Deliverable 4 – Project Phase 2
CSCE 5430 (Fall 2024)

Group Name: **Group-3**

Group Members:

- Vaishnavi Shastrula (Project Management Lead)
- Karunya Mekala (Requirements Lead)
- Teja Nagendra Sirigineedi (Configuration Management Lead & Demo and Presentation Lead)
- Sai Sowjanya Edupuganti (Design Lead)
- Katikala Damodar Reddy (Implementation Lead for Frontend)
- Gayathri Vutla (Implementation Lead for Backend)
- Chopra Sai Arani (Testing Lead)
- Preethi Medipelli (Documentation Lead)
- Shraehitha Reddy Banda (System Administrator Lead)

Phase 2 Requirements

The development of Phase 2 in the Toy Station project focuses on enhancing user experience and operational functionality by adding advanced features. The scope remains largely the same as outlined in Deliverable 2, but this report provides a more detailed description of each requirement. We have also evaluated the feasibility of certain features based on technical constraints and project resources.

1. Recommendation Engine (High Priority)

- **Purpose:** The recommendation engine is essential for enhancing user engagement by suggesting products based on user behavior. This feature improves the usability of the platform, guiding users to discover products that align with their preferences, thereby promoting cross-selling.
- **Key Features:**

    - A "Recommended Shelf" displays personalized product suggestions alongside each viewed product.
    - Related items, such as similar or complementary toys, are displayed below each product.
    - Recommendations are achieved through a mapping function where each product ID points to related product IDs.
- **Implementation Details:**
    - For instance, if a user views a "Toy Car," suggestions like "Toy Train" or "Toy Plane" appear beneath it, encouraging exploration. This feature is achieved through a simple `recommendations` mapping to enhance browsing without complex algorithms.
- Impact: Helps users find items faster, promotes cross-selling, and makes the shopping experience more engaging and intuitive.

2. Payment Gateway Integration (High Priority)

- **Purpose:** Integrating a secure payment gateway is critical for completing transactions. This feature allows users to pay for their purchases safely, ensuring data protection and compliance with financial regulations.
- **Key Features:**
    - Secure payment methods with third-party integrations (e.g., Stripe, Razorpay).
    - Support for multiple payment options, providing flexibility for users.
- **Reason for Change:**
    - **Security Compliance:** The need for PCI DSS compliance and handling sensitive financial data poses technical and regulatory challenges.
    - **Resource Constraints:** Integrating and testing a payment gateway within the given timeframe and budget requires specialized expertise and robust infrastructure.
- **Impact:** Although essential, the integration may be postponed due to the significant technical, security, and regulatory requirements involved. Alternative methods, such as mock transactions, could be considered as placeholders.

3. Order Tracking, Returns, and Refunds (High Priority)

- **Purpose:** These features enhance customer service by allowing users to track their orders, request returns, and manage refunds, thus building trust and ensuring customer satisfaction.
- **Key Features:**
    - **Order Tracking:** Provides real-time status updates from processing to delivery.
    - **Returns and Refunds**: Users can initiate returns and request refunds directly through their order history.
- **Reason for Change:**
    - **Technical Limitations:** Real-time order tracking requires integration with shipping APIs, which could be challenging given the current back-end infrastructure.
    - **Complexity in State Management:** Ensuring accurate and updated order statuses demands complex front-end state management and server coordination, which may require additional resources.
- **Impact:** Order tracking might be deprioritized until a robust API integration solution is feasible. However, a basic returns and refund feature has been implemented, allowing users to initiate return requests, enhancing post-purchase satisfaction.

4. Customer Support System (Medium Priority)

- **Purpose:** The customer support system assists users in resolving queries and issues, offering a more accessible and reliable shopping experience.
- **Key Features:**
    - **FAQ Section:** Provides answers to frequently asked questions regarding order tracking, returns, and other common inquiries.
    - **Support Request Form:** Users can submit detailed support requests, providing necessary information for the support team.
- **Implementation Details:**
    - The FAQ is designed in an accordion format, allowing users to expand and collapse responses for a streamlined view.
    - Support requests are submitted via a form, ensuring quick access to customer service when FAQs do not suffice.

- **Impact:** This dual-function support page addresses user queries quickly and offers a formal channel for specific issues, improving customer satisfaction.

5. Product Reviews and Ratings (Medium Priority)

- **Purpose:** The reviews and ratings feature allows customers to share their experiences, aiding others in making informed purchasing decisions.
- **Key Features:**
    - Average product ratings are displayed on product pages.
    - A "Write a Review" button opens a modal where users can submit their feedback, rating products from 1 to 5 stars and leaving comments.
- **Implementation Details:**
    - Submitted reviews update the product's average rating, fostering transparency and a sense of community interaction on the platform.
- **Impact:** This feature enhances decision-making for buyers and builds trust within the user community by showcasing honest reviews from other customers.

Explanation of Adjusted Features

As per technical requirements, the following features have been reprioritized for future phases. This adjustment can be helpful to focus on core functionalities while maintaining quality.

1. Chat Bot Implementation (Initially Planned for Requirement 4)

- **Reason for Change:** Live chat requires a dedicated back-end service for real-time communication, user authentication, and a responsive front-end interface, which is resource-intensive.
- **Decision:** We are postponing the chat bot feature to allow for further research into real-time data handling and backend requirements. For this phase, the support request form will serve as the primary user support tool. The chat bot will be revisited in a later phase with a more detailed implementation plan.

2. Order Tracking (Initially Part of Requirement 3)

- **Reason for Change:** Implementing a full-fledged order tracking system involves integrating with external shipping APIs and managing complex state transitions on the front end, which is challenging given the current architecture.
- **Decision:** For this phase, we will implement basic order status updates as an interim solution. Advanced order tracking capabilities will be introduced in future phases, following research on third-party API integration and state management.

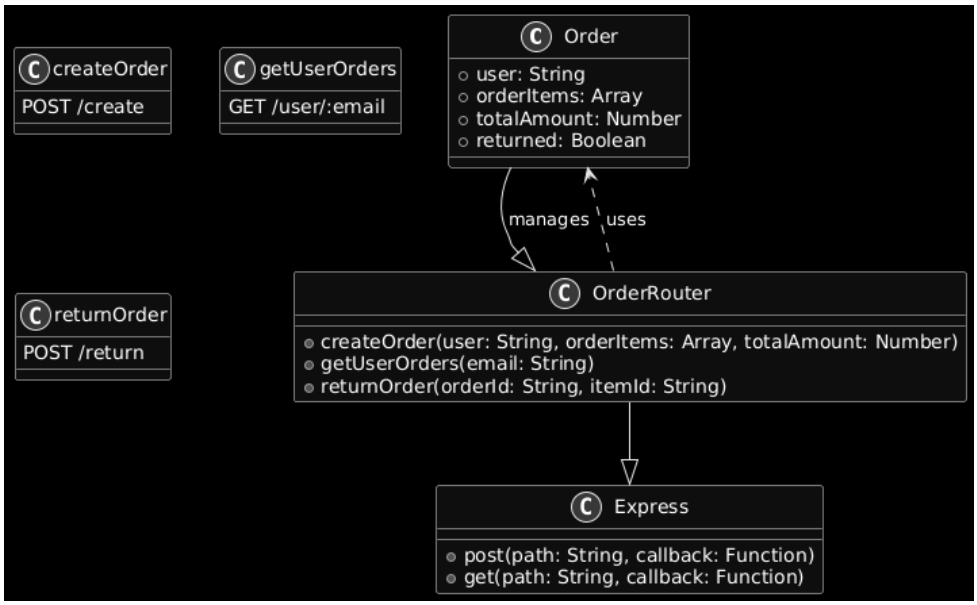
3. Mock Payment Integration (Initially Part of Requirement 2)

- **Reason for Change:** Full payment gateway integration demands compliance with security standards and rigorous testing to handle sensitive data securely, which may exceed current resources.
- **Decision:** Instead of a full payment gateway, we will integrate a mock payment system for the current phase. This approach provides basic checkout functionality while allowing us to prioritize data security. A full payment integration will be revisited once sufficient research into security protocols is completed.

