Project Title: ToyStation
Deliverable 5

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)

Requirements

1. Reason for Not Implementing the Recommendation Engine

The "Recommendation Engine" is a sophisticated feature designed to enhance user engagement and drive personalized experiences by suggesting items that align with user preferences, past purchases, or browsing history. While this feature has significant benefits, its implementation was not feasible during the development of this project due to the following reasons:

1. Complexity of Recommendation Algorithms

- **Key Challenge**: Developing an effective recommendation engine requires advanced algorithms, such as collaborative filtering, content-based filtering, or hybrid approaches. These algorithms involve analyzing large datasets, identifying patterns, and predicting user preferences, which demand a strong foundation in machine learning or data science.
- **Reason for Difficulty:** The current project timeline and skill set were not sufficient to implement or integrate such algorithms, as they require extensive research, testing, and fine-tuning to achieve meaningful and accurate recommendations**.**

2. Lack of Sufficient User Data

- Key Challenge: Recommendation engines heavily rely on large volumes of user data, such as purchase history, browsing behavior, and item ratings. The current application did not yet have an active user base or historical data to feed into such algorithms.

- **Reason for Difficulty:** With no existing user activity or transactional data, it was impossible to generate personalized recommendations. The absence of a dataset rendered even basic recommendation logic (like frequently bought items or similar product tags) unviable.

3. Resource and Time Constraints

- **Key Challenge**: Building a recommendation system, even a basic one, requires dedicated resources, including backend infrastructure, additional APIs, and time to design and test the system.
- **Reason for Difficulty:** The project was focused on core functionalities like order tracking, payment integration, and delivery management, which were prioritized due to resource and time constraints. This left insufficient capacity to allocate toward a recommendation feature.

4. Limited Backend Infrastructure

- **Key Challenge:** Implementing a recommendation engine necessitates robust backend infrastructure capable of handling dynamic data analysis, generating personalized outputs, and ensuring real-time responsiveness.
- **Reason for Difficulty:** The current backend system was designed for CRUD operations and basic data management, without the computational and storage capabilities required for recommendation algorithms. Enhancements to the backend architecture were beyond the scope of this phase of development.

5. Dependency on Machine Learning Expertise

- **Key Challenge:** A well-optimized recommendation engine often requires leveraging machine learning libraries or services to analyze user data and predict preferences.
- **Reason for Difficulty:** The development team did not have expertise in machine learning or access to pre-trained recommendation models. While third-party APIs like Google Recommendations AI or AWS Personalize could be explored, integrating these services would have required additional time and budget.

6. Alternative Implementation Attempted

- **Hardcoded Recommendations:** An attempt was made to simulate the recommendation feature using hardcoded suggestions based on product categories or tags. For example, related items like "Toy Train" or "Toy Plane" could be displayed when viewing a "Toy Car."
- **Reason for Not Fully Implementing**: Even this basic approach required manual mapping of related products and lacked the dynamic, personalized nature of a true recommendation engine. It was deemed insufficient to meet the desired user experience goals.

4. **Reason for Not Implementing the Customer Support System**
The "Help & Support" system was conceptualized to enhance user experience by providing instant assistance through FAQs and personalized support requests. However, the following reasons prevented the feature from being implemented within the project timeline:

1. Backend Infrastructure and Data Management Constraints

- **Lack of a dedicated backend for FAQ management**:
    - A dynamic FAQ system requires a database or CMS integration to store, update, and retrieve frequently asked questions efficiently.
    - Setting up this infrastructure was deprioritized to focus on core functionalities like user management, orders, and payments.
- **Support ticketing system**:
    - Building a system to collect, store, and manage support requests in the backend (e.g., creating APIs for handling form submissions) requires robust backend support.
    - This would include integrating a notification system to alert the support team about new submissions, which was not scoped within the timeline.

2. Limited Time for Frontend Development

- Designing an accordion-based FAQ interface and integrating it with a form submission system requires additional time for seamless UI/UX implementation.
- Prioritizing higher-impact features like order management and delivery tracking left insufficient time to implement this medium-priority feature.

3. Lack of Third-Party Support Integration

- Many modern customer support systems rely on third-party tools like Zendesk or Freshdesk for easy setup and scalability.
- Evaluating, integrating, and customizing such tools for the project was not feasible within the development timeline.

