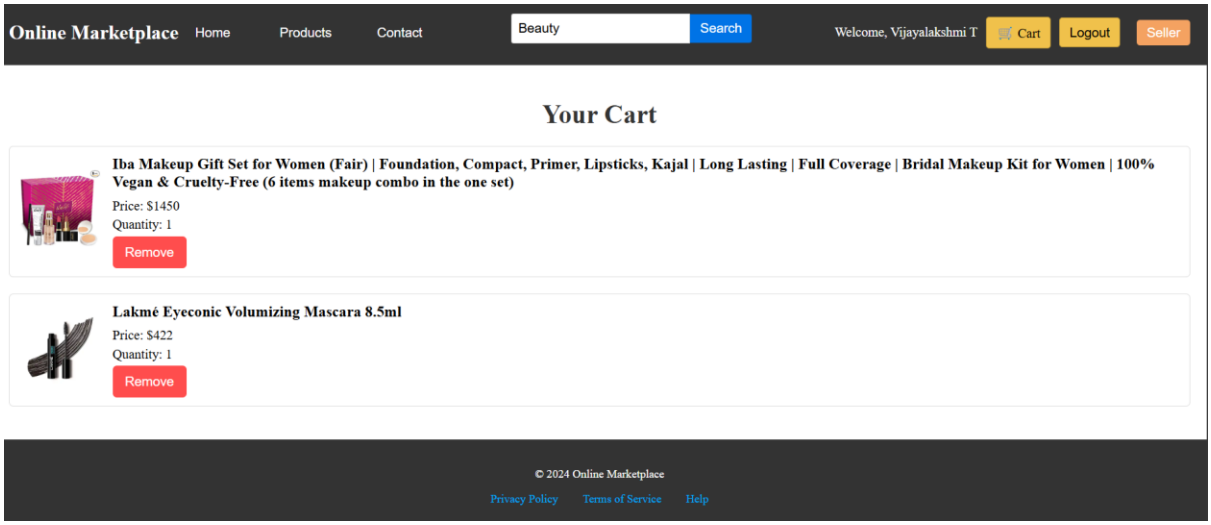

Fig:7.17 After Removing products

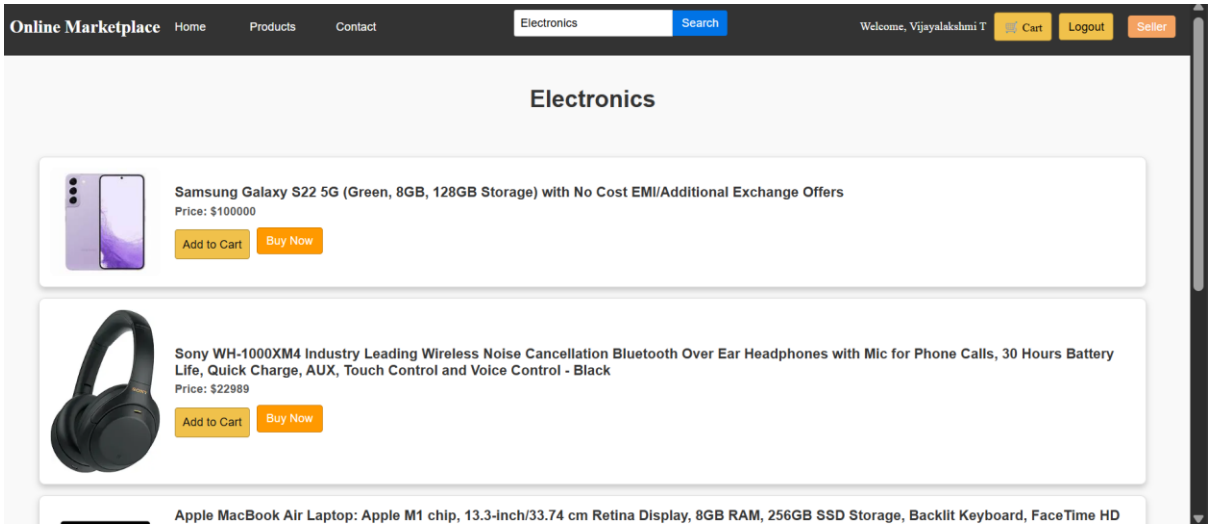

Fig:7.18 Electronics Page

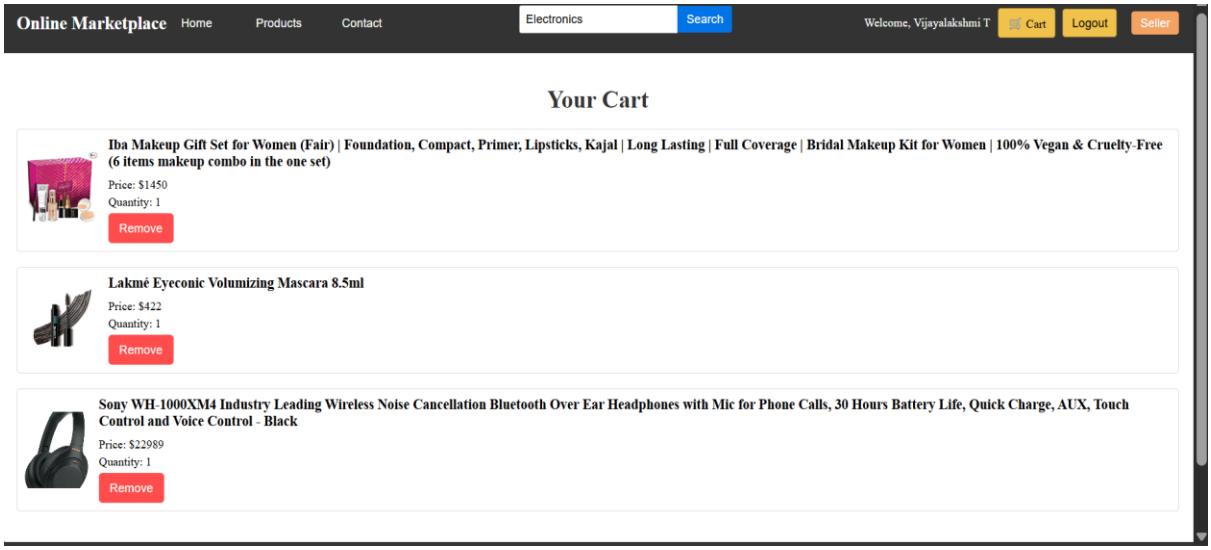

Fig:7.19 After adding Electronics Product to the cart

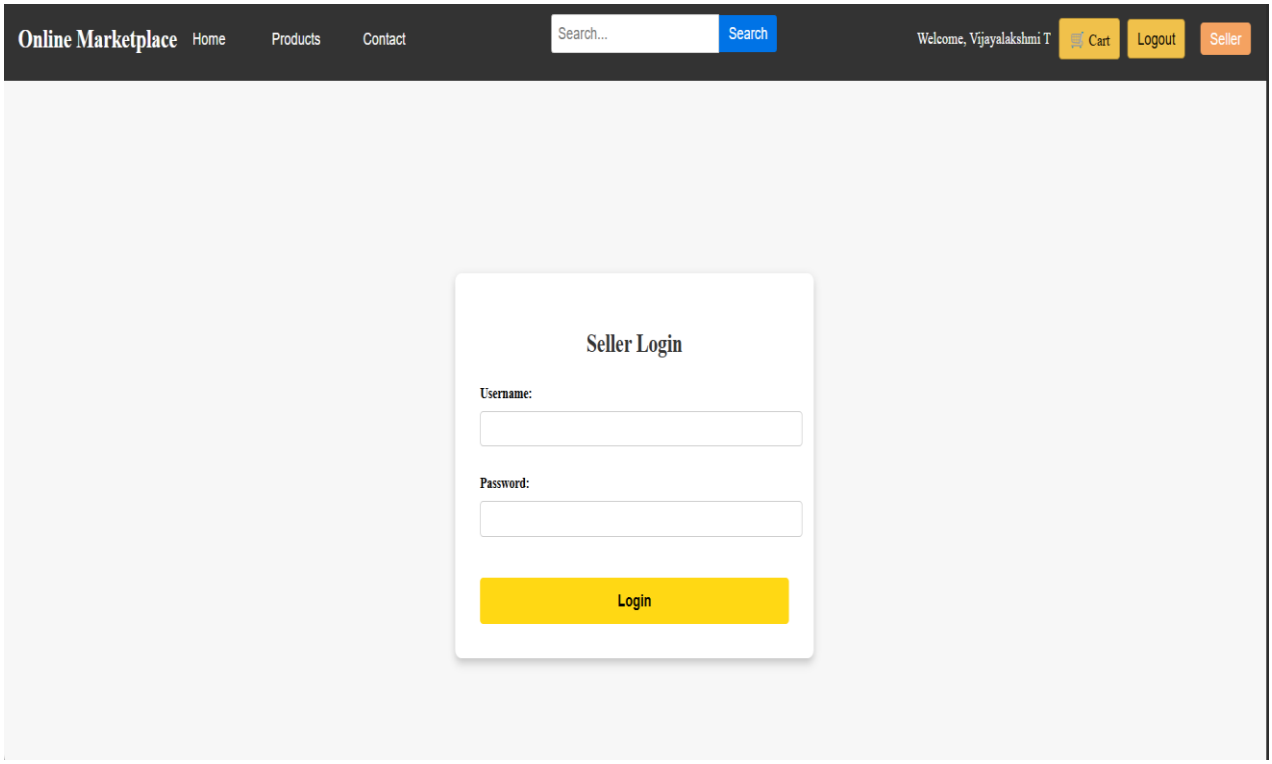

Fig:7.20 Seller Login to add the products

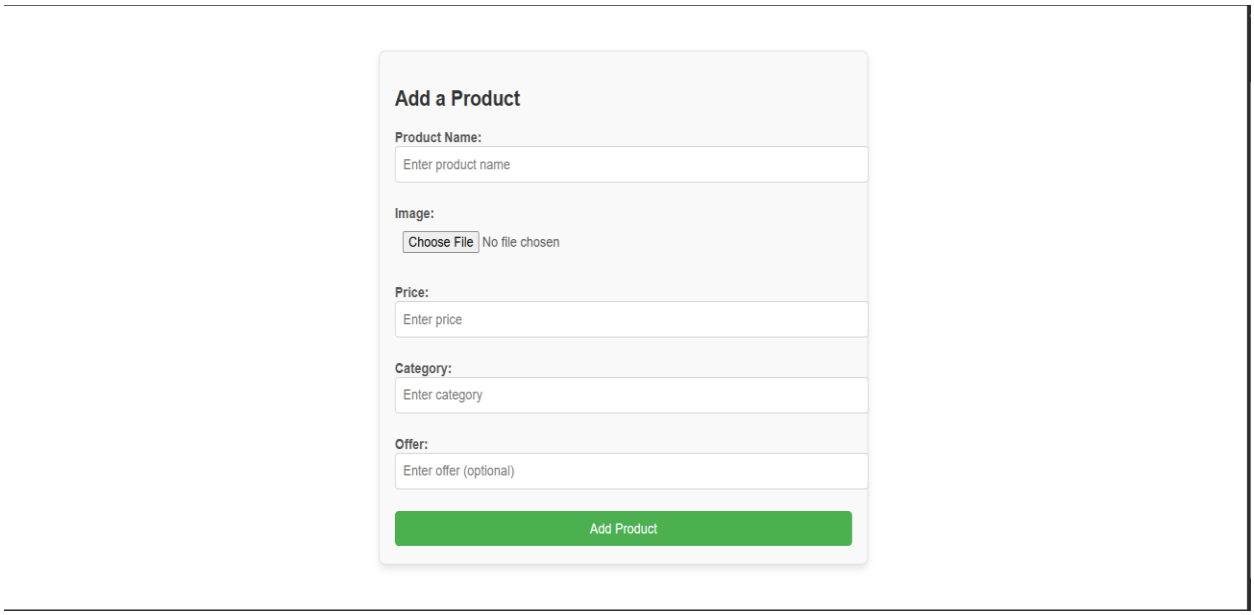

Fig:7.21 Add product page to add the product

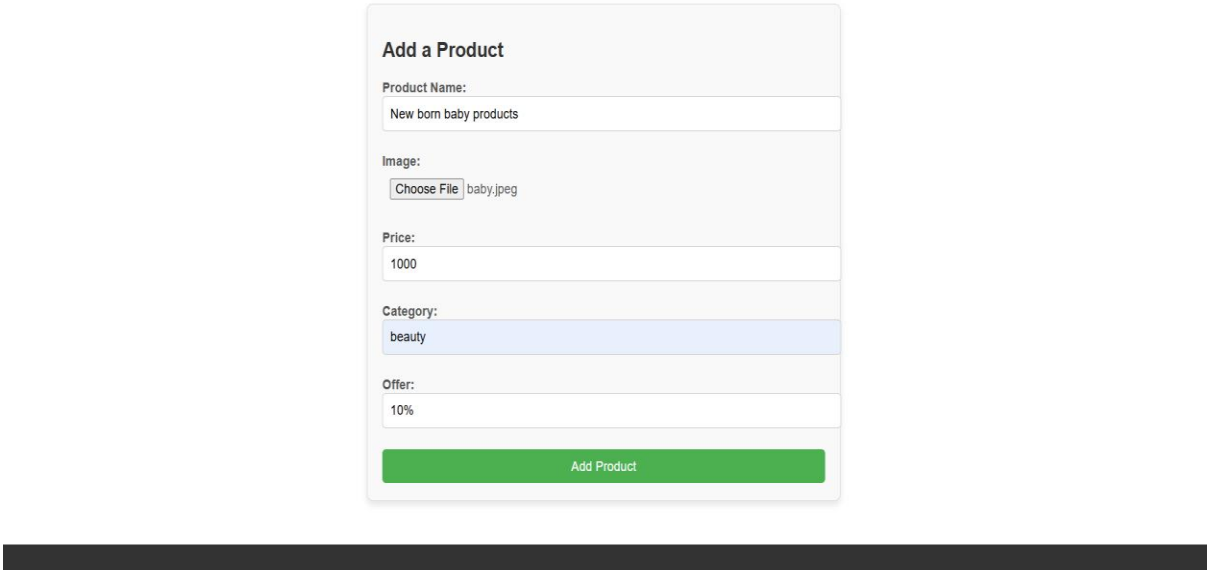

Fig:7.22 Filling Product Details

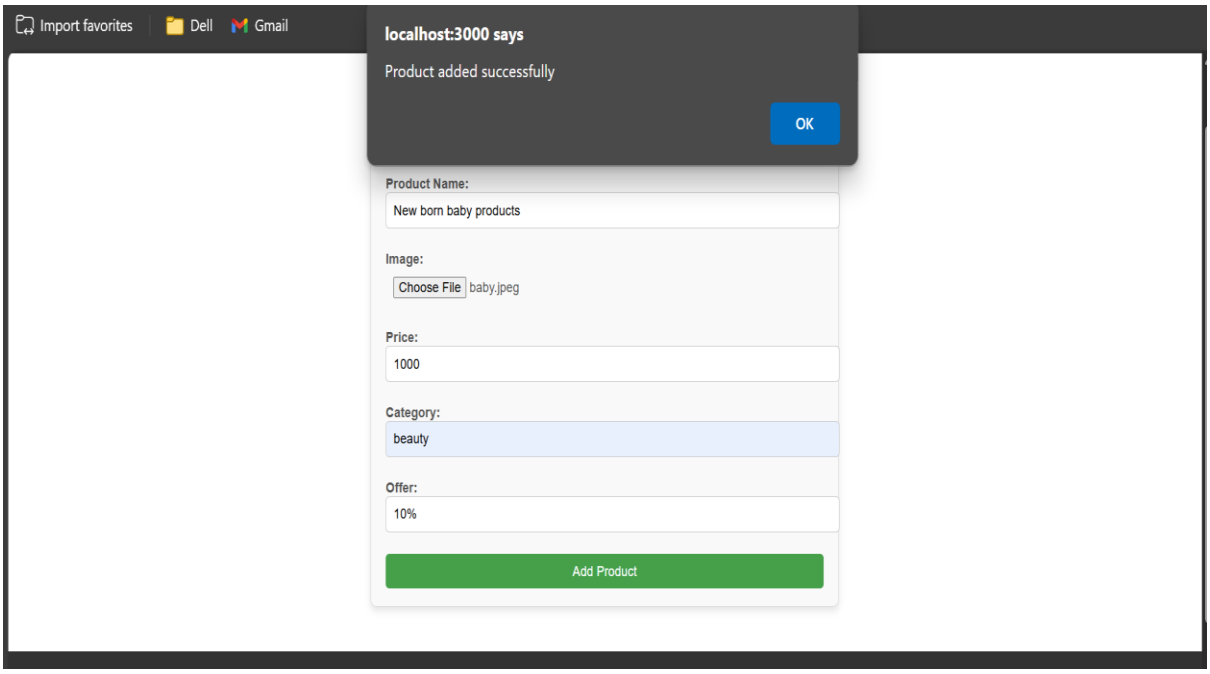

Fig:7.23 Success message for the product added

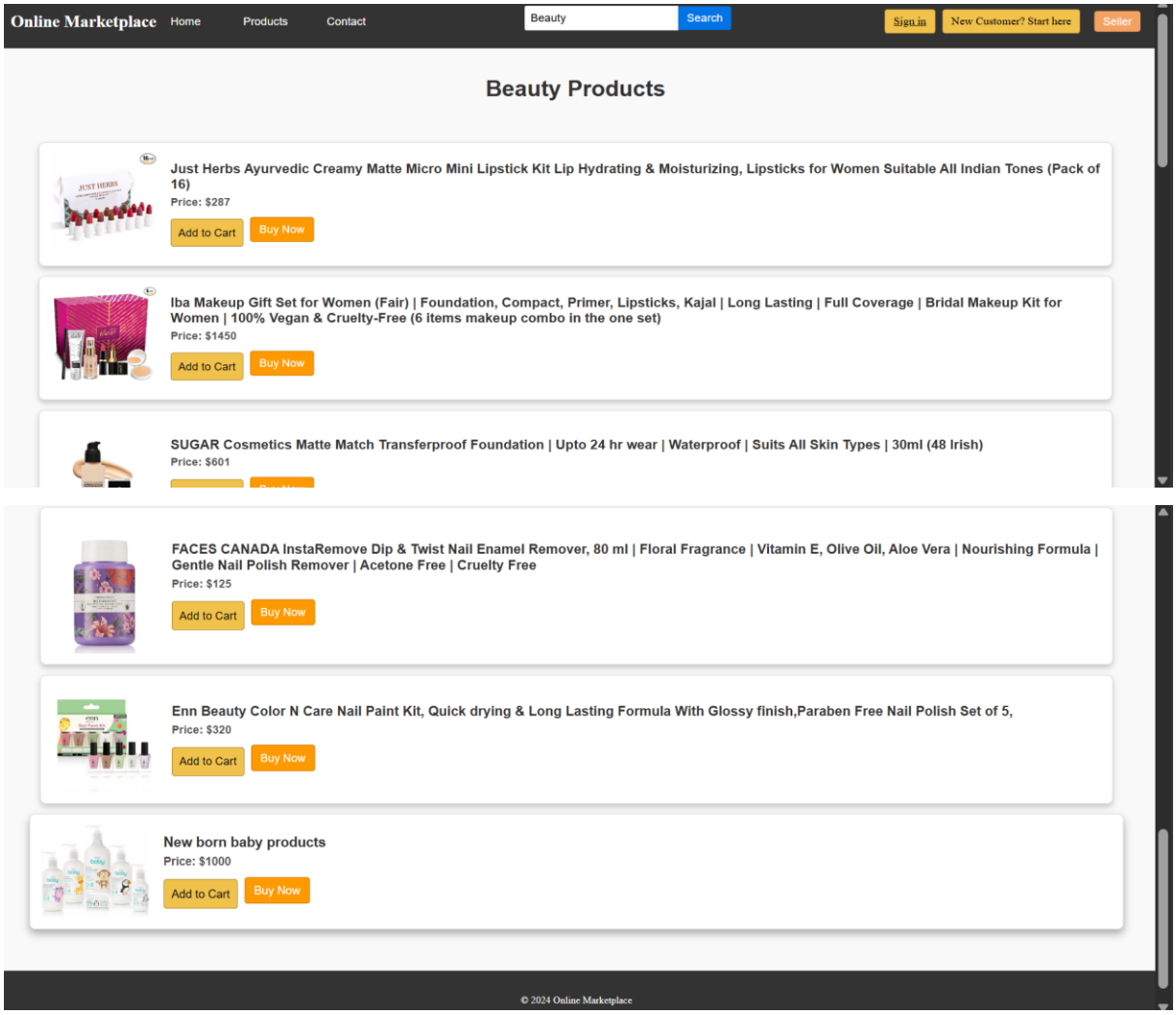

Fig:7.24 The New born baby products was added and displayed in the beauty page

```
C:\My react project\myfirstreactapp>node server.js
(node:38324) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Dri
ver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:38324) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.
js Driver version 4.0.0 and will be removed in the next major version
Server running on http://localhost:5000
Connected to MongoDB
Received product data: [Object: null prototype] {
  name: 'New born baby products',
  price: '1000',
  category: 'beauty',
  offer: '10%',
  sellerId: '6756b53fb16627b7080d8190'
}
Uploaded file: {
  fieldname: 'image',
  originalname: 'baby.jpeg',
  encoding: '7bit',
  mimetype: 'image/jpeg',
  destination: 'uploads/',
  filename: '1733850515073-baby.jpeg',
  path: 'uploads\\1733850515073-baby.jpeg',
  size: 18662
}
```

Fig:7.25 Console output

8.CHALLENGES AND LIMITATIONS

Frontend Challenges:

- **State Management Complexity:** Managing multiple states like user authentication, product categories, and dynamic filtering for the online marketplace.