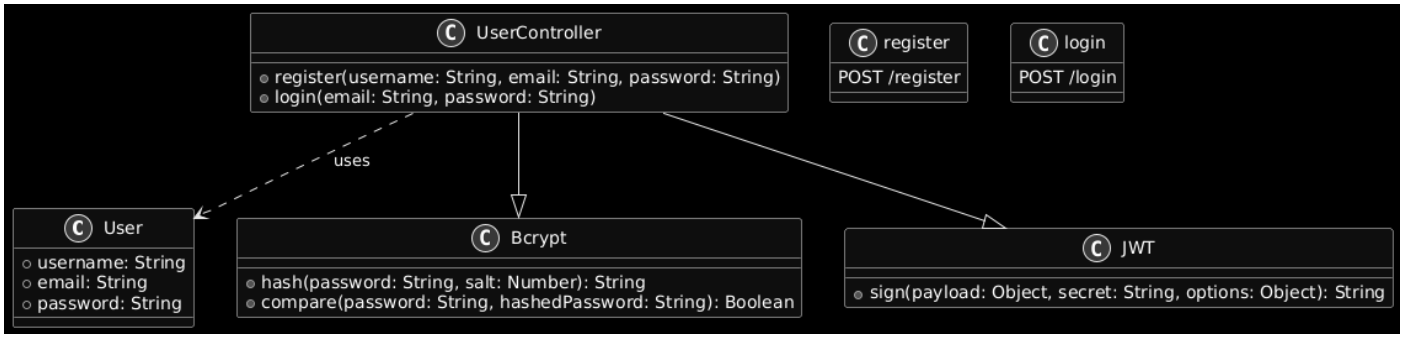
UML Design

Class Diagram

1. **Order:**

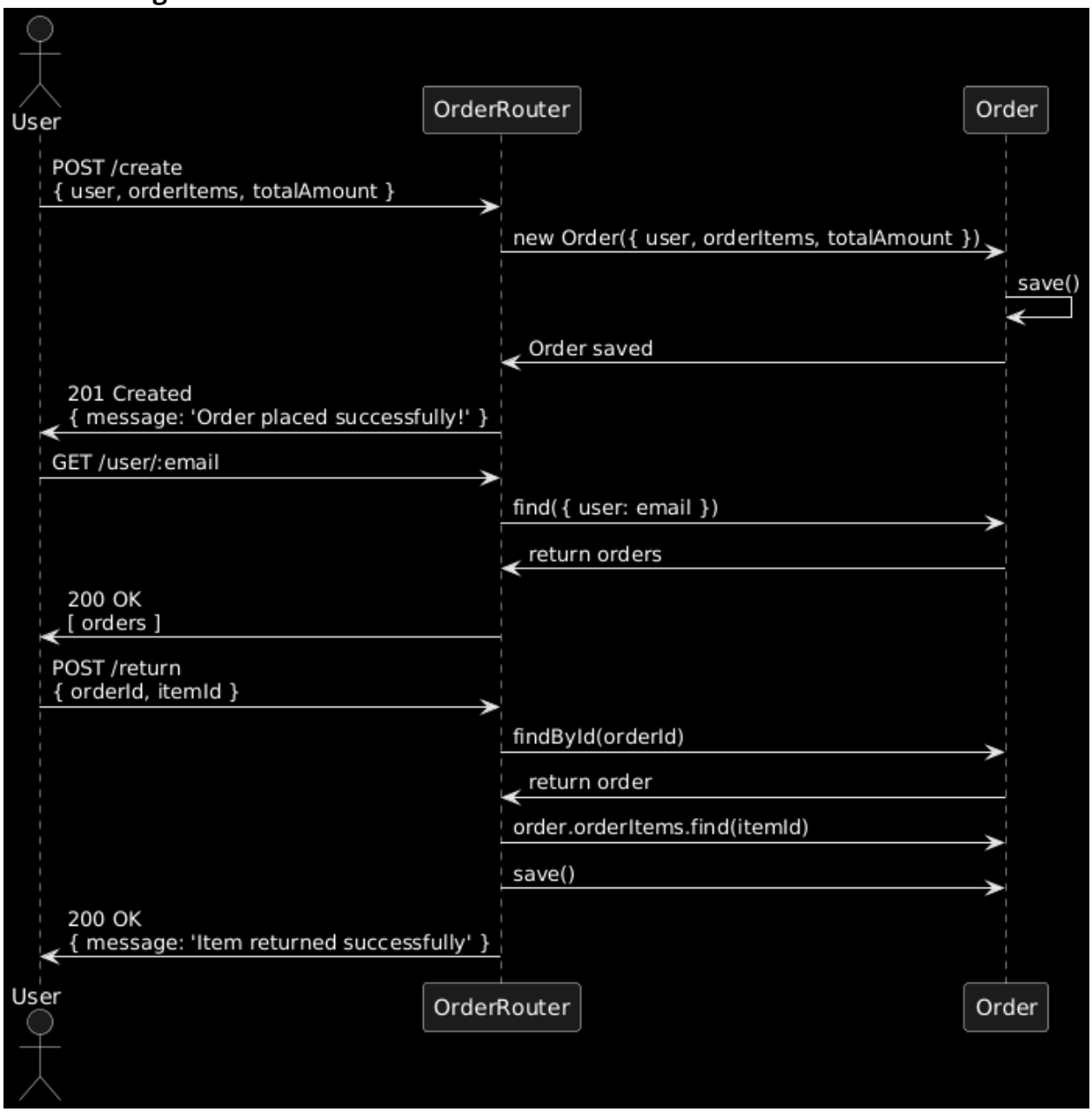


2. **User:**

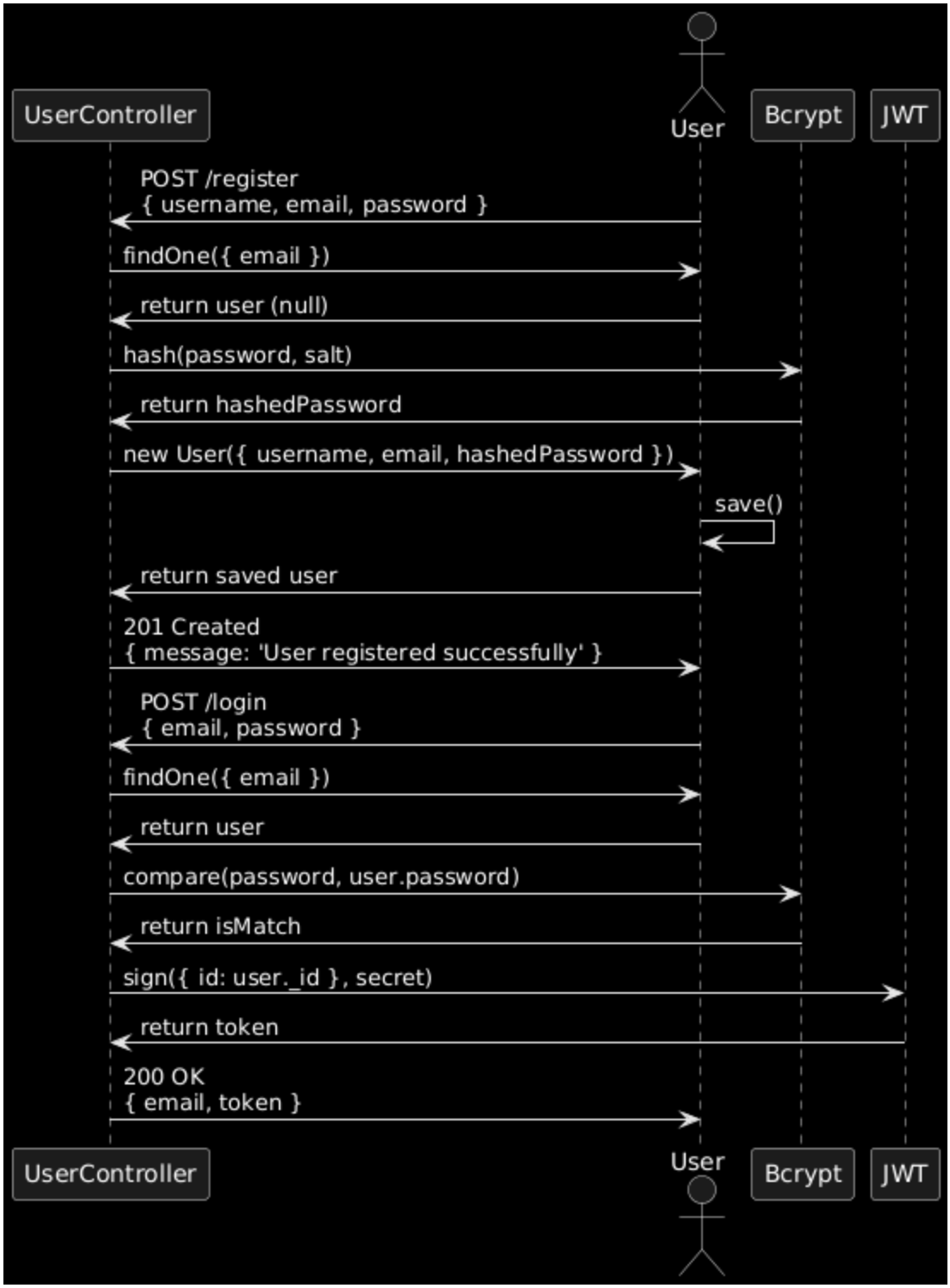


Sequence Diagram

1. **Order Management:**

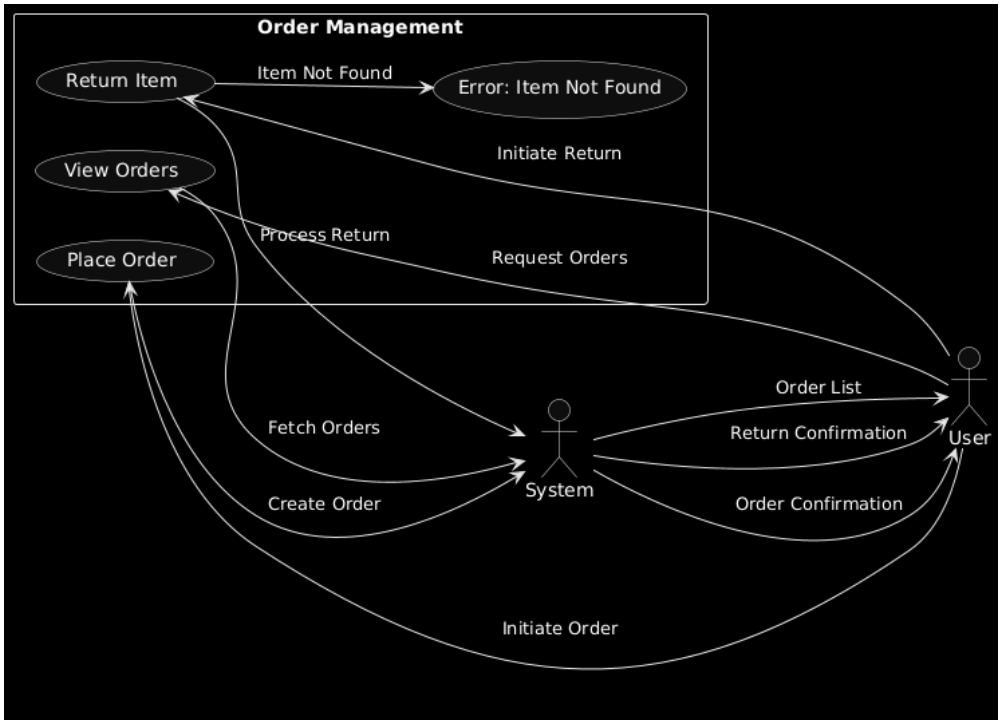


2. **User Management:**

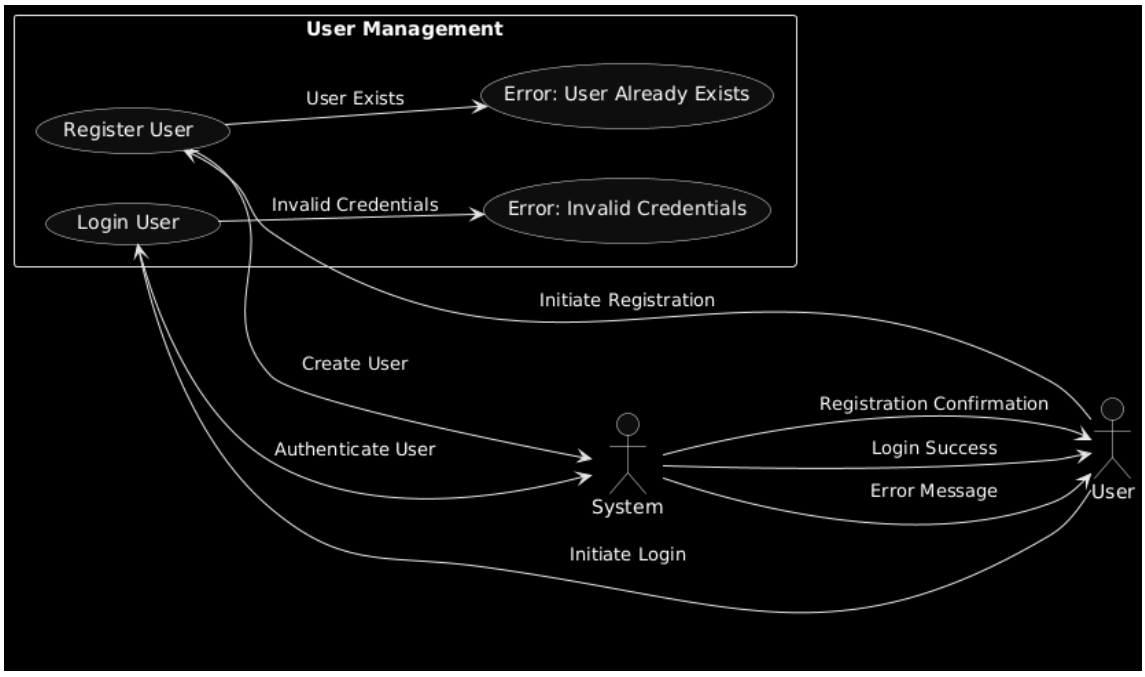


Use Case Diagram

1. **Order Management:**



2. **User Management**



Unit Test Cases for Order Management

Test Case 1: Create Order Successfully

- Verifies the ability to create a new order successfully.
- **Prerequisites:**
  - User is logged in.
  - At least one item is available in stock.
- **Inputs**:
  - `user`: "testuser@example.com"
  - `orderItems`: [{ id: 1, name: "Toy Car", quantity: 2, total: 30 }]
  - `totalAmount`: 30
- **Expected Output**: Status 201 with message "Order placed successfully!"

Test Case 2: Create Order with Missing Fields

- Tests the response when creating an order without required fields.
- **Prerequisites:**
  - User is logged in.
  - Item is added to the cart.
  - User has not provided all required fields.
- **Inputs**:
  - `orderItems`: [] (empty array)
  - `totalAmount`: 0
- **Expected Output**: Status 400 with message "Failed to place order."

Test Case 3: Fetch User Orders Successfully

- Checks if orders can be successfully fetched for a specific user.
- **Prerequisites:**
  - User is logged in.
  - User has placed at least one previous order.

- **Inputs**:
    - `userEmail`: "testuser@example.com"
- **Expected Output**: Status 200 with an array of orders for the user.

Test Case 4: Fetch Orders for Non-Existing User

- Validates the response when fetching orders for a non-existing user.
- **Prerequisites:**
    - Attempt to fetch orders using a user ID that does not exist in the system.
- **Inputs**:
    - `userEmail`: "nonexistentuser@example.com"
- **Expected Output**: Status 404 with message "No orders found."

Test Case 5: Return Item Successfully

- Ensures that an item can be returned successfully from an order.
- **Prerequisites:**
    - User is logged in.
    - User has an order with an item eligible for return.
- **Inputs**:
    - `orderId`: "validOrderId"
    - `itemId`: "validItemId"
- **Expected Output**: Status 200 with message "Item returned successfully."

Test Case 6: Return Non-Existent Item

- Tests the response when attempting to return a non-existent item in the order.
- **Prerequisites:**
    - User is logged in.
    - User attempts to return a product which is not part of any order in their order history.
- **Inputs**:
    - `orderId`: "validOrderId"
    - `itemId`: "invalidItemId"

- **Expected Output**: Status 404 with message "Item not found in order."

Unit Test Cases for User Management

Test Case 1: Register User Successfully

- Validates successful registration of a new user.
- **Prerequisites:**
  - None (User is not logged in).
- **Inputs**:
  - `username`: "testuser"
  - `email`: "testuser@example.com"
  - `password`: "password123"
- **Expected Output**: Status 201 with message "User registered successfully."

Test Case 2: Register User with Existing Email

- Tests the system's response when attempting to register an already existing email.
- **Prerequisites:**
  - An account already exists in the system with the same credentials as the email address.
- **Inputs**:
  - `username`: "testuser"
  - `email`: "existinguser@example.com"
  - `password`: "password123"
- **Expected Output**: Status 400 with message "User already exists."