4. Testing and Validation Challenges

- The feature would require thorough testing to ensure:
    - Accurate FAQ retrieval and display.
    - Successful form submission and backend response handling.
- Due to time constraints, other high-priority features consumed the testing window.

Reason for Not Implementing the Product Reviews and Ratings Feature
The "Product Reviews and Ratings" feature was designed to enhance user interaction and decision-making by enabling customers to share and view feedback. However, it was not implemented due to the following reasons:

1. Backend Infrastructure and Database Complexity

- **Dynamic Storage for Reviews**:
    - Implementing a review system requires designing a database schema that supports storing user reviews, ratings, and linking them to specific products and users.

    - Features such as calculating average ratings, storing timestamps, and preventing duplicate reviews demand advanced data handling, which was not prioritized within the project timeline.
- **Real-Time Updates**:
    - Updating product ratings dynamically based on new reviews would require real-time data syncing between the frontend and backend, which adds to the development complexity.

2. Frontend Development Challenges

- **UI/UX Implementation**:
    - Designing a modal for submitting reviews and integrating it seamlessly with product cards requires additional time and effort.
    - Displaying average ratings dynamically and ensuring an intuitive user experience for writing reviews were deprioritized to focus on core functionality like order placement and tracking.
- **Error Handling**:
    - Addressing potential issues such as incomplete reviews, duplicate submissions, or rating mismatches was an added layer of complexity that required more development and testing time.

3. Moderation and Content Validation

- **User-Generated Content Risks**:
    - Reviews involve user-generated content, which raises concerns about inappropriate or misleading comments.
    - Building a moderation system to filter or approve reviews before displaying them was beyond the scope of the current phase.

4. Limited Time and Resource Allocation

- The feature was categorized as medium priority and deferred to prioritize features like delivery tracking, payment integration, and restaurant management, which were critical to the core functionality of the platform.
- Adequate testing and debugging time for this feature were unavailable due to the focus on higher-priority functionalities.

5. Third-Party Integration Delays

- While third-party tools such as Firebase or review-specific APIs could have expedited the implementation, integrating and customizing them required additional research and development time that was unavailable during this phase.

.

Reason for Not Implementing the Returns and Refunds Feature

The "Returns and Refunds" feature was planned to improve customer satisfaction by enabling seamless post-purchase management. However, it was not implemented due to the following reasons:

1. Backend Complexity and Infrastructure Challenges

- **Database Schema Adjustments**:

    - Implementing this feature required additional fields in the orders database, such as return_status, refund_amount, and timestamps for return requests, which were not part of the initial schema design.
- **Return and Refund Logic**:
    - Establishing workflows for validating return requests, approving or rejecting them, and processing refunds added significant complexity. This required well-defined business rules and decision-making logic that were beyond the scope of the current development phase.
- **Payment Gateway Integration**:
    - Integrating with Stripe or another payment processor to handle refunds automatically required additional research and implementation effort, which conflicted with other high-priority tasks.

2. Frontend Development Challenges

- **User Interface Design**:
    - The feature required creating intuitive and secure UI components, such as a "Return" button and a modal or form for collecting return-related details (e.g., reason for return, photos of damaged items). Designing these components with a user-friendly flow was deprioritized in favor of essential features like order placement and delivery tracking.
- **Dynamic Order Updates**:
    - Updating the order history page dynamically to reflect the return status in real-time added further complexity.

3. Business Process and Policy Dependencies

- **Return Policies**:
    - Clearly defining and implementing return policies (e.g., return window, eligibility criteria) required input from stakeholders that was unavailable during this phase.
- **Logistics Coordination**:
    - Handling the logistics of product returns (e.g., pickup scheduling, warehouse management) was not integrated into the system, making the feature incomplete without these operational aspects.

4. Limited Resources and Timeline Constraints

- This feature, though marked as high priority, was deferred due to resource constraints and the need to focus on foundational aspects of the application, such as user registration, restaurant management, and payment processing.
- Development time was insufficient to implement, test, and debug the full lifecycle of a return request, from initiation to refund completion.

5. Dependence on External Systems

- The feature required coordination with external systems such as courier services for return pickups and payment gateways for refunds, which added technical and logistical dependencies that could not be resolved within the current project phase.

Live Chat Implementation (Requirement 4)

Reason for Change: Implementing a live chat feature requires a robust back-end service to handle real-time messaging and user authentication, as well as a front-end interface for users to interact with. Depending on the existing architecture, this could involve significant modifications or the integration of third-party services like Firebase, Socket.IO, or dedicated customer service platforms. If the current setup lacks the necessary infrastructure or if time constraints are a factor, it might not be feasible to implement a fully functional live chat feature at this stage.