- **Responsive Design:** Ensuring a consistent and user-friendly interface across various devices and screen sizes.
- **Error Handling:** Providing meaningful and comprehensive error messages for failed requests, invalid inputs, and connectivity issues.

Backend Challenges:

- **Data Validation and Security:** Ensuring secure data handling through encryption, password hashing, and JWT authentication..
- **Database Query Optimization:** Effectively managing the addition of products into dynamic categories like Electronics, Beauty, and others.
- **API Integration:** Smooth communication between the frontend and backend through well-structured RESTful APIs.

Database Challenges:

- **Data Consistency:** Ensuring consistency during CRUD operations, especially when managing relationships between users,sellers and products.
- **Scalability:** Ensuring the database can handle growing datasets as the number of users, sellers, and products increases.
- **Indexing and Searching:** Optimizing MongoDB queries for fast filtering and sorting based on category, price, and availability.

General Limitations:

- **Limited Real-Time Updates:** Optimizing MongoDB queries for fast filtering and sorting based on category, price, and availability.
- **Third-Party Integration:** There is no integration with third-party services like payment gateways or delivery systems.
- **Offline Functionality:** The application requires constant internet connectivity for full functionality.

9.CONCLUSION AND FUTURE SCOPE

Conclusion:

The **Online Marketplace** successfully demonstrates a comprehensive full-stack implementation using the MERN stack. It efficiently bridges the gap between buyers and sellers by offering secure authentication, dynamic product management, and a user-friendly shopping interface. The system's modular design ensures scalability, maintainability, and the potential for future enhancements. By addressing key challenges in online shopping, the project delivers a seamless and personalized user experience while enabling sellers to manage their inventory effectively.

Future Scope:

1. **Enhanced User Experience :** Incorporate advanced features like personalized product recommendations based on user preferences and browsing history.
2. **Real-Time Updates :** Enable real-time updates for product stock, discounts, and flash sales to enhance the shopping experience.
3. **Advanced Filters and Search :** Introduce advanced filtering options such as price range, delivery options, and seller ratings for more precise search results.
4. **Integration with Maps :** Add map integration (e.g., Google Maps) for displaying seller locations and enabling location-based product filtering.
5. **Mobile Application Development :** Develop dedicated mobile apps for Android and iOS platforms to increase accessibility and convenience.
6. **Multilingual Support :** Expand language options to cater to a diverse and global user base.
7. **Payment Gateway Integration :** Integrate secure payment systems to enable seamless transactions directly on the platform.
8. **Seller Analytics Dashboard :** Provide sellers with a dashboard to view sales performance, product trends, and user interactions for better decision-making.

By implementing these enhancements, the Online Marketplace will evolve into a more robust and comprehensive platform, catering to diverse user needs while staying competitive in the e-commerce domain.