Test Case 3: Login User Successfully

- Checks successful login for a registered user.
- **Prerequisites:**
  - User is already registered with valid credentials.
- **Inputs**:
  - `email`: "testuser@example.com"

    - `password`: "password123"
- **Expected Output**: Status 200 with email and a valid JWT token.

Test Case 4: Login User with Invalid Credentials

- Verifies the response when a user attempts to log in with incorrect credentials.
- **Prerequisites:**
    - User account exists in the system.
    - Invalid credentials (incorrect password or email).
- **Inputs**:
    - `email`: "testuser@example.com"
    - `password`: "wrongpassword"
- **Expected Output**: Status 400 with message "Invalid credentials."

Test Case 5: Login Non-Existing User

- Ensures the system correctly handles login attempts from non-existing users.
- **Prerequisites:**
    - Attempt to log in with an email address that does not exist in the system.
- **Inputs**:
    - `email`: "nonexistinguser@example.com"
    - `password`: "password123"
- **Expected Output**: Status 400 with message "Invalid credentials."

Instructions for Compiling and Running the Program

Compiling and Running the Toy Station Program

The following instructions guide you through compiling and running the Toy Station program, as well as executing test cases.

1. Compiling the Program

- **Frontend (React.js):**

    ○ No explicit compilation is required. The frontend uses React.js, which bundles everything when you run npm start.Ensure you have Node.js installed, and run the following commands in the client folder:

    npm install

    npm start

- **Backend (Node.js and Express.js):**

    ○ No explicit compilation is needed. The backend is built using Node.js and Express.js, and it runs directly from the source.Install the necessary dependencies and start the server by running these commands in the server folder:

    npm install

    npm start

2. Running the Program

- **Frontend:**

    ○ After running npm start in the client folder, the frontend will automatically launch at

    http://localhost:3000.

- **Backend:**

○ Once you run npm start in the server folder, the backend will start on

http://localhost:5000.

○ Make sure MongoDB is running on your local machine or cloud to store data for users,

products, and orders.

3. Running Test Cases

● Unit Tests:

○ Unit tests for both frontend and backend can be run using the following commands:

npm test

○ For the backend (Node.js): npm run test

● Test Suite:

The test suite includes tests for:

■ User registration and login

■ Product addition to the cart

■ Checkout process

■ API responses for different routes

Ensure the servers are running and MongoDB is connected before running tests.

4. Test Case Results

● Successful test cases will show PASS in the terminal.

● If any test fails, the error will be displayed in the console, showing the cause of failure.

User Manual for Toy Station

Toy Station provides fun and easy shopping for toys to meet the wants of children, parents and toy enthusiasts that can also reduce the time and effort in the shopping process.

This user manual will guide you through using the application, covering registration, logging in, placing orders, viewing orders, returning items, and leaving product reviews.

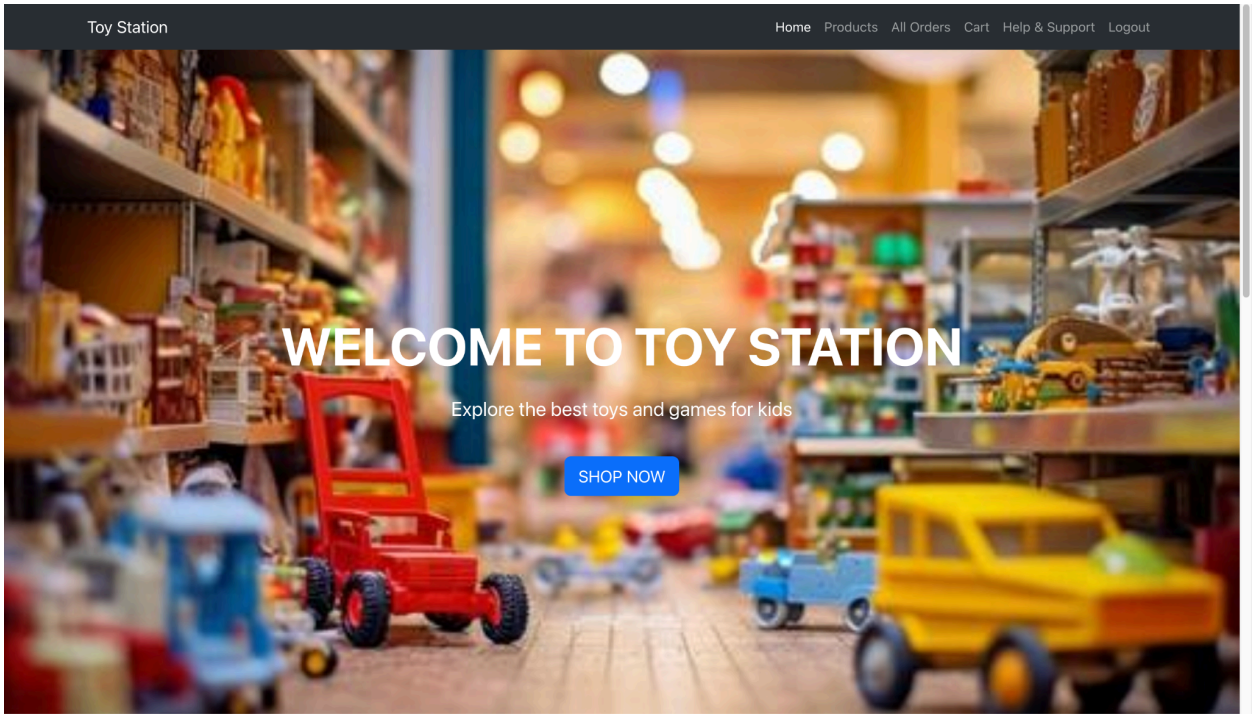
Table of Contents

Getting Started

To begin using the application, ensure that you have the necessary access. The application is accessible through a web browser.