Stripe Payment Gateway Integration

The **Stripe Payment Gateway** was integrated into the application to enable secure, seamless, and reliable online payments for customer orders. This implementation plays a vital role in the platform's functionality, ensuring a smooth transaction process and building trust with users by offering a globally recognized payment solution.

Implementation Details

1. **Frontend Integration**:
    - The Stripe SDK (@stripe/react-stripe-js and @stripe/stripe-js) was used to incorporate payment functionality into the frontend.
    - A **Checkout Page** was designed, allowing customers to:
        - Review their order summary, including item details, quantity, and total price.
        - Enter their payment details via Stripe’s secure payment elements (e.g., card number, expiry date, and CVV fields).
    - Real-time validation and error handling were implemented to guide users through the payment process and ensure accuracy.
2. **Backend Setup**:
    - A dedicated backend API endpoint was created to handle payment intent creation. This endpoint:
        - Generates a **Payment Intent** using the Stripe API, which includes the total amount to be paid and the currency.

        - Returns the client secret to the frontend, enabling Stripe to process the transaction securely.
    - Upon successful payment, a webhook was configured to handle events from Stripe, such as payment_intent.succeeded, ensuring that the order status is updated to reflect the payment.

3. **Security Measures**:
    - All sensitive operations (e.g., amount calculations and payment intent creation) were performed on the server side to prevent tampering.
    - Stripe's PCI DSS compliance ensures that card information is handled securely without exposing it to the application.
4. **Order Association**:
    - Once the payment succeeds, the corresponding order is finalized, and the payment details are saved in the database. This ensures:
        - Transparency in the transaction process.
        - A clear audit trail for both the platform and customers.
    - Customers receive a payment confirmation via email, enhancing communication and trust.

Benefits of Stripe Integration

1. **Ease of Use**:
    - Stripe's pre-built UI components, such as **Stripe Elements**, simplified the development process while ensuring a professional and user-friendly experience.
2. **Global Support**:
    - Stripe supports multiple currencies and payment methods, making the platform ready for international scalability.
3. **Security and Compliance**:
    - Stripe ensures PCI compliance, fraud prevention, and encryption, relieving the development team from handling sensitive financial data directly.

4. **Seamless Refunds**:
    - Future enhancements can leverage Stripe's built-in refund functionality to align with planned features like **Returns and Refunds**.
5. **Scalability**:
    - Stripe's robust APIs allow easy extension of payment features, such as adding wallet payments, installment plans, or subscription services in the future.

User Experience

Customers benefit from a secure and efficient checkout process:

- **Streamlined Payment Flow**: Payments are completed within a few clicks, reducing friction during checkout.
- **Real-Time Feedback**: Instantaneous success or error messages ensure clarity and reduce customer frustration.
- **Trustworthy Interface**: Familiar card input forms reassure users about the platform's credibility.

Google Maps Integration for Delivery Directions

The integration of **Google Maps** into the application provides a seamless navigation experience, allowing delivery agents or customers to view and follow the most efficient route to the delivery location. This feature enhances the practicality and user experience of the application by offering accurate location tracking and step-by-step directions.

Implementation Details

1. **Frontend Integration**:
    - **Google Maps JavaScript API** and **Google Maps URL Scheme** were used to implement the navigation feature.
    - A **"Get Directions"** button was added to the order details page, adjacent to each order or delivery location.
    - When the button is clicked, the following steps are executed:

        - The application retrieves the **current location** of the user using the browser's Geolocation API.
        - The **delivery address** is retrieved from the order details, which contains latitude and longitude coordinates.
        - These two points (current location and delivery address) are formatted into a URL that redirects the user to Google Maps with the route pre-loaded.

2. **Backend Support**:
    - The backend provides the delivery location (latitude and longitude) as part of the order details, ensuring accurate data availability for the frontend.
    - No additional backend logic was needed, as the routing functionality is handled by Google Maps.
3. **User Experience Enhancements**:
    - A **loading indicator** is displayed while retrieving the current location to inform users of ongoing processes.
    - Error handling is implemented to manage cases where the location cannot be retrieved (e.g., permission denied by the user) or if the delivery location is invalid.