10.REFERENCES

1. **React.js:** https://reactjs.org/
2. **Node.js:** https://nodejs.org
3. **Express.js:** https://expressjs.com/
4. **MongoDB:** https://www.mongodb.com/
5. **Mongoose:** https://mongoosejs.com/
6. **Axios:** https://axios-http.com/
7. **React Router:** https://reactrouter.com/

11.APPENDIX

11.1 CODE SNIPPETS:

FRONTEND:

App.js

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Header from './Header';
import Footer from './Footer';
import Login from './Login';
import Signup from './Signup';
import Shoppingpage from './Shoppingpage';
import CartPage from './CartPage';
import ProductList from './ProductList';
import ElectronicsPage from './ElectronicsPage';
import Beauty from './Beauty';
import SellerLogin from './SellerLogin';
import ContactUs from './ContactUs';
import Settings from './Settings';
import Todaysdeal from './Todaysdeal';
import CustomerService from './CustomerService';
import './App.css';
import SellProduct from './SellProduct';
function App() {
   const [cartItems, setCartItems] = React.useState([]);
   const addToCart = (product) => {
      setCartItems((prevItems) => {
         const existingItem = prevItems.find((item) => item.name === product.name);
         if (existingItem) {
            return prevItems.map((item) =>
               item.name === product.name
                  ? { ...item, quantity: (item.quantity || 1) + 1 }
                  : item
```

```
            );
        } else {
            return [...prevItems, { ...product, quantity: 1 }];
        }
    });
};
const removeFromCart = (productName) => {
    setCartItems((prevItems) => prevItems.filter((item) => item.name !== productName));
};
return (
    <Router>
        <div id="app-layout">
            <Header />
            <main>
                <Routes>
                    <Route path="/" element={<ProductList addToCart={addToCart} />} />
                    <Route path="/login" element={<Login />} />
                    <Route path="/signup" element={<Signup />} />
                    <Route
                        path="/shopping"
                        element={<Shoppingpage addToCart={addToCart} />}
                    />
                    <Route
                        path="/cart"
                        element={
                            <CartPage
                                cartItems={cartItems}
                                removeFromCart={removeFromCart}
                            />
                        }
                    />
                    <Route
                        path="/electronics"
```

```
                    element={<ElectronicsPage addToCart={addToCart} />}
                  />
                  <Route
                    path="/beauty"
                    element={<Beauty addToCart={addToCart} />}
                  />
                  <Route path="/contact" element={<ContactUs />} />
                  <Route path="/seller-login" element={<SellerLogin />} />
                  <Route path="/sellproduct" element={<SellProduct/>}/>
                  <Route path="/todays-deal" element={<Todaysdeal />} />
                  <Route path="/customer-service" element={<CustomerService />} />
                  <Route path="/settings" element={<Settings />} />
                </Routes>
              </main>
              <Footer />
            </div>
          </Router>
    );
}
export default App;
```

App.css

```
html, body {
  margin: 0;
  padding: 0;
  height: 100%;
  display: flex;
  flex-direction: column;
}
#root {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}
```

```
main {
  flex: 1; /* Ensures main content takes up available space */
}
footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 10px;
  font-size: 14px;
}
.cart-page {
  flex: 1; /* Make the cart-page take up the remaining space */
  padding: 20px;
}
header {
  background-color: #333;
  padding: 20px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  color: #fff;
}
header h1 {
  font-size: 1.8rem;
  color: #fff;
  margin: 0;
}
.navbar {
  display: flex;
  gap: 15px;
}
.navbar a {
  color: #cce7ff;
  text-decoration: none;
```

```
  font-weight: bold;
}
.navbar a:hover {
  color: #ffcc00;
}
.login-signup {
  display: flex;
  gap: 10px;
}
.login-signup button {
  background-color: #007bff;
  color: #fff;
  border: none;
  padding: 10px 15px;
  cursor: pointer;
  border-radius: 5px;
}
.login-signup button:hover {
  background-color: #0056b3;
}
.product-list {
  display: flex;
  justify-content: center;
  gap: 20px;
  padding: 40px;
}
.product-item {
  background-color: #fff;
  padding: 20px;
  text-align: center;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  transition: transform 0.2s;
}
```

```
.product-item:hover {
  transform: scale(1.05);
}
.product-item img {
  width: 150px;
  height: 150px;
  object-fit: contain;
  margin-bottom: 10px;
}
.product-item h3 {
  font-size: 1.2rem;
  color: #333;
}
.add-to-cart-btn {
  margin-top: 10px;
  background-color: #28a745;
  color: #fff;
  border: none;
  padding: 10px;
  border-radius: 5px;
  cursor: pointer;
}
.add-to-cart-btn:hover {
  background-color: #218838;
}
.footer {
  background-color: #333;
  color: #fff;
  text-align: center;
  padding: 20px;
  font-size: 14px;
}
.footer-links {
  margin-top: 10px;
```

```
}
.footer-links a {
  color: #00aaff;
  margin: 0 15px;
  text-decoration: none;
  font-weight: 500;
}
.footer-links a:hover {
  text-decoration: underline;
  color: #0077cc;
}
```

Header.js

```
import React, { useEffect, useState } from 'react';
import { Link, useLocation, useNavigate } from 'react-router-dom';
import './Header.css';
function Header() {
  const location = useLocation();
  const navigate = useNavigate();
  const [username, setUsername] = useState(localStorage.getItem('username'));
  const [searchQuery, setSearchQuery] = useState('');
  const [suggestions, setSuggestions] = useState([]);
  const categories = [
    'Beauty',
    'Electronics',
  ];
  useEffect(() => {
    const storedUsername = localStorage.getItem('username');
    setUsername(storedUsername);
  }, [location]);
  const handleLogout = () => {
    localStorage.removeItem('username');
    setUsername(null);
    navigate('/');
```

```
  };
  const handleSearchInput = (e) => {
    const query = e.target.value;
    setSearchQuery(query);
    if (query) {
      const filteredSuggestions = categories.filter((category) =>
        category.toLowerCase().startsWith(query.toLowerCase())
      );
      setSuggestions(filteredSuggestions);
    } else {
      setSuggestions([]);
    }
  };
  const handleSuggestionClick = (category) => {
    setSearchQuery(category);
    setSuggestions([]);
    navigate(`/${category.toLowerCase().replace(/ & /g, '-').replace(/ /g, '-')}`);
  };
  return (
    <header className="header">
      <div className="header-title">Online Marketplace</div>
      <nav className="nav">
        <div className="dropdown">
          <button className="dropbtn" onClick={() => navigate('/')}>
            Home
          </button>
          <div className="dropdown-content">
            <button className="dropdown-item" onClick={()=>navigate('/todays-deal')}>Today's
Deal</button>
                    <button    className="dropdown-item"    onClick={()=>navigate('/customer-
service')}>Customer Service</button>
                                                <button           className="dropdown-item"
onClick={()=>navigate('/settings')}>Settings</button>
```

```
  </div>
 </div>
 <div className="dropdown">
  <button className="dropbtn">Products</button>
  <div className="dropdown-content">
   {categories.map((category) => (
    <button
     key={category}
     className="dropdown-item"
     onClick={() =>
      navigate(`/${category.toLowerCase().replace(/ & /g, '-').replace(/ /g, '-')}`)
     }
    >
     {category}
    </button>
   ))}
  </div>
 </div>
 {/* Navigate to ContactUs page */}
 <button className="nav-link" onClick={() => navigate('/contact')}>
  Contact
 </button>
</nav>
<div className="search-bar">
 <div style={{ position: 'relative' }}>
  <input
   type="text"
   placeholder="Search..."
   value={searchQuery}
   onChange={handleSearchInput}
   className="search-input"
  />
  <button onClick={() => navigate(`/${searchQuery.toLowerCase()}`)}>Search</button>
```

```
    {suggestions.length > 0 && (
     <div className="suggestions-dropdown">
      {suggestions.map((suggestion, index) => (
       <div
        key={index}
        className="suggestion-item"
        onClick={() => handleSuggestionClick(suggestion)}
       >
        {suggestion}
       </div>
      ))}
     </div>
    )}
   </div>
  </div>
  <div className="header-actions">
   {username ? (
    <>
     <span className="welcome-message">Welcome, {username}</span>
     <Link to="/cart" className="cart-button">
      🛒 Cart
     </Link>
     <button className="logout-button" onClick={handleLogout}>
      Logout
     </button>
    </>
   ) : (
    <>
     <Link to="/login" className="login-button">
      Sign in
     </Link>
     <Link to="/signup" className="signup-button">
      New Customer? Start here
```

```
          </Link>
        </>
      )}
      <button className="seller-button" onClick={() => navigate('/seller-login')}>
        Seller
      </button>
    </div>
  </header>
 );
}
export default Header;
```

Header.css

```
/* Fullscreen Banner */
.banner {
 width: 100%;
 height: 100vh; /* Full viewport height */
 overflow: hidden;
 position: relative;
 margin-bottom: 20px; /* Spacing between banner and product list */
}
.banner-image {
 width: 100%;
 height: 100%;
 object-fit: cover; /* Or try 'contain' if 'cover' is cropping */
}
/* Header Styles */
.header {
 width: 100%;
 display: flex;
 justify-content: space-between;
 align-items: center;
 padding: 10px 20px;
 background-color: #333;
```