Home Page

Once you open the application, you'll land on the home page, where you can browse available products.



User Registration

1. **Navigate to the Registration Page**: Click on the "Register" link located at the top right corner of the homepage.

2. **Fill in the Registration Form**:

    - Enter your **username**.
    - Provide your **email address**.
    - Create a **password**.
3. **Submit the Form**: Click the "Register" button to create your account.

4. **Confirmation**: Upon successful registration, a message will indicate that you have registered successfully.

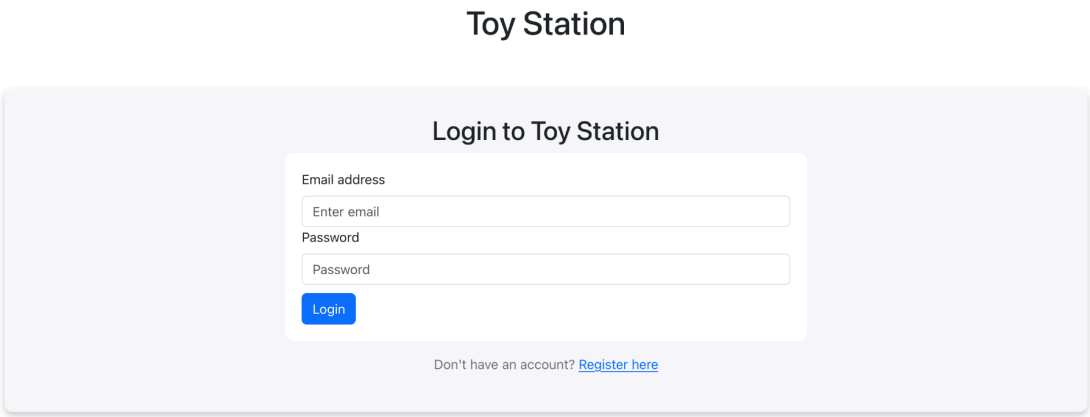
Logging In

1. **Navigate to the Login Page**: Click on the "Login" link at the top right corner of the homepage.

2. **Fill in the Login Form**:

    - Enter your registered **email address**.
    - Provide your **password**.
3. **Submit the Form**: Click the "Login" button to access your account.

4. **Successful Login**: Upon successful login, you will be redirected to the home page, where you can view products.



Placing an Order

1. **Browse Products**: From the home page, scroll through the list of available products.

2. **Select a Product**: Click on a product card to view its details, including description and price.

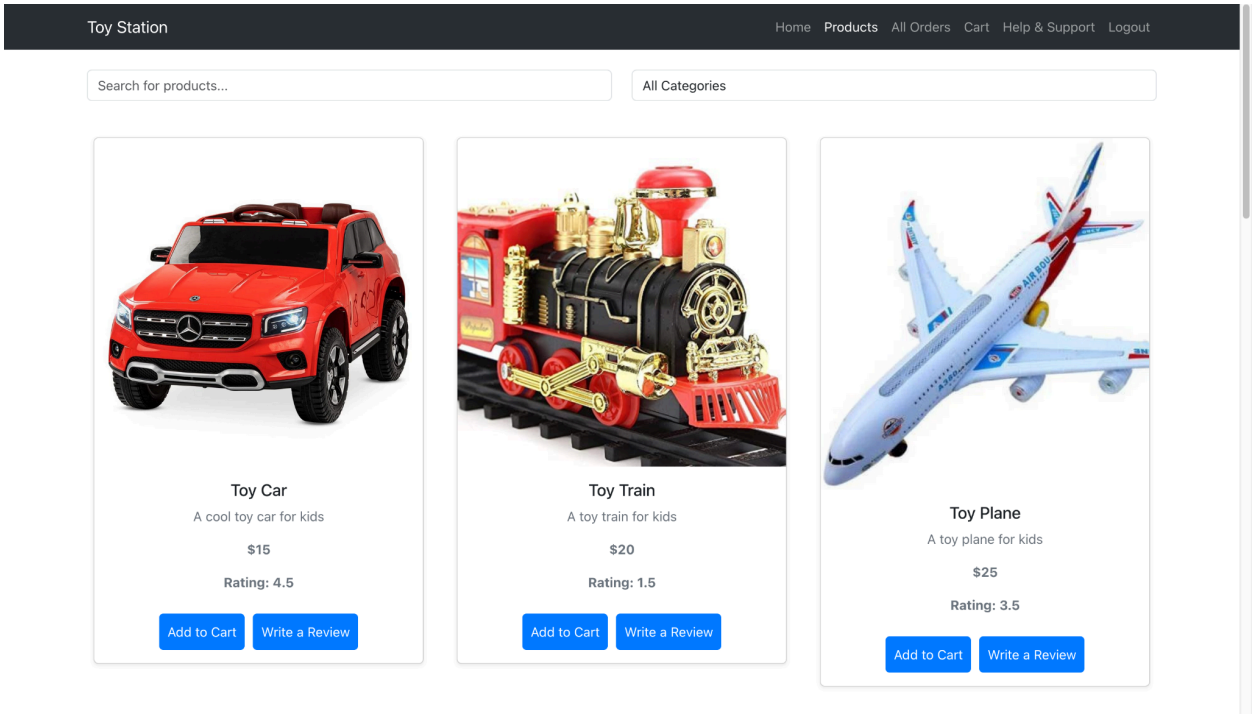
3. **Add to Cart**:

    - Choose the quantity you wish to purchase.
    - Click the "Add to Cart" button.
4. **View Cart**: After adding items, click on the cart icon in the top right corner to review your selected products.

5. **Proceed to Checkout**: Click on the "Checkout" button.

6. **Confirm Order**: Review your order details and click the "Confirm Order" button to complete your purchase.

=

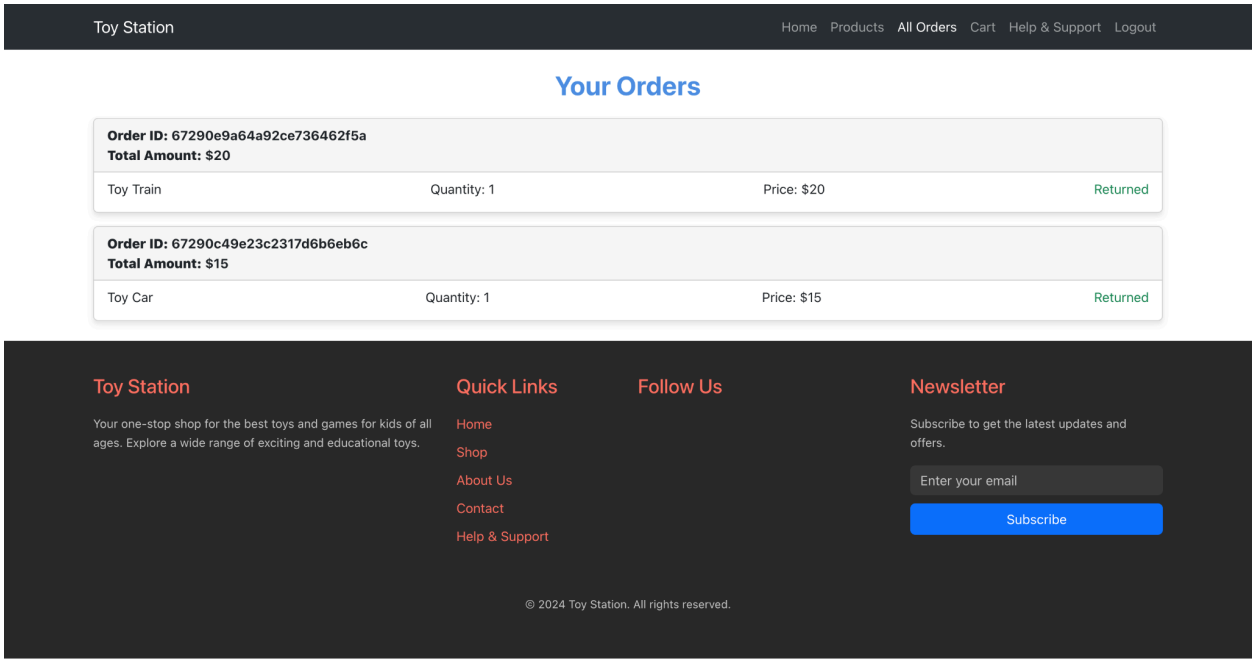


Viewing Orders

1. **Access Your Orders**: Click on the "Your Orders" link in the navigation menu.

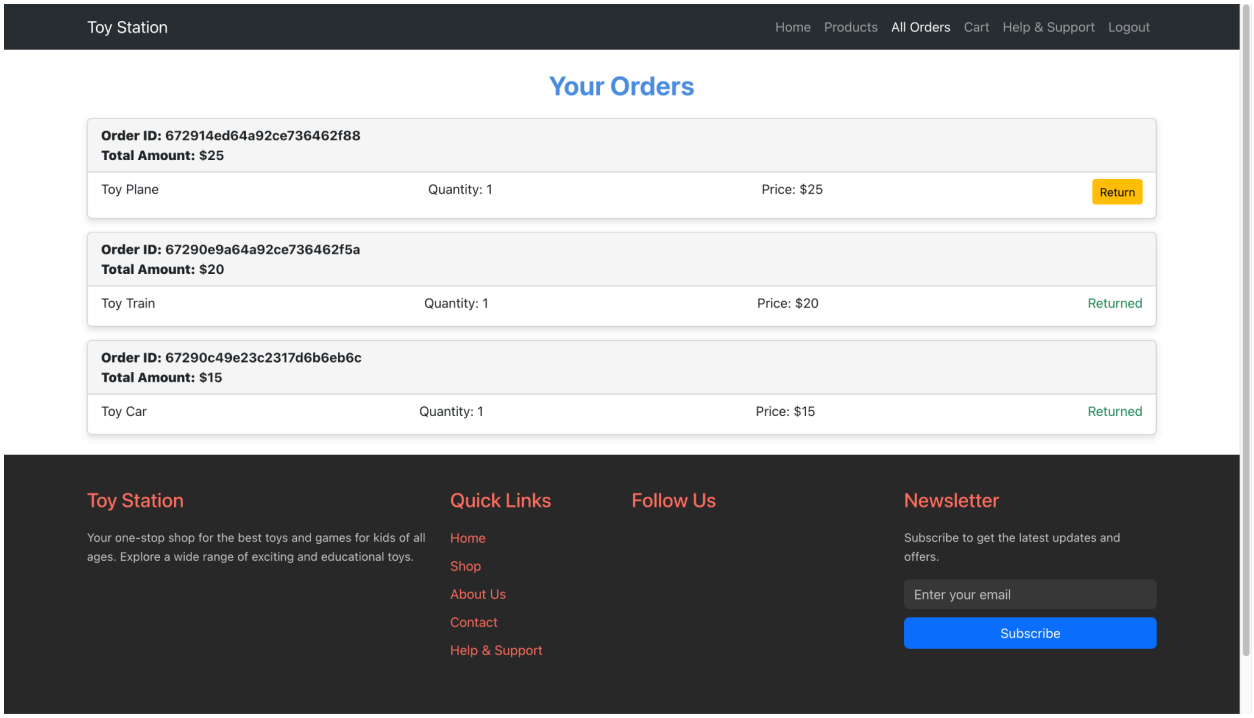
2. **Order List**: You will see a list of your past orders, including details such as order ID, total amount, and order items.