Benefits of Google Maps Integration

1. **Efficient Navigation**:
    - Delivery agents can quickly and easily find the fastest route to the destination, saving time and improving operational efficiency.
2. **Accuracy**:
    - Leveraging Google Maps ensures reliable routing and navigation, as it accounts for real-time traffic updates and road conditions.
3. **User-Friendly**:
    - The feature requires minimal user effort; with a single click, users are directed to Google Maps with the route pre-configured.

4. **Cross-Platform Compatibility**:
    - By opening Google Maps in the browser or via the mobile app, the feature works seamlessly across both desktop and mobile platforms.
5. **Scalability**:
    - The integration can be extended to incorporate additional features like **estimated time of arrival (ETA)**, delivery route optimization for multiple orders, or tracking the delivery agent's live location.

Challenges Addressed

- **Dynamic Location Handling**:
    - By utilizing the Geolocation API to fetch the user's current position and combining it with order-specific delivery coordinates, the feature dynamically adapts to any location.
- **Real-Time Redirection**:
    - The use of Google Maps URL Scheme ensures real-time redirection without requiring complex in-app mapping functionality, reducing development overhead while leveraging Google's robust mapping services.

User Workflow

1. The delivery agent views their assigned order in the application.
2. They click the **"Get Directions"** button next to the order.
3. The application:
    - Fetches the agent's current location.
    - Retrieves the delivery location from the order details.
    - Opens Google Maps in a new browser tab or app, displaying the route.
4. The agent follows the route displayed in Google Maps to reach the destination.

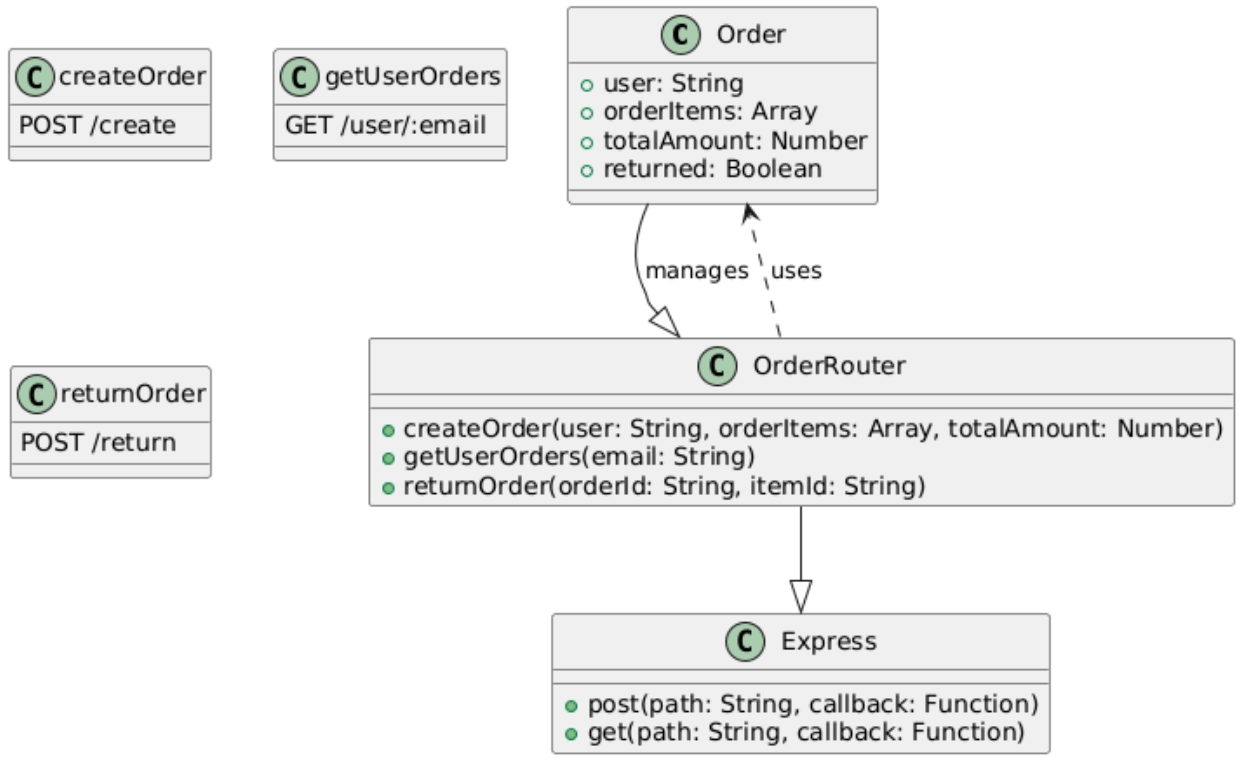
Overall Impact

These adjustments highlight the importance of aligning feature implementation with the current capabilities of the system and the team’s expertise. While these features would significantly enhance the application's functionality, prioritizing feasible and achievable tasks ensures that the development process remains on track and focused on delivering a stable product within the available resources and time constraints.