```
  color: #fff;
  box-sizing: border-box;
}
.header-title {
  font-size: 24px;
  font-weight: bold;
}
/* Navigation Styles */
.nav {
  display: flex;
  gap: 20px;
}
.nav .dropdown {
  position: relative;
}
.nav .dropbtn {
  background-color: #333;
  color: white;
  padding: 20px;
  border: none;
  font-size: 16px;
  cursor: pointer;
}
.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 180px;
  box-shadow: 0px 8px 16px rgba(0, 0, 0, 0.2);
  z-index: 1;
  left: 0;
}
.dropdown-content .dropdown-item {
```

```
  background: none;
  border: none;
  color: black;
  padding: 12px 16px;
  text-align: left;
  cursor: pointer;
  width: 100%;
}
.dropdown-content .dropdown-item:hover {
  background-color: #ddd;
}
.dropdown:hover .dropdown-content {
  display: block;
}
.nav-link {
  background: none;
  color: white;
  border: none;
  font-size: 16px;
  cursor: pointer;
}
/* Search bar container */
.search-bar {
  display: flex;
  align-items: center; /* Align input and button vertically */
  gap: 10px; /* Space between input and button */
}
/* Style for the search input */
.search-input {
  width: 200px; /* Adjust the width as needed */
  padding: 5px; /* Add some padding for a better look */
  border: 1px solid #ccc;
  border-radius: 4px;
```

```
}
/* Style for the search button */
.search-button {
  padding: 6px 12px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
.search-button:hover {
  background-color: #0056b3;
}
.suggestions-dropdown {
  position: absolute;
  top: 40px; /* Adjust to appear below the input */
  left: 0;
  width: 100%; /* Match the width of the input field */
  background-color: white;
  border: 1px solid #ddd;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
  z-index: 1000;
  max-height: 200px; /* Scrollable for long lists */
  overflow-y: auto;
  border-radius: 4px;
}
.suggestion-item {
  padding: 10px;
  cursor: pointer;
  font-size: 14px;
  color: #333;
}
.suggestion-item:hover {
```

```
  background-color: #f0f0f0;
  color: #000;
}
.search-bar {
  display: flex; /* Use flexbox for alignment */
  align-items: center; /* Vertically align the items */
  width: 300px; /* Adjust as needed */
  margin: 0 auto; /* Center the search bar */
}
.search-bar input[type="text"] {
  flex: 1; /* Let the input take up available space */
  padding: 8px;
 font-size: 16px;
  border: 1px solid #ccc;
  border-radius: 3px 0 0 3px; /* Round left corners */
  outline: none;
}
.search-bar button {
  padding: 8px 12px;
  background-color: #0073e6;
  color: white;
  border: none;
  border-radius: 0 3px 3px 0; /* Round right corners */
  cursor: pointer;
  font-size: 16px;
}
/* Authentication Buttons */
.auth-buttons {
  display: flex;
  gap: 10px;
}
.auth-buttons .login,
.auth-buttons .signup {
```

```
  background-color: #0073e6;
  color: white;
  border: none;
  padding: 8px 12px;
  cursor: pointer;
  border-radius: 3px;
}
.auth-buttons .signup {
  background-color: #ff5722;
}
.auth-buttons .login:hover,
.auth-buttons .signup:hover {
  opacity: 0.9;
}
/* Modal Button Hover Effects */
.auth-buttons .login:hover,
.auth-buttons .signup:hover {
  opacity: 0.9;
  transition: opacity 0.3s ease;
}
.header-actions {
  display: flex;
  align-items: center;
  gap: 10px; /* Adds space between all children in .header-actions */
}
.cart-button {
  background-color: #f0c14b;
  border: 1px solid #a88734;
  padding: 8px 12px;
  text-decoration: none;
  color: black;
  border-radius: 4px;
  cursor: pointer;
```

```
}
.logout-button {
  background-color: #f0c14b;
  border: 1px solid #a88734;
  padding: 8px 12px;
  color: black;
  border-radius: 4px;
  cursor: pointer;
}
.signup-button {
  background-color: #f0c14b;
  border: 1px solid #a88734;
  padding: 8px 12px;
  text-decoration: none;
  color: black;
  border-radius: 4px;
  cursor: pointer;
}
.todays-deal-container {
  padding: 20px;
  font-family: Arial, sans-serif;
}
.todays-deal-container h2 {
  margin-bottom: 15px;
}
.todays-deal-container ul {
  list-style: none;
  padding: 0;
}
.todays-deal-container li {
  margin-bottom: 20px;
  padding: 10px;
  border: 1px solid #ddd;
```

```
  border-radius: 5px;
}
.login-button {
  background-color: #f0c14b;
  border: 1px solid #a88734;
  padding: 8px 12px;
  color: black;
  border-radius: 4px;
  cursor: pointer;
}
.seller-button {
  padding: 6px 12px;
  background-color: #f4a261; /* Unique color for seller button */
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  margin-left: 10px;
}
.seller-button:hover {
  background-color: #e76f51;
}
.cart-button:hover, .logout-button:hover,.login-button:hover,.signup-button:hover {
  background-color: #ddb347;
}
/* Additional Styles */
.header .nav-link,
.header .auth-buttons button,
.header .search-bar button {
  transition: background-color 0.3s ease;
}
```

Footer.js

```
import React from 'react';
```

```
import './App.css';
function Footer() {
  return (
    <footer className="footer">
      <p>© 2024 Online Marketplace</p>
      <div className="footer-links">
        <a href="#privacy">Privacy Policy</a>
        <a href="#terms">Terms of Service</a>
        <a href="#help">Help</a>
      </div>
    </footer>
  );
}
export default Footer;
```

Footer.css

```
/* src/components/Footer.css */
.footer {
  background-color: #333;
  color: white;
  padding: 20px 0;
  text-align: center;
}
.footer p {
  margin: 0;
  font-size: 0.9em;
}
.footer ul {
  list-style-type: none;
  padding: 0;
  margin: 10px 0;
}
.footer ul li {
  display: inline;
```

```
    margin-right: 15px;
  }
  .footer ul li a {
    color: white;
    text-decoration: none;
    font-size: 0.9em;
  }
  .footer ul li a:hover {
    text-decoration: underline;
  }
```

ProductItem.js

```
import React from 'react';
function ProductItem({ name, imageUrl, onAddToCart }) {
    return (
        <div className="product-item">
            <img src={imageUrl} alt={name} />
            <h2>{name}</h2>
            <button className="add-to-cart-btn" onClick={onAddToCart}>
                Add to Cart
            </button>
        </div>
    );
}
export default ProductItem;
```

ProductItem.css

```
/* src/components/ProductItem.css */
.product-item {
    width: 200px;
    padding: 15px;
    border: 1px solid #ddd;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.2);
    text-align: center;
```

```
}
.product-item img {
  width: 100%;
  height: auto;
  border-radius: 4px;
}
.product-item h2 {
  font-size: 1em;
  margin: 10px 0;
}
.product-item p {
  font-size: 0.9em;
}
.price {
  font-weight: bold;
  color: #b12704;
}
button {
  background-color: #f0c14b;
  border: 1px solid #a88734;
  color: #111;
  padding: 5px;
  border-radius: 4px;
  cursor: pointer;
}
```

ProductList.js

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom'; // Import useNavigate
function ProductList({ addToCart }) {
  const navigate = useNavigate(); // Initialize navigate function
                    const [bannerImage, setBannerImage] = useState('https://cdn.cdnparenting.com/articles/2018/03/518478826-H.jpg');
  const bannerImages = [
```

```
    'https://cdn.cdnparenting.com/articles/2018/03/518478826-H.jpg',
                                                    'https://th.bing.com/th/id/OIP.JzmvlxsV9-x9wvit9zCfYQHaHa?w=1000&h=1000&rs=1&pid=ImgDetMain',
     'https://th.bing.com/th/id/OIP.WbV_ndlvIIP0nOiSJAerHQHaFT?rs=1&pid=ImgDetMain',
  ];
  // Change banner image every 4 seconds
  useEffect(() => {
    const intervalId = setInterval(() => {
      setBannerImage(prevImage => {
        const currentIndex = bannerImages.indexOf(prevImage);
        const nextIndex = (currentIndex + 1) % bannerImages.length;
        return bannerImages[nextIndex];
      });
    }, 4000);
    return () => clearInterval(intervalId); // Cleanup interval on unmount
  }, [bannerImages]);
  const products = [
                                    { name: 'ELECTRONICS', imageUrl: 'https://th.bing.com/th/id/OIP.WCCq2nZelTZuFIRbJF7AuAHaEK?rs=1&pid=ImgDetMain', route: '/electronics' },
                                    { name: 'BEAUTY', imageUrl: 'https://tse3.mm.bing.net/th?id=OIP.sV40XtGYkcO4TFNzxD9JIwHaDt&pid=Api&P=0&h=180', route: '/beauty' },
  ];
  return (
    <div>
      {/* Banner Section */}
      <div className="banner">
        <img src={bannerImage} alt="Banner" className="banner-image" />
      </div>
      {/* Product List */}
      <div className="product-list">
```

```
            {products.map((product, index) => (
              <div
                key={index}
                className="product-item"
                onClick={() => navigate(product.route)} // Navigate to the category page
              >
                <img src={product.imageUrl} alt={product.name} className="product-image"
/>
                <h3 className="product-name">{product.name}</h3>
              </div>
            ))}
          </div>
        </div>
    );
}
export default ProductList;
```