3. **Order Details**: Click on an order to view more detailed information.



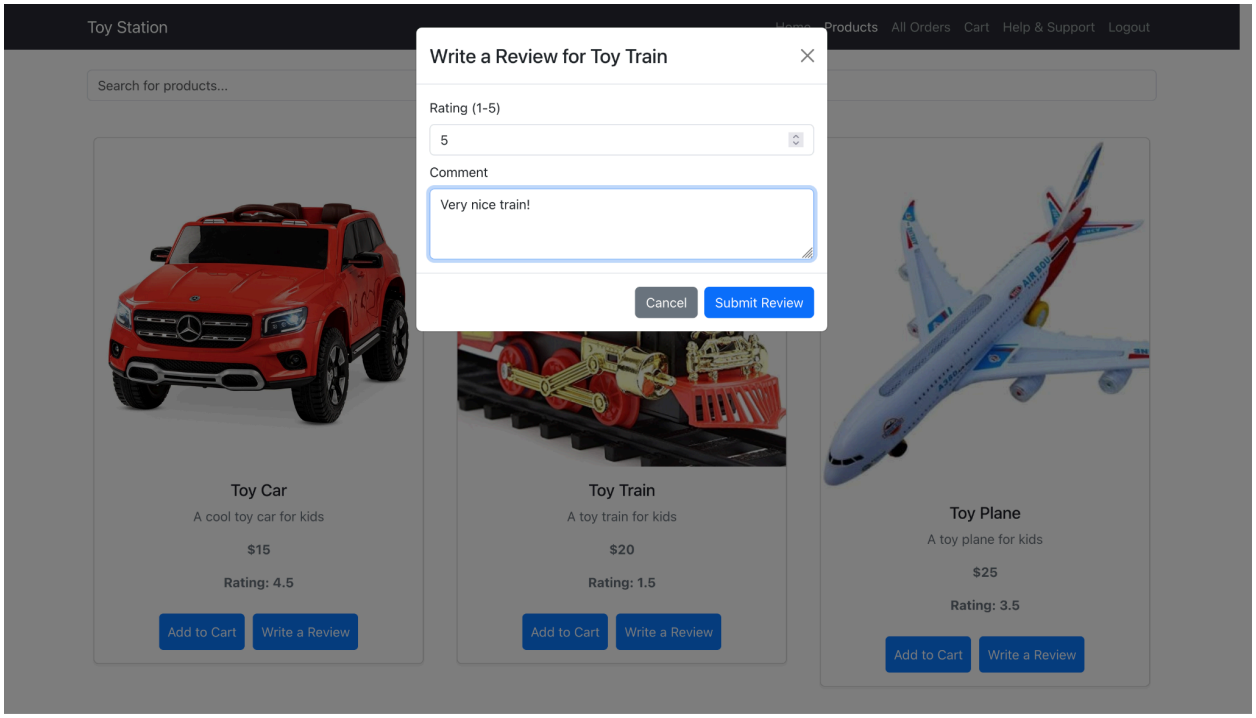
Returning an Item

1. **Navigate to Your Orders**: Go to the "Your Orders" page as described above.
2. **Select an Order**: Click on the order from which you want to return an item.
3. **Return Item**: Click on the "Return" button next to the item you wish to return.
4. **Confirmation**: You will receive a confirmation message indicating that the item has been returned successfully.



Leaving a Product Review

1. **Select a Product**: Go to the product page of the item you want to review.
2. **Write a Review**: Click on the "Leave a Review" button or link.
3. **Fill in the Review Form**: Provide your rating and write your comments.
4. **Submit Your Review**: Click the "Submit" button to share your feedback.
5. **Review Confirmation**: You will see a confirmation that your review has been posted.



Contact Support

If you encounter any issues or have questions, please contact our support team:

- **Email**: support@toystation.com
- **Phone**: +1 (123) 456-7890

Thank you for using our Toy Station! We hope you have a smooth and enjoyable experience.

Code Inspection Feedback

During the code inspection session, our team received valuable feedback aimed at improving the quality, maintainability, and readability of the codebase. Key feedback points and the corresponding actions taken to address them are summarized below:

1. Code Readability

- **Feedback**: Reviewers highlighted the need to improve code readability by using more descriptive variable names and maintaining a consistent coding style throughout the project.
- **Actions Taken**:
    - Refactored variable and function names to ensure they accurately reflect their purpose, making the code more self-explanatory.
    - Adopted a consistent code style guide across the project, aligning with common standards to enhance readability.
    - Added inline comments to explain complex logic within certain functions, making the code easier to understand for future contributors.

2. Modularity

- **Feedback**: The inspection team noted that some components and functions were overly large and could benefit from further modularization. This would increase reusability and simplify future maintenance.
- **Actions Taken**:
    - Broke down larger components and functions into smaller, reusable modules, each focusing on a single responsibility.
    - Organized the code into a modular structure, grouping related functions and components logically to make the codebase more manageable.

    - Leveraged helper functions where possible to encapsulate repeated logic, reducing redundancy and enhancing code efficiency.

3. Documentation

- **Feedback**: Certain functions and classes lacked sufficient documentation, which could hinder understanding for new developers or contributors.
- **Actions Taken**:
    - Created detailed docstrings and comments for all major functions and classes, following a consistent documentation style to ensure clarity.
    - Provided descriptions of input parameters, expected outputs, and functionality for each function, offering clear guidelines for usage.
    - Compiled an updated README file and inline documentation, making it easier for new team members to get familiar with the codebase and understand its structure.

4. Error Handling and Logging

- **Feedback**: Peers recommended improving error handling by adding custom error messages and logging to facilitate debugging and enhance user feedback.
- **Actions Taken**:
    - Implemented custom error messages to give users more informative feedback on issues, especially during registration, login, and order processes.
    - Added logging for critical operations, enabling easier tracking of errors and monitoring of the application's performance during runtime.

Reflection on Accomplishments

Throughout the development of Phase 2 in the **Toy Station** project, our team achieved significant milestones and overcame challenges to deliver a functional and user-friendly application. This reflection highlights what was accomplished, what went well, and areas where improvements could be made.

Accomplishments

- **Order Management System**: Successfully implemented an order management system that allows users to place orders, view their order history, and initiate returns for purchased items. This feature contributes to an efficient shopping experience and provides essential order tracking capabilities.
- **User Authentication**: Implemented secure user registration and login functionality, utilizing encrypted passwords and JWT tokens. This ensures that users can access the platform safely, protecting their information.
- **Product Reviews and Ratings**: Added a review system where users can leave feedback on products. This feature enhances user engagement and provides valuable insights for future buyers, fostering a sense of community within the platform.

What Went Well

- **Team Collaboration**: The team demonstrated strong collaboration, maintaining effective communication and supporting each other throughout the project. Regular check-ins and updates ensured that everyone was aligned with the project goals and deadlines.
- **Adherence to Deadlines**: We successfully met major milestones and adhered to the project timeline. This disciplined approach enabled us to maintain a steady pace and address each requirement in a timely manner.

- **Code Quality and Structure**: By prioritizing modularity and readability, we created a well-structured codebase that facilitates future maintenance and scalability. This effort during Phase 2 lays a solid foundation for subsequent phases.

Areas for Improvement

- **Testing Coverage**: Although we implemented unit tests for core functionalities, there is room to expand our test coverage. Adding more edge cases and conducting integration tests would improve reliability, ensuring that all aspects of the system function as intended under various scenarios.
- **Proactive Feedback Incorporation**: Some feedback from the code inspection session was implemented post-deadline. Establishing a more proactive approach to incorporate feedback throughout the development cycle, rather than waiting for formal sessions, could improve efficiency and lead to earlier identification of potential issues.
- **Error Handling and Logging**: While basic error handling was implemented, adding more detailed error logs and custom error messages for various scenarios could aid in debugging and enhance user feedback. This is an area to address in the next phases.

Member Contribution Table

Member Name	Role	Tasks Assigned	Report Contribution	Overall Contribution (%)

Vaishnavi Shastrula	Project Management Lead	Project planning, timeline management, risk management, team coordination	Wrote the introduction and project planning sections,assembled the final report.	14%
Karunya Mekala	Requirements Lead	Requirements gathering, documentation, wireframe creation	Contributed to the system requirements section	11%
Teja Nagendra Sirigineedi	Configuration Management Lead & Demo Lead	Configuration management, version control, environment setup, demo preparation	Contributed to the system architecture and risk management sections	16%
Sai Sowjanya Edupuganti	Design Lead	UI/UX design, prototyping, design consistency	Contributed to the user interface and sequence diagrams	12%
Katikala Damodar Reddy	Implementation Lead for Frontend	Frontend development, UI components, integration with backend	Contributed to frontend component descriptions in the report	12%
Gayathri Vutla	Implementation Lead for Backend	Backend development, server-side logic,	Contributed to backend and	12%

		database management	database design sections	
Chopra Sai Arani	Testing Lead	Test plan creation, execution of test cases, bug reporting	Contributed to testing strategy and test case descriptions	12%
Preethi Medipelli	Documentation Lead	Project documentation, README management, report compilation	Contributed to the user manual	11%
Shraehitha Reddy Banda	System Administrator Lead	Server management, deployment, system monitoring	Contributed to deployment instructions and system monitoring sections	11%

Core Functionalities: Code Inspection

Phase 1

1. User Registration and Authentication

Description: This module handles user registration, login, and authentication. It ensures secure login by encrypting user passwords using bcrypt.js and authenticates users via JWT tokens.