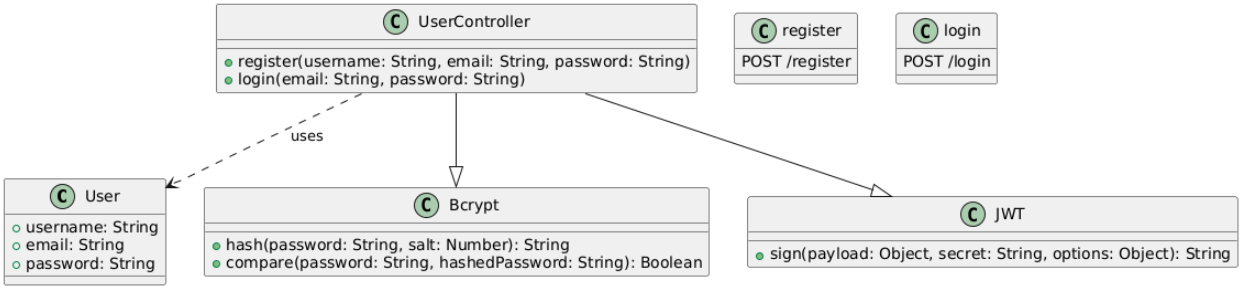
Report UML

Class Diagram

Order