ElectronicsPage.js

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import './ElectronicPage.css';
function ElectronicsPage({ addToCart }) {
    const [products, setProducts] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
    const navigate = useNavigate();
    useEffect(() => {
        fetch('http://localhost:5000/api/products')
            .then((response) => {
                if (!response.ok) {
                    throw new Error('Error fetching products');
                }
                return response.json();
            })
```

```
        .then((data) => {
            setProducts(data);
            setLoading(false);
        })
        .catch((error) => {
            console.error('Error fetching products:', error);
            setError('Error fetching products.');
            setLoading(false);
        });
}, []);
return (
    <div className="electronics-page">
        <h1>Electronics</h1>
        {loading && <p>Loading products...</p>}
        {error && <p>{error}</p>}
        <div className="product-list">
            {products.length > 0 ? (
                products.map((product, index) => (
                    <div key={index} className="product-card">
                        <img
                            src={product.imageUrl}
                            alt={product.name}
                            className="product-image"
                        />
                        <div className="product-details">
                            <h3>{product.name}</h3>
                            <p className="price">Price: ${product.price}</p>
                            <div className="actions">
                                <button
                                    className="add-to-cart-btn"
                                    onClick={() => {
                                        addToCart(product);
                                        /*navigate('/cart');*/
```

```
                        }}
                    >
                        Add to Cart
                    </button>
                    <button
                        className="buy-now-btn"
                        onClick={() => {
                            addToCart(product);
                            navigate('/checkout'); // Navigates to the checkout page
                        }}
                    >
                        Buy Now
                    </button>
                    <span
                        className="wishlist-icon"
                        onClick={() =>
                            console.log(`Added to wishlist: ${product.name}`)
                        }
                    >
                    </span>
                </div>
            </div>
        </div>
    ))
) : (
    !loading && <p>No products found.</p>
)}
</div>
</div>
);
}
export default ElectronicsPage;
```

Beauty.js

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import './ElectronicPage.css';
function Beauty({ addToCart }) {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  useEffect(() => {
    fetch('http://localhost:5000/api/cosmetics')
      .then((response) => {
        console.log('Fetch response:', response); // Debug response
        if (!response.ok) {
          throw new Error('Error fetching products');
        }
        return response.json();
      })
      .then((data) => {
        console.log('Fetched products:', data); // Debug data
        setProducts(data);
        setLoading(false);
      })
      .catch((error) => {
        console.error('Error fetching products:', error);
        setError('Error fetching products.');
        setLoading(false);
      });
  }, []);
  return (
    <div className="beauty-page">
      <h1>Beauty Products</h1>
      {loading && <p>Loading products...</p>}
```

```
{error && <p>{error}</p>}
<div className="product-list">
    {products.length > 0 ? (
        products.map((product, index) => (
            <div key={index} className="product-card">
                <img
                    src={product.imageUrl}
                    alt={product.name}
                    className="product-image"
                />
                <div className="product-details">
                    <h3>{product.name}</h3>
                    <p className="price">Price: ${product.price}</p>
                    <div className="actions">
                        <button
                            className="add-to-cart-btn"
                            onClick={() => {
                                addToCart(product);
                                /*navigate('/cart');*/
                            }}
                        >
                            Add to Cart
                        </button>
                        <button
                            className="buy-now-btn"
                            onClick={() => {
                                addToCart(product);
                                navigate('/checkout'); // Navigates to the checkout page
                            }}
                        >
                            Buy Now
                        </button>
                        <span
```

```
                        className="wishlist-icon"
                        onClick={() =>
                            console.log(`Added to wishlist: ${product.name}`)
                        }
                      >
                      </span>
                    </div>
                  </div>
                </div>
              ))
            ) : (
              !loading && <p>No products found.</p>
            )}
          </div>
        </div>
    );
}
export default Beauty;
```

Shoppingpage.js

```
import React from 'react';
import ProductList from './ProductList';
function Shoppingpage({ addToCart }) {
    return (
        <div>
            <ProductList addToCart={addToCart} />
        </div>
    );
}
export default Shoppingpage;
```

Login.js

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
```

```
import './Modal.css';
function Login() {
    const navigate = useNavigate();
    const [formData, setFormData] = useState({ email: '', password: '' });
    const [errorMessage, setErrorMessage] = useState('');
    const [validationErrors, setValidationErrors] = useState({ email: '', password: '' });
    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData({ ...formData, [name]: value });
        setValidationErrors({ ...validationErrors, [name]: '' }); // Clear field-specific errors
    };
    const validateForm = () => {
        let isValid = true;
        const errors = {};
        // Email validation
        if (!formData.email) {
            errors.email = 'Email is required.';
            isValid = false;
        } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
            errors.email = 'Please enter a valid email address.';
            isValid = false;
        }
        // Password validation
        if (!formData.password) {
            errors.password = 'Password is required.';
            isValid = false;
        }
        setValidationErrors(errors);
        return isValid;
    };
    const handleLogin = async () => {
        if (!validateForm()) return;
        try {
```

```
      const response = await axios.post('http://localhost:5000/login', formData);
      alert(response.data.message);
      if (response.status === 200) {
        // Store the username in localStorage or sessionStorage
        localStorage.setItem('username', response.data.user.name); // or sessionStorage
        navigate('/shopping'); // Navigate to the shopping page
      }
    } catch (err) {
      console.error('Error during login:', err);
      setErrorMessage(err.response?.data?.message || 'Failed to log in. Please try again.');
    }
  };
  return (
    <div className="signup-page">
      <div className="signup-form">
        <h2>Login</h2>
        {errorMessage && <p style={{ color: 'red' }}>{errorMessage}</p>}
        <label>Enter your email</label>
        <input
          type="email"
          name="email"
          placeholder="abc@gmail.com"
          value={formData.email}
          onChange={handleChange}
        />
                                {validationErrors.email && <p className="error-
text">{validationErrors.email}</p>}
        <label>Password</label>
        <input
          type="password"
          name="password"
          placeholder="Enter your password"
          value={formData.password}
```

```
                onChange={handleChange}
              />
 {validationErrors.password && <p className="error-text">{validationErrors.password}</p>}
              <button onClick={handleLogin} className="continue-button">
                Continue
              </button>
              <button onClick={() => navigate('/')} className="close-button">
                Close
              </button>
            </div>
          </div>
    );
}
export default Login;
```

Signup.js

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import './Modal.css';
function Signup() {
    const navigate = useNavigate();
    const [formData, setFormData] = useState({
        name: '',
        mobileNumber: '',
        countryCode: 'IN +91',
        email: '',
        password: ''
    });
    const [errorMessage, setErrorMessage] = useState('');
    const [validationErrors, setValidationErrors] = useState({});

    const handleChange = (e) => {
```

```
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
    setValidationErrors({ ...validationErrors, [name]: '' }); // Clear field-specific errors
  };
  const validateForm = () => {
    let isValid = true;
    const errors = {};
    // Name validation
    if (!formData.name) {
      errors.name = 'Name is required.';
      isValid = false;
    }
    // Mobile number validation
    if (!formData.mobileNumber) {
      errors.mobileNumber = 'Mobile number is required.';
      isValid = false;
    } else if (!/^\d+$/.test(formData.mobileNumber)) {
      errors.mobileNumber = 'Mobile number must be numeric.';
      isValid = false;
    } else if (formData.mobileNumber.length <10) {
      errors.mobileNumber = 'Mobile number must be 10 digits.';
      isValid = false;
    }
    // Email validation
    if (!formData.email) {
      errors.email = 'Email is required.';
      isValid = false;
    } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
      errors.email = 'Please enter a valid email address.';
      isValid = false;
    }
    // Password validation
    if (!formData.password) {
```

```
        errors.password = 'Password is required.';
        isValid = false;
    } else if (formData.password.length < 6) {
        errors.password = 'Password must be at least 6 characters.';
        isValid = false;
    }
    setValidationErrors(errors);
    return isValid;
};
const handleSignup = async () => {
    if (!validateForm()) return;
    try {
        const response = await axios.post('http://localhost:5000/signup', formData);
        alert(response.data.message);
        if (response.status === 201) {
            navigate('/login'); // Navigate to login page after signup
        }
    } catch (err) {
        console.error('Error during signup:', err);
        setErrorMessage(err.response?.data?.message || 'Failed to sign up. Please try again.');
    }
};
return (
    <div className="signup-page">
        <div className="signup-form">
            <h2>Signup</h2>
            {errorMessage && <p style={{ color: 'red' }}>{errorMessage}</p>}
            <label>Your name</label>
            <input
                type="text"
                name="name"
                placeholder="Enter your name"
                value={formData.name}
```

```
            onChange={handleChange}
          />
                                {validationErrors.name && <p className="error-
text">{validationErrors.name}</p>}
          <label>Mobile number</label>
          <div className="mobile-input">
            <select
              name="countryCode"
              value={formData.countryCode}
              onChange={handleChange}
            >
              <option value="IN +91">IN +91</option>
              <option value="US +1">US +1</option>
              <option value="UK +44">UK +44</option>
            </select>
            <input
              type="text"
              name="mobileNumber"
              placeholder="Mobile number"
              value={formData.mobileNumber}
              onChange={handleChange}
            />
          </div>
                          {validationErrors.mobileNumber && <p className="error-
text">{validationErrors.mobileNumber}</p>}
          <label>Email</label>
          <input
            type="email"
            name="email"
            placeholder="abc@gmail.com"
            value={formData.email}
            onChange={handleChange}
          />
```

```
                                {validationErrors.email && <p className="error-
text">{validationErrors.email}</p>}
            <label>Password</label>
            <input
              type="password"
              name="password"
              placeholder="Password"
              value={formData.password}
              onChange={handleChange}
            />
                                {validationErrors.password && <p className="error-
text">{validationErrors.password}</p>}
            <button className="continue-button" onClick={handleSignup}>
              Continue
            </button>
            <button onClick={() => navigate('/')} className="close-button">
              Close
            </button>
          </div>
        </div>
    );
}
export default Signup;
```