Source Code:

```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
});

module.exports = mongoose.model('User', UserSchema);
```

Description:

- The user schema stores user information.
- Passwords are encrypted using bcrypt before saving them.
- JWT tokens are generated to authenticate users securely.

---

2. Product Management

Description: This module allows users to view, search, and filter products. It interacts with the MongoDB database to retrieve product information.

Source Code:

```
import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { Button, Card, Col, Container, Row, Form, Dropdown } from 'react-bootstrap';
import { addToCart } from '../redux/actions/cartActions';
import NavbarTop from './NavbarTop';
```

```
import Footer from './Footer';

const ProductList = [
  {
    id: 1,
    name: 'Toy Car',
    description: 'A cool toy car for kids',
    price: 15,
    image:
'https://isakaabengaluru.s3.ap-south-1.amazonaws.com/wp-content/uploads/2022/04/29125018/71dEW
XwH0cL._SL1500_.jpg',
    category: 'Toys',
  },
  {
    id: 2,
    name: 'Toy Train',
    description: 'A toy train for kids',
    price: 20,
    image: 'https://m.media-amazon.com/images/I/511sKmv7JHL.jpg',
    category: 'Toys',
  },
  {
    id: 3,
    name: 'Toy Plane',
    description: 'A toy plane for kids',
    price: 25,
    image:
'https://rukminim2.flixcart.com/image/850/1000/kfk0e4w0/vehicle-pull-along/z/d/h/musical-air-p
lane-running-with-music-not-flying-white-topup-original-imafvzfvkhxu5apr.jpeg?q=90&crop=false'
,
    category: 'Toys',
  },
  {
    id: 4,
    name: 'Toy Robot',
    description: 'A toy robot for kids',
    price: 30,
    image: 'https://m.media-amazon.com/images/I/71j6OP1zHPL.jpg',
    category: 'Toys',
  },
  {
    id: 5,
    name: 'Toy Doll',
    description: 'A toy doll for kids',
    price: 35,
    image: 'https://images.meesho.com/images/products/393169341/ueedc_512.webp',
    category: 'Toys',
  },
  {
    id: 6,
```

```
    name: 'Toy Truck',
    description: 'A toy truck for kids',
    price: 40,
    image: 'https://toyzone.in/cdn/shop/products/72522-02_2048x.jpg?v=1662612960',
    category: 'Toys',
  },
  {
    id: 7,
    name: 'Football',
    description: 'Football for kids',
    price: 45,
    image:
'https://mmtoyworld.com/cdn/shop/files/mm-toys-high-quality-pvc-football-3-no-size-for-age-3-to-10-years-pack-of-1-pc-free-inflating-pin-multicolor_1.jpg?v=1684300546',
    category: 'Sports',
  },
  {
    id: 8,
    name: 'Basketball',
    description: 'Basketball for kids',
    price: 50,
    image:
'https://cdn.pixelbin.io/v2/black-bread-289bfa/HrdP6X/original/hamleys-product/491603946/665/491603946-1_3253.webp',
    category: 'Sports',
  },
  {
    id: 9,
    name: 'Cricket Bat',
    description: 'Cricket bat for kids',
    price: 55,
    image:
'https://gmcricket.in/media/catalog/product/cache/757ea7d2b7282843694bdb6de7a23598/d/i/diamond-606-english-willow-cricket-bat_9.jpg',
    category: 'Sports',
  }
]

const Products = () => {
  const dispatch = useDispatch();
  const cartItems = useSelector((state) => state.cart.cartItems);

  // States for search and filter
  const [searchTerm, setSearchTerm] = useState('');
  const [selectedCategory, setSelectedCategory] = useState('All');

  const handleAddToCart = (product, quantity) => {
    dispatch(addToCart(product, quantity));
  };
```

```
  const handleQuantityChange = (product, change) => {
    const existingItem = cartItems.find((item) => item.id === product.id);
    const newQty = existingItem ? existingItem.quantity + change : 1;

    if (newQty > 0) {
      dispatch(addToCart(product, newQty));
    }
  };

  // Filter products by search term and category
  const filteredProducts = ProductList.filter((product) => {
    const matchesCategory = selectedCategory === 'All' || product.category ===
selectedCategory;
    const matchesSearch = product.name.toLowerCase().includes(searchTerm.toLowerCase());
    return matchesCategory && matchesSearch;
  });

  return (
    <>
      <NavbarTop />
      <Container>
        {/* Search and Filter Section */}
        <Row className="mt-4 mb-4">
          <Col md={6}>
            <Form.Control
              type="text"
              placeholder="Search for products..."
              value={searchTerm}
              onChange={(e) => setSearchTerm(e.target.value)}
            />
          </Col>
          <Col md={6}>
            <Form.Control as="select" value={selectedCategory} onChange={(e) =>
setSelectedCategory(e.target.value)}>
              <option value="All">All Categories</option>
              <option value="Toys">Toys</option>
              <option value="Sports">Sports</option>
            </Form.Control>
          </Col>
        </Row>

        {/* Products Display Section */}
        <Row>
          {filteredProducts.length === 0 ? (
            <p>No products found</p>
          ) : (
            filteredProducts.map((product) => {
              const existingItem = cartItems.find((item) => item.id === product.id);
              const quantity = existingItem ? existingItem.quantity : 0;
```

```
              return (
                <Col key={product.id} md={4} className="mb-4" style={{padding:"20px",}}>
                  <Card className="product-card shadow-sm">
                    <Card.Img variant="top" src={product.image} className="product-image"
style={{width:"100%",}}/>
                    <Card.Body>
                      <Card.Title>{product.name}</Card.Title>
                      <Card.Text>{product.description}</Card.Text>
                      <Card.Text><strong>${product.price}</strong></Card.Text>

                      {quantity > 0 ? (
                        <div className="d-flex align-items-center justify-content-between">
                          <Button variant="danger" onClick={() =>
handleQuantityChange(product, -1)}>
                            -
                          </Button>
                          <span className="mx-2">{quantity}</span>
                          <Button variant="success" onClick={() =>
handleQuantityChange(product, 1)}>
                            +
                          </Button>
                        </div>
                      ) : (
                        <Button variant="primary" onClick={() => handleAddToCart(product, 1)}>
                          Add to Cart
                        </Button>
                      )}
                    </Card.Body>
                  </Card>
                </Col>
              );
            })
          )}
        </Row>
      </Container>
      <Footer />
    </>
  );
};

export default Products;
```

Description:

- The `ProductSchema` defines the structure for product information.
- The `getProducts` function fetches products from the database and allows filtering by category and price range

- React using redux is used for state management which displays product data.

- ---

3. Shopping Cart Functionality

Description: The shopping cart allows users to add products, view the cart, and proceed to checkout.

Source Code:

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

```
import { useDispatch, useSelector } from 'react-redux';

import { Button, Row, Col, ListGroup, Container, Card } from 'react-bootstrap';

import { removeFromCart, updateCartQuantity, clearCart } from
'../redux/actions/cartActions';

import NavbarTop from './NavbarTop';

import axios from 'axios';

import { useNavigate } from 'react-router-dom';

import Footer from './Footer';



const Cart = () => {

  const dispatch = useDispatch();

  const navigate = useNavigate(); // Initialize useNavigate hook

  const cartItems = useSelector((state) => state.cart.cartItems);

  const handleRemoveFromCart = (id) => {

    dispatch(removeFromCart(id));

  };

  const handleQuantityChange = (id, quantity) => {
```

```
    if (quantity > 0) {
      dispatch(updateCartQuantity(id, quantity));
    }
  };

  const totalAmount = cartItems.reduce((acc, item) => acc + item.price *
item.quantity, 0).toFixed(2);

  const handlePlaceOrder = async () => {
    const userEmail = localStorage.getItem('email');

    if (!userEmail) {
      alert('Please log in to place an order.');
      return;
    }

    try {
      const orderData = {
        user: userEmail,
        orderItems: cartItems.map(item => ({
          name: item.name,
          quantity: item.quantity,
          price: item.price,
          total: (item.price * item.quantity),
```

```
      })),
      totalAmount: totalAmount,
    };

    await axios.post('/api/orders/create', orderData); // Ensure this URL is correct
    dispatch(clearCart()); // Clear the cart after order is placed
    alert('Order placed successfully!');
  } catch (error) {
    console.error('Error placing order:', error.response?.data || error.message);
    alert('Failed to place order. Please try again.');
  }
};

return (
  <>
    <NavbarTop />
    <Container className="my-4">
      <Row>
        <Col md={8}>
          <h2>Your Cart</h2>
          <ListGroup variant="flush">
            {cartItems.length === 0 ? (
              <ListGroup.Item>Your cart is empty</ListGroup.Item>
            ) : (
```

```
                cartItems.map((item) => {
                  const itemTotal = (item.price * item.quantity).toFixed(2);
                  return (
                    <ListGroup.Item key={item.id}>
                      <Row className="align-items-center">
                        <Col md={4} className="d-flex align-items-center">
                          <img src={item.image} alt={item.name} className="cart-image"
/>
                          <span className="ms-2">{item.name}</span>
                        </Col>
                        <Col md={4} className="d-flex align-items-center">
                          <Button
                            variant="danger"
                            onClick={() => handleQuantityChange(item.id, item.quantity
- 1)}
                            disabled={item.quantity <= 1}
                          >
                            -
                          </Button>
                          <span className="mx-2">{item.quantity}</span>
                          <Button
                            variant="success"
                            onClick={() => handleQuantityChange(item.id, item.quantity
+ 1)}
                          >
```

```
                        +
                      </Button>
                    </Col>
                    <Col md={2}>${item.price.toFixed(2)}</Col>
                    <Col md={2}>${itemTotal}</Col>
                    <Col md={2}>
                      <Button
                        variant="danger"
                        onClick={() => handleRemoveFromCart(item.id)}
                      >
                        Remove
                      </Button>
                    </Col>
                  </Row>
                </ListGroup.Item>
              );
            })
          )}
        </ListGroup>
      </Col>
      <Col md={4}>
        <Card className="mt-4">
          <Card.Header as="h5">Cart Summary</Card.Header>
          <Card.Body>
```

```
                <Card.Text>
                  <strong>Total Amount:</strong> ${totalAmount}
                </Card.Text>
                <Button variant="primary" className="w-100"
onClick={handlePlaceOrder}>
                  Place order
                </Button>
              </Card.Body>
            </Card>
          </Col>
        </Row>
      </Container>
      <Footer />
    </>
  );
};
export default Cart
```

Description:

- The `CartSchema` tracks user-selected products.
- The `addToCart` function allows users to add items to their cart or update the quantity of existing items. Can view cart summary and real time updates to cart.