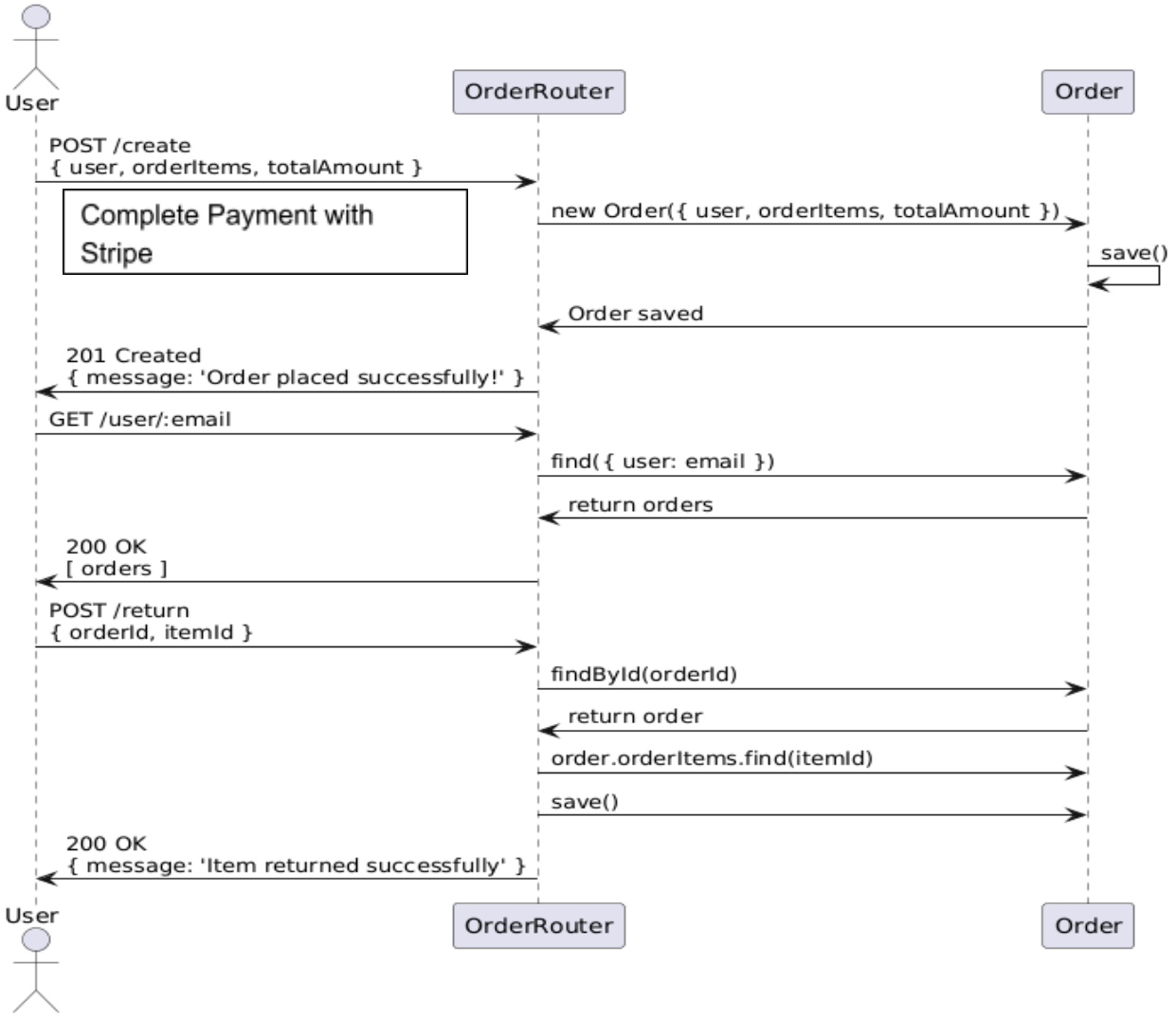


User

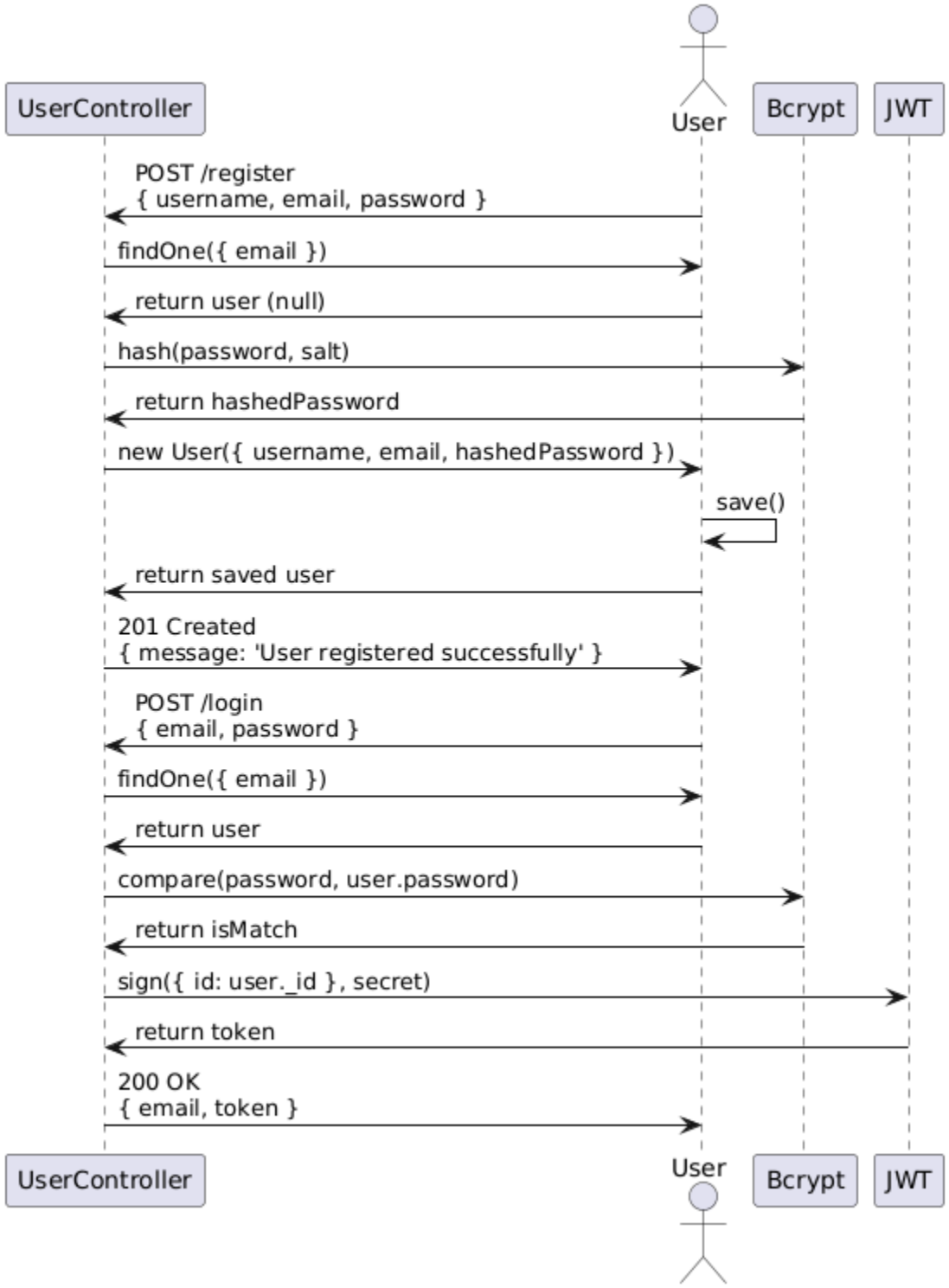