SellerLogin.js

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import './Modal.css';
const SellerLogin = () => {
  const [formData, setFormData] = useState({ username: '', password: '' });
  const [errors, setErrors] = useState({ username: '', password: '' });
  const [message, setMessage] = useState('');
  const navigate = useNavigate();
```

```
const handleInputChange = (e) => {
  const { name, value } = e.target;
  setFormData({ ...formData, [name]: value });
  setErrors({ ...errors, [name]: '' });
};
const validateForm = () => {
  let valid = true;
  const newErrors = { username: '', password: '' };
  if (!formData.username) {
    newErrors.username = 'Username is required.';
    valid = false;
  }
  if (!formData.password) {
    newErrors.password = 'Password is required.';
    valid = false;
  } else if (formData.password.length < 6) {
    newErrors.password = 'Password must be at least 6 characters long.';
    valid = false;
  }
  setErrors(newErrors);
  return valid;
};
const handleSubmit = async (e) => {
  e.preventDefault();
  if (validateForm()) {
    try {
      const response = await axios.post('http://localhost:5000/api/sellers/login', formData);
      setMessage('Login successful');
      navigate('/sellproduct', { state: { username: formData.username } });
    } catch (error) {
      console.error('Login error:', error);
      setMessage('Login failed: Invalid credentials');
    }
```

```
    }
  };
  return (
    <div className="modal-page">
      <form className="modal-form" onSubmit={handleSubmit}>
        <h2>Seller Login</h2>
        <label>
          Username:
                        <input     type="text"     name="username"     value={formData.username}
onChange={handleInputChange} />
          {errors.username && <p className="error-text">{errors.username}</p>}
        </label>
        <label>
          Password:
                    <input    type="password"    name="password"    value={formData.password}
onChange={handleInputChange} />
          {errors.password && <p className="error-text">{errors.password}</p>}
        </label>
        <button type="submit" className="continue-button">Login</button>
        {message && <p className="message-text">{message}</p>}
      </form>
    </div>
  );
};
export default SellerLogin;
```

SellProduct.js

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useLocation } from 'react-router-dom';
import './SellProduct.css';
const SellProduct = () => {
  const { state } = useLocation(); // Access username passed from previous page
  const [sellerId, setSellerId] = useState(null); // State to store the sellerId after login
```

```
const [credentials, setCredentials] = useState({
  username: '',
  password: '',
});
const [product, setProduct] = useState({
  name: '',
  price: '',
  category: '',
  offer: '',
  image: null,
});
// Sync sellerId from localStorage
useEffect(() => {
  const storedSellerId = localStorage.getItem('sellerId');
  if (storedSellerId) {
    setSellerId(storedSellerId);
  }
}, []);
// Handle input changes for product
const handleChange = (e) => {
  const { name, value } = e.target;
  setProduct({ ...product, [name]: value });
};
// Handle file input changes for product image
const handleFileChange = (e) => {
  setProduct({ ...product, image: e.target.files[0] });
};
// Handle login
const handleLogin = async () => {
  try {
    const response = await axios.post('http://localhost:5000/api/sellers/login', {
      username: credentials.username,
      password: credentials.password,
```

```
    });
    if (response.data.sellerId) {
      localStorage.setItem('sellerId', response.data.sellerId); // Store sellerId in localStorage
      setSellerId(response.data.sellerId); // Update state
      alert('Login successful');
    }
  } catch (error) {
    alert('Invalid credentials');
  }
};
// Handle adding product
const handleAddProduct = async (e) => {
  e.preventDefault();
  if (!sellerId) {
    alert('Please login to add a product');
    return;
  }
  const formData = new FormData();
  formData.append('name', product.name);
  formData.append('price', product.price);
  formData.append('category', product.category);
  formData.append('offer', product.offer);
  formData.append('image', product.image);
  formData.append('sellerId', sellerId); // Add sellerId to form data
  try {
    await axios.post('http://localhost:5000/api/sell-products/add', formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
      },
    });
    alert('Product added successfully');
  } catch (error) {
    console.error('Error adding product:', error);
```

```
    alert('Error adding product');
  }
};
return (
  <div className="sell-product-form-container">
    {sellerId ? (
      <>
        <h2>Add a Product</h2>
        <form onSubmit={handleAddProduct} className="sell-product-form">
          <label>
            Product Name:
            <input
              type="text"
              name="name"
              value={product.name}
              onChange={handleChange}
              placeholder="Enter product name"
              required
            />
          </label>
          <label>
            Image:
            <input
              type="file"
              name="image"
              accept="image/*"
              onChange={handleFileChange}
              required
            />
          </label>
          <label>
            Price:
            <input
```

```
            type="number"
            name="price"
            value={product.price}
            onChange={handleChange}
            placeholder="Enter price"
            required
          />
        </label>
        <label>
          Category:
          <input
            type="text"
            name="category"
            value={product.category}
            onChange={handleChange}
            placeholder="Enter category"
            required
          />
        </label>
        <label>
          Offer:
          <input
            type="text"
            name="offer"
            value={product.offer}
            onChange={handleChange}
            placeholder="Enter offer (optional)"
          />
        </label>
        <button type="submit">Add Product</button>
      </form>
    </>
  ) : (
```

```
        <div className="seller-login">
          <h3>Seller Login</h3>
          <label>
            Username:
            <input
              type="text"
              placeholder="Enter your username"
              value={credentials.username}
              onChange={(e) => setCredentials({ ...credentials, username: e.target.value })}
              required
            />
          </label>
          <label>
            Password:
            <input
              type="password"
              placeholder="Enter your password"
              value={credentials.password}
              onChange={(e) => setCredentials({ ...credentials, password: e.target.value })}
              required
            />
          </label>
          <button onClick={handleLogin}>Login</button>
        </div>
      )}
    </div>
  );
};
export default SellProduct;
```

index.js

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './components/App.css';
import App from './components/App';
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

BACKEND:

Server.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const multer = require('multer');
const path = require('path');
const app = express();
// Middleware
app.use(cors());
app.use(bodyParser.json());
// Serve static files from 'uploads' folder
app.use('/uploads', express.static('uploads'));
// Multer Storage Setup for File Uploads
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/'); // Directory to save uploaded files
    },
    filename: (req, file, cb) => {
        cb(null, Date.now() + '-' + file.originalname); // Unique filenames
```

```
  },
});
const upload = multer({ storage });
// MongoDB Connection
mongoose
  .connect('mongodb://localhost:27017/vijidb', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log('Connected to MongoDB'))
  .catch((err) => console.error('Error connecting to MongoDB:', err));
// Schemas
const userSchema = new mongoose.Schema({
  name: { type: String },
  mobileNumber: { type: String },
  countryCode: { type: String },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});
const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  imageUrl: { type: String, required: true },
  price: { type: Number, required: true },
  category: { type: String, required: true },
});
const beautySchema = new mongoose.Schema({
  name: { type: String, required: true },
  imageUrl: { type: String, required: true },
  price: { type: Number, required: true },
  category: { type: String, required: true },
});
const sellerSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
```