---

4. Order Management and Checkout

Description: This module allows users to place orders and tracks their status.

Source Code:

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  user: {
    type: String,
    required: true, // Assuming user will be stored as email
  },
  orderItems: [
    {
      name: { type: String, required: true },
      quantity: { type: Number, required: true },
      price: { type: Number, required: true },
      total: { type: Number, required: true },
    }
  ],
  totalAmount: {
    type: Number,
    required: true,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  }
});

const Order = mongoose.model('Order', orderSchema);
module.exports = Order;
```

Description:

- The `OrderSchema` stores details of the user's orders.
- The `createOrder` function handles the creation of a new order when the user proceeds to checkout.
- Order details and user information is securely stored in MongoDB.

---

5. Backend and Frontend Integration

Description: The backend and frontend communicate using RESTful APIs. The backend handles all user requests and interactions with the database, while the frontend displays data to users.

Source Code (Backend - Example API Endpoint):

Source Code (Frontend - Fetch User Data):

```
const User = require('../models/User');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

// Register a new user
exports.register = async (req, res) => {
  const { username, email, password } = req.body;

  try {
    // Check if the user already exists
    let user = await User.findOne({ email });
    if (user) {
      return res.status(400).json({ msg: 'User already exists' });
    }

    // Create a new user
    user = new User({
      username,
      email,
      password: await bcrypt.hash(password, 10),
    });

    await user.save();

    res.status(201).json({ msg: 'User registered successfully' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ msg: 'Server error' });
  }
};

// Login user
exports.login = async (req, res) => {
  const { email, password } = req.body;
```

```
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ msg: 'Invalid credentials' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ msg: 'Invalid credentials' });
    }

    // Create and assign a token
    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
      expiresIn: '1h',
    });

    res.json({ email: user.email, token });
  } catch (error) {
    console.error(error);
    res.status(500).json({ msg: 'Server error' });
  }
};
```

Description:

- The backend API serves user data, which is fetched and displayed on the frontend using React components.
- For handling data requests a RESTful API is used with Express.

Phase 2

1. Customer Support System

Description:

Focusing on the customer support System it is designed to make help very much accessible for users .

Offers a FAQ section and support request form for accessible and reliable customer support.

Users can reach out for additional support.

Users can submit request form about their issues directly for support team. The form contains all the details along with their concern.

Source Code:

```
import React, { useState } from 'react';
import { Container, Accordion, Card, Form, Button, Col, Row } from
'react-bootstrap';

const HelpSupport = () => {
  // List of Frequently Asked Questions
  const faqs = [
    { question: "How can I track my order?", answer: "You can track your
order from the Orders section in your account." },
    { question: "What is the return policy?", answer: "You can return most
items within 30 days of purchase. Check the Return Policy section for more
details." },
    { question: "How do I contact customer support?", answer: "You can
reach out to us via email or through this support form." },
    { question: "Are there any discounts for new customers?", answer:
"Yes, new customers get a 10% discount on their first order with code
NEW10." },
    { question: "How do I cancel my order?", answer: "To cancel an order,
go to Orders, select the order, and click on 'Cancel Order'." }
  ];

  // State for handling form submission
  const [formData, setFormData] = useState({ name: '', email: '', subject:
'', message: '' });
  const [formSubmitted, setFormSubmitted] = useState(false);

  const handleChange = (e) => {
    const { name, value } = e.target;
```

```
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    // Logic for submitting the form (e.g., send to backend)
    console.log('Form data:', formData);
    setFormSubmitted(true);
    setFormData({ name: '', email: '', subject: '', message: '' });
  };

  return (
    <Container className="mt-5">
      <h2>Help & Support</h2>
      <p>Find answers to common questions or submit a request if you need
further assistance.</p>

      {/* FAQ Section */}
      <h4>Frequently Asked Questions</h4>
      <Accordion>
        {faqs.map((faq, index) => (
          <Accordion.Item eventKey={index.toString()} key={index}>
            <Accordion.Header>{faq.question}</Accordion.Header>
            <Accordion.Body>{faq.answer}</Accordion.Body>
          </Accordion.Item>
        ))}
      </Accordion>

      {/* Request Submission Form */}
      <h4 className="mt-5">Submit a Request</h4>
      {formSubmitted && <p className="text-success">Your request has been
submitted successfully!</p>}
      <Form onSubmit={handleSubmit}>
        <Row>
          <Col md={6}>
            <Form.Group className="mb-3">
              <Form.Label>Name</Form.Label>
```

```
            <Form.Control
              type="text"
              name="name"
              value={formData.name}
              onChange={handleChange}
              required
            />
          </Form.Group>
        </Col>
        <Col md={6}>
          <Form.Group className="mb-3">
            <Form.Label>Email</Form.Label>
            <Form.Control
              type="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              required
            />
          </Form.Group>
        </Col>
      </Row>
      <Form.Group className="mb-3">
        <Form.Label>Subject</Form.Label>
        <Form.Control
          type="text"
          name="subject"
          value={formData.subject}
          onChange={handleChange}
          required
        />
      </Form.Group>
      <Form.Group className="mb-3">
        <Form.Label>Message</Form.Label>
        <Form.Control
          as="textarea"
          rows={4}
```

```
            name="message"
            value={formData.message}
            onChange={handleChange}
            required
          />
        </Form.Group>
        <Button variant="primary" type="submit">Submit Request</Button>
      </Form>
    </Container>
  );
};

export default HelpSupport;
```

---

2. Product Reviews and Ratings

Description: Allows users to submit product feedback, displaying average ratings and fostering community trust.

It makes it easy for buyers to evaluate the products.

Users can leave ratings and comments on each product.

Each product provides an average ratings based on all the reviews which leads to trust on platform.

Source Code:

```
import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { Button, Card, Col, Container, Row, Form, Dropdown, Modal } from
'react-bootstrap';
import { addToCart } from '../redux/actions/cartActions';
import NavbarTop from './NavbarTop';
import Footer from './Footer';
```

```
const ProductList = [
  {
    id: 1,
    name: 'Toy Car',
    description: 'A cool toy car for kids',
    price: 15,
    image:
'https://isakaabengaluru.s3.ap-south-1.amazonaws.com/wp-content/uploads/2022/04/29125018/71dEWXwH0cL._SL1500_.jpg',
    category: 'Toys',
    reviews: [
      { rating: 4, comment: 'Great toy, my kid loves it!' },
      { rating: 5, comment: 'Excellent quality for the price.' },
    ],
  },
  {
    id: 2,
    name: 'Toy Train',
    description: 'A toy train for kids',
    price: 20,
    image: 'https://m.media-amazon.com/images/I/511sKmv7JHL.jpg',
    category: 'Toys',
    reviews: [
      { rating: 1, comment: 'Disappointed with the quality.' },
      { rating: 2, comment: 'It was okay, but could have been better.' },
    ],
  },
  {
    id: 3,
    name: 'Toy Plane',
    description: 'A toy plane for kids',
    price: 25,
    image:
'https://rukminim2.flixcart.com/image/850/1000/kfk0e4w0/vehicle-pull-along/z/d/h/musical-air-plane-running-with-music-not-flying-white-topup-original-imafvzfvkhxu5apr.jpeg?q=90&crop=false',
    category: 'Toys',
```

```
    reviews: [
      { rating: 3, comment: 'Good quality, but could have been better.' },
      { rating: 4, comment: 'I love it!' },
    ],
  },
  {
    id: 4,
    name: 'Toy Robot',
    description: 'A toy robot for kids',
    price: 30,
    image: 'https://m.media-amazon.com/images/I/71j6OP1zHPL.jpg',
    category: 'Toys',
    reviews: [
      { rating: 5, comment: 'I love it!' },
      { rating: 5, comment: 'I love it!' },
    ],
  },
  {
    id: 5,
    name: 'Toy Doll',
    description: 'A toy doll for kids',
    price: 35,
    image:
'https://images.meesho.com/images/products/393169341/ueedc_512.webp',
    category: 'Toys',
    reviews: [
      { rating: 4, comment: 'Superb quality, my kid loves it!' },
      { rating: 5, comment: 'Excellent quality for the price.' },
    ],
  },
  {
    id: 6,
    name: 'Toy Truck',
    description: 'A toy truck for kids',
    price: 40,
    image:
'https://toyzone.in/cdn/shop/products/72522-02_2048x.jpg?v=1662612960',
```

```
    category: 'Toys',
    reviews: [
      { rating: 2, comment: 'Average quality, the toy is not as good as
expected.' },
      { rating: 3, comment: 'Could have been better.' },
    ],
  },
  {
    id: 7,
    name: 'Football',
    description: 'Football for kids',
    price: 45,
    image:
'https://mmtoyworld.com/cdn/shop/files/mm-toys-high-quality-pvc-football-3
-no-size-for-age-3-to-10-years-pack-of-1-pc-free-inflating-pin-multicolor_
1.jpg?v=1684300546',
    category: 'Sports',
    reviews: [
      { rating: 4, comment: 'More quality, but could have been better.' },
      { rating: 5, comment: 'The ball is too small for my kid.' },
    ],
  },
  {
    id: 8,
    name: 'Basketball',
    description: 'Basketball for kids',
    price: 50,
    image:
'https://cdn.pixelbin.io/v2/black-bread-289bfa/HrdP6X/original/hamleys-pro
duct/491603946/665/491603946-1_3253.webp',
    category: 'Sports',
    reviews: [
      { rating: 3, comment: 'Basketball is not as good as expected.' },
      { rating: 4, comment: 'I love it!' },
    ],
  },
  {
```