Sequence Diagram

Order Management

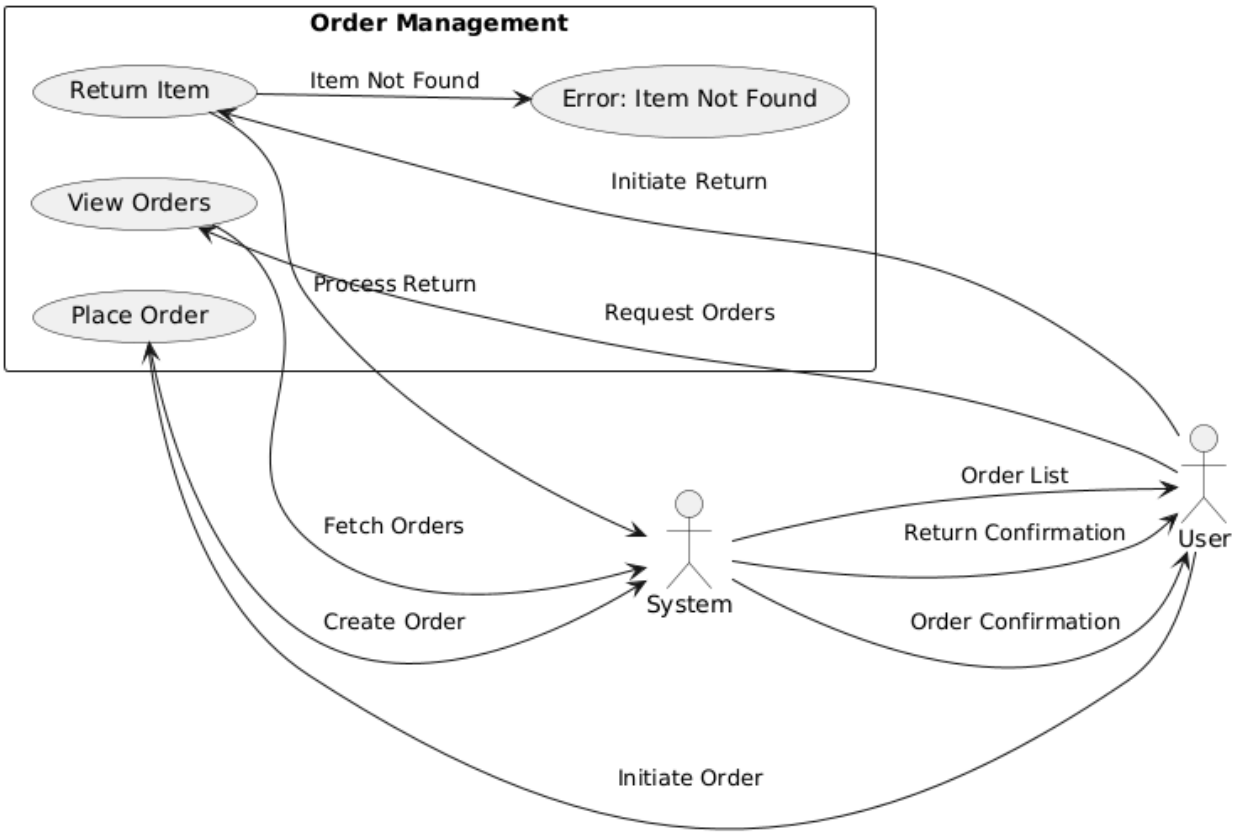


User Management