```
});
// Models
const User = mongoose.model('User', userSchema);
const Product = mongoose.model('Product', productSchema);
const Beauty = mongoose.model('Cosmetic', beautySchema);
const Seller = mongoose.model('Seller', sellerSchema);
const sellProductSchema = new mongoose.Schema({
  sellerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Seller', required: true },
  userid: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  name: { type: String, required: true },
  imageUrl: { type: String, required: true },
  price: { type: Number, required: true },
  category: { type: String, required: true },
  description: { type: String },
  discount: {
    type: Number,
    set: (value) => {
      if (typeof value === 'string' && value.includes('%')) {
        return parseFloat(value.replace('%', ''));
      }
      return value;
    },
  },
});
// SellProduct Model
const SellProduct = mongoose.model('SellProduct', sellProductSchema);
app.post('/api/sell-products/add', upload.single('image'), async (req, res) => {
  console.log('Received product data:', req.body); // Log the incoming product data
  console.log('Uploaded file:', req.file); // Log the uploaded file
  const { sellerId, name, price, category, description, offer } = req.body;
  if (!sellerId || !name || !price || !category || !req.file) {
    return res.status(400).json({ message: 'All fields including image are required' });
  }
  // Process the discount value
```

```
    let discount = offer;
    if (typeof discount === 'string' && discount.includes('%')) {
        discount = parseFloat(discount.replace('%', '')); // Remove the % and convert to number
    }
    try {
        // Store sellerId as userid in the SellProduct collection
        const newProduct = new SellProduct({
            sellerId, // Using sellerId from the form data
            userid: sellerId, // Storing sellerId as userid
            name,
            price,
            category,
            description,
            discount, // Use the processed discount value
            imageUrl: `/uploads/${req.file.filename}`,
        });
        await newProduct.save();
        // Check the category and save in the appropriate collection
        const imageUrl = `/uploads/${req.file.filename}`;
        if (category.toLowerCase() === 'electronics') {
            const electronicsProduct = new Product({ name, imageUrl, price, category });
            await electronicsProduct.save();
        } else if (category.toLowerCase() === 'cosmetics' || category.toLowerCase() === 'beauty')
{
            const beautyProduct = new Beauty({ name, imageUrl, price, category });
            await beautyProduct.save();
        }
        res.status(201).json({ message: 'Product added successfully to both collections!' });
    } catch (error) {
        console.error('Error adding product:', error);
        res.status(500).json({ message: 'Error adding product', error: error.message });
    }
});
// Fetch Sell Products for a Specific Seller
```

```
app.get('/api/sell-products/:sellerId', async (req, res) => {
    const { sellerId } = req.params;
    try {
        const products = await SellProduct.find({ sellerId });
        res.status(200).json(products);
    } catch (error) {
        res.status(500).json({ message: 'Error fetching sell products', error: error.message });
    }
});
// Seller Signup
app.post('/api/sellers/signup', async (req, res) => {
    const { username, password } = req.body;
    if (!username || !password) {
        return res.status(400).json({ message: 'Username and password are required' });
    }
    try {
        const existingSeller = await Seller.findOne({ username });
        if (existingSeller) {
            return res.status(400).json({ message: 'Username is already taken' });
        }
        const newSeller = new Seller({ username, password });
        await newSeller.save();
        res.status(201).json({ message: 'Seller signed up successfully!' });
    } catch (err) {
        res.status(500).json({ message: 'Error signing up seller', error: err.message });
    }
});
// Seller Login
app.post('/api/sellers/login', async (req, res) => {
    const { username, password } = req.body;

    if (!username || !password) {
        return res.status(400).json({ message: 'Username and password are required' });
    }
```

```
    try {
        const seller = await Seller.findOne({ username, password });
        if (seller) {
            // Return the sellerId (MongoDB's _id)
            res.status(200).json({ message: 'Login successful', sellerId: seller._id });
        } else {
            res.status(401).json({ message: 'Invalid credentials' });
        }
    } catch (error) {
        res.status(500).json({ message: 'Error logging in', error: error.message });
    }
});
// User Signup
app.post('/signup', async (req, res) => {
    const { email, password, name, mobileNumber, countryCode } = req.body;
    if (!email || !password) {
        return res.status(400).json({ message: 'Email and password are required' });
    }
    try {
        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ message: 'Email is already registered' });
        }
        const newUser = new User({ email, password, name, mobileNumber, countryCode });
        await newUser.save();
        res.status(201).json({ message: 'User signed up successfully!' });
    } catch (err) {
        res.status(500).json({ message: 'Error signing up user', error: err.message });
    }
});
// User Login
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    if (!email || !password) {
```

```
        return res.status(400).json({ message: 'Email and password are required' });
    }
    try {
        const user = await User.findOne({ email, password });
        if (user) {
            res.status(200).json({ message: 'Login successful', user });
        } else {
            res.status(401).json({ message: 'Invalid credentials' });
        }
    } catch (error) {
        res.status(500).json({ message: 'Error logging in', error: error.message });
    }
});
// Fetch Electronics Products
app.get('/api/products', async (req, res) => {
    try {
        const electronicsProducts = await Product.find();
        res.status(200).json(electronicsProducts);
    } catch (error) {
        res.status(500).json({ message: 'Error fetching products', error: error.message });
    }
});
// Fetch Cosmetics Products
app.get('/api/cosmetics', async (req, res) => {
    try {
        const cosmeticsProducts = await Beauty.find();
        res.status(200).json(cosmeticsProducts);
    } catch (error) {
         res.status(500).json({ message: 'Error fetching cosmetics products', error: error.message
});
    }
});
// Add Product with Image Upload
```

```
app.post('/api/products/add', upload.single('image'), async (req, res) => {
    const { name, price, category } = req.body;

    if (!name || !price || !category || !req.file) {
        return res.status(400).json({ message: 'All fields including image are required' });
    }
    const imageUrl = `/uploads/${req.file.filename}`;
    try {
        const newProduct = new Product({ name, imageUrl, price, category });
        await newProduct.save();
        res.status(201).json({ message: 'Product added successfully!', product: newProduct });
    } catch (error) {
        res.status(500).json({ message: 'Error adding product', error: error.message });
    }
});
// Server Setup
const PORT = 5000;
app.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
});
```

11.2 **INSTALLATION AND USAGE INSTRUCTIONS:**

1. Prerequisites:

- Node.js (latest LTS version)
- MongoDB Compass (for database management)
- Code editor (e.g., Visual Studio Code)
- Web browser

2. Installation Steps:

Step 1: Create the Backend Directory

1. Open a terminal and create the backend folder:

   ```
   mkdir server
   cd server
   ```

2. Initialize the backend project:

   ```
   npm init -y
   ```

3. Install necessary backend packages:

   ```
   npm install express mongoose bcrypt jsonwebtoken cors dotenv
   ```

4. Create a upload file structure in the myfirstreactapp directory:

   ```
   /myfirstreactapp
     |-- node_modules
     |-- public
     |-- src
     |-- uploads
     |-- .env
     |-- index.js
   ```

5. Configure the .env file:

   ```
   PORT=5000
   MONGO_URI=<your-mongo-db-connection-string>
   JWT_SECRET=<your-jwt-secret>
   ```

6. Start the backend server:

   ```
   node server.js
   ```

Step 2: Set Up the Frontend (Client Directory)

1. Open a new terminal window and create the frontend directory:

   mkdir myfirstreactapp

   cd myfirstreactapp
2. Initialize a React project:

   npx create-react-app
3. Install additional frontend packages:

   npm install axios react-router-dom
4. Start the frontend development server:

   npm start
5. Access the application at http://localhost:3000 in your web browser.

3. Database Configuration:

1. Open MongoDB Compass.
2. Connect to your MongoDB server using the URI specified in the .env file.
3. Create a new database named vijidb with the following collections:
   - users (for storing user details)
   - sellers (for storing sellers information)
   - products (for storing electronics products information)
   - sellproducts (for storing the seller products)
   - cosmetics (for storing beauty products details)

4. Usage Instructions:

1. **Register & Login:**
   - Access the application in your web browser.
   - Register a new account and log in with your credentials.
2. **Search & Explore Products:**
   - Use filters like category name to search for products.
   - View detailed information about the products.
3. **Manage Cart:**
   - Save your favourite products using the "Add to Cart" button.
   - Access and manage saved products from your profile page.