```
    id: 9,
    name: 'Cricket Bat',
    description: 'Cricket bat for kids',
    price: 55,
    image:
'https://gmcricket.in/media/catalog/product/cache/757ea7d2b7282843694bdb6d
e7a23598/d/i/diamond-606-english-willow-cricket-bat_9.jpg',
    category: 'Sports',
    reviews: [
      { rating: 4, comment: 'Absolutely love it!' },
      { rating: 3, comment: 'Just okay.' },
    ],
  }
]

const calculateAverageRating = (reviews) => {
  if (reviews.length === 0) return 0;
  const totalRating = reviews.reduce((acc, review) => acc + review.rating,
0);
  return (totalRating / reviews.length).toFixed(1);
};


const Products = () => {
  const [newReview, setNewReview] = useState({ rating: 0, comment: '' });
  const [allReviews, setAllReviews] = useState(ProductList);
  const [showModal, setShowModal] = useState(false);
  const [selectedProduct, setSelectedProduct] = useState(null);

  const handleReviewSubmit = () => {
    if (selectedProduct) {
      const updatedReviews = allReviews.map((product) => {
        if (product.id === selectedProduct.id) {
          return {
            ...product,
            reviews: [...product.reviews, newReview],
          };
```

```
      }
      return product;
    });
    setAllReviews(updatedReviews);
    setShowModal(false);
    setNewReview({ rating: 0, comment: '' });
  }
};

const openReviewModal = (product) => {
  setSelectedProduct(product);
  setShowModal(true);
};

const closeReviewModal = () => {
  setShowModal(false);
  setNewReview({ rating: 0, comment: '' });
};

const dispatch = useDispatch();
const cartItems = useSelector((state) => state.cart.cartItems);

// States for search and filter
const [searchTerm, setSearchTerm] = useState('');
const [selectedCategory, setSelectedCategory] = useState('All');

const handleAddToCart = (product, quantity) => {
  dispatch(addToCart(product, quantity));
};

const handleQuantityChange = (product, change) => {
  const existingItem = cartItems.find((item) => item.id === product.id);
  const newQty = existingItem ? existingItem.quantity + change : 1;

  if (newQty > 0) {
    dispatch(addToCart(product, newQty));
  }
```

```
};

// Filter products by search term and category
const filteredProducts = ProductList.filter((product) => {
```

```
    const matchesCategory = selectedCategory === 'All' || product.category
=== selectedCategory;
    const matchesSearch =
product.name.toLowerCase().includes(searchTerm.toLowerCase());
    return matchesCategory && matchesSearch;
  });
```

3. Recommended Products

Description: It displays suggested products based on user cart and their purchase history.

Recommendations are based on predefined relations between products. Like for “Toy Car” they might get recommendations like “Toy Train” or “Toy Bus”.

It also considers the browsing pattern and previous purchase history.

Source Code:

```
  const recommendations = {
    1: [2, 3],
    2: [5],
    3: [1, 4],
    4: [3, 6],
    5: [2],
    // Add other recommendations as needed
  };

  // Function to get recommended products based on product ID
  const getRecommendedProducts = (productId) => {
    return ProductList.filter(product =>
recommendations[productId]?.includes(product.id));
  };

  return (
    <>
      <NavbarTop />
      <Container>
        {/* Search and Filter Section */}
        <Row className="mt-4 mb-4">
          <Col md={6}>
            <Form.Control
              type="text"
              placeholder="Search for products..."
```

```
              value={searchTerm}
              onChange={(e) => setSearchTerm(e.target.value)}
            />
          </Col>
          <Col md={6}>
            <Form.Control as="select" value={selectedCategory}
onChange={(e) => setSelectedCategory(e.target.value)}>
              <option value="All">All Categories</option>
              <option value="Toys">Toys</option>
              <option value="Sports">Sports</option>
            </Form.Control>
          </Col>
        </Row>

        {/* Products Display Section */}
        <Row>
          {filteredProducts.length === 0 ? (
            <p>No products found</p>
          ) : (
            filteredProducts.map((product) => {
              const productWithReviews = allReviews.find((p) => p.id ===
product.id);
              const averageRating =
calculateAverageRating(productWithReviews.reviews);

              const existingItem = cartItems.find((item) => item.id ===
product.id);
              const quantity = existingItem ? existingItem.quantity : 0;

              // show recommended products only if the product is in the
cart
              const recommendedProducts = existingItem ?
getRecommendedProducts(product.id) : [];

              return (
                <Col key={product.id} md={4} className="mb-4"
style={{padding:"20px",}}>
```

```
                  <Card className="product-card shadow-sm">
                    <Card.Img variant="top" src={product.image}
classNameName="product-image"  style={{width:"100%",}}/>
                    <Card.Body>
                      <Card.Title>{product.name}</Card.Title>
                      <Card.Text>{product.description}</Card.Text>

<Card.Text><strong>${product.price}</strong></Card.Text>
                      <Card.Text><strong>Rating:
{averageRating}</strong></Card.Text>

                      {quantity > 0 ? (
                        <div className="d-flex align-items-center
justify-content-between">
                          <Button variant="danger" onClick={() =>
handleQuantityChange(product, -1)}>
                            -
                          </Button>
                          <span className="mx-2">{quantity}</span>
                          <Button variant="success" onClick={() =>
handleQuantityChange(product, 1)}>
                            +
                          </Button>
                        </div>
                      ) : (
                        <Button variant="primary" onClick={() =>
handleAddToCart(product, 1)}>
                          Add to Cart
                        </Button>
                      )}

                      <Button variant="secondary" onClick={() =>
openReviewModal(product)} style={{marginLeft:"10px"}}>
                        Write a Review
                      </Button>

                    </Card.Body>
```

```
                  </Card>

                  {/* Recommended Shelf */}
                {recommendedProducts.length > 0 && (
                  <div className="mt-3">
                    <h5>Recommended for you</h5>
                    <Row>
                      {recommendedProducts.map((recommendedProduct) => (
                        <Col key={recommendedProduct.id} md={6}
className="mb-2">
                          <Card className="recommend-card shadow-sm">
                            <Card.Img variant="top"
src={recommendedProduct.image} className="product-image"
style={{height:"80px"}}/>
                            <Card.Body>
                              <Card.Title className="text-truncate"
style={{fontSize:"14px"}}>{recommendedProduct.name}</Card.Title>
                              <Button variant="primary" size="sm"
onClick={() => handleAddToCart(recommendedProduct, 1)}>
                                Add
                              </Button>
                            </Card.Body>
                          </Card>
                        </Col>
                      ))}
                    </Row>
                  </div>
                )}
                </Col>
              );
            })
          )}
        </Row>
      </Container>

      <Modal show={showModal} onHide={closeReviewModal}>
      <Modal.Header closeButton>
```

```
        <Modal.Title>Write a Review for
{selectedProduct?.name}</Modal.Title>
      </Modal.Header>
      <Modal.Body>
        <Form>
          <Form.Group>
            <Form.Label>Rating (1-5)</Form.Label>
            <Form.Control
              type="number"
              min="1"
              max="5"
              value={newReview.rating}
              onChange={(e) => setNewReview({ ...newReview, rating:
parseInt(e.target.value) })}
            />
          </Form.Group>
          <Form.Group className="mt-2">
            <Form.Label>Comment</Form.Label>
            <Form.Control
              as="textarea"
              rows={3}
              value={newReview.comment}
              onChange={(e) => setNewReview({ ...newReview, comment:
e.target.value })}
            />
          </Form.Group>
        </Form>
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={closeReviewModal}>
          Cancel
        </Button>
        <Button variant="primary" onClick={handleReviewSubmit}>
          Submit Review
        </Button>
      </Modal.Footer>
      </Modal>
```

```
      <Footer />
    </>
  );
};

export default Products;
```

Meeting minutes

Meeting 1

• Date: October 23nd, 2024

• Time: 3:00 PM – 4:00 PM

• Location: Google Meet

• Attendees: All team members

Agenda:

1) Finding and allocating new tasks for Deliverable 4 after reviewing the feedback from Deliverable 3.

2) Talk about updated backend and frontend functionalities like tracking orders, managing returns, and filtering.

Action Items:

1) Assign tasks for improved functionalities and finalize the scope.

2) Work on API for order tracking and frontend filtering options.

3) Set up a demonstration environment for continued work.

Meeting 2

• Date: October 30th, 2024

• Time: 3:00 PM – 4:00 PM

• Location: Google Meet

• Attendees: All team members

Agenda:

1) Evaluate the status of backend and frontend improvements.

- Options for filtering and a redesigned checkout user interface.
- payment API and tracking order.

2) Talk about the new features' documentation requirements

3) Arrange for preliminary testing of the new features.

Action Items:

1) Finish order tracking and frontend filtering by the following meeting.

2) Get started writing API documentation for the most recent backend endpoints.

3) For new frontend and backend features, set up preliminary testing scenarios.

Meeting 3

• Date: November 3rd, 2024

• Time: 3:00 PM – 4:00 PM

• Location: Teams

• Attendees: All team members

Agenda:

1) Complete the backend and frontend features.

2) Verify the integration of the payment gateway and test the payment process.

3) Add deployment setup instructions and update the user manual.

4) Start testing for integration.

Action Items:

1) Update the user manual to include new features such as order tracking and payment processing.

2) Create a checklist to ensure you are ready for deployment.

Meeting 4

• Date: November 8th, 2024

• Time: 4:00 PM – 5:00 PM

• Location: Google Meet

• Attendees: All team members

Agenda:

1) Examine the outcomes of security and integration testing.

2) Ask for opinions on improvements to the user manual and documentation.

3) Talk about the presentation's outline and demo setup.

Action Items:

1) Complete all required documentation such as the setup and user manual.

2) Make sure that every feature is represented.

3) Prepare presentation slides that highlight the main results and milestones of the project.

Meeting 5

• Date: November 10th, 2024

• Time: 3:00 PM – 4:00 PM

• Location: Teams

• Attendees: All team members

Agenda:

1) Final review of the submission of Deliverable 4.

2) Verify that all tasks, documentation, and demo preparations have been completed.

3) Once submitted, review the project's objectives and future actions.

Action Items:

1) Make sure that the report, documentation, and code are all up to date before submitting Deliverable 4.

2) Get ready for the final project and get input for changes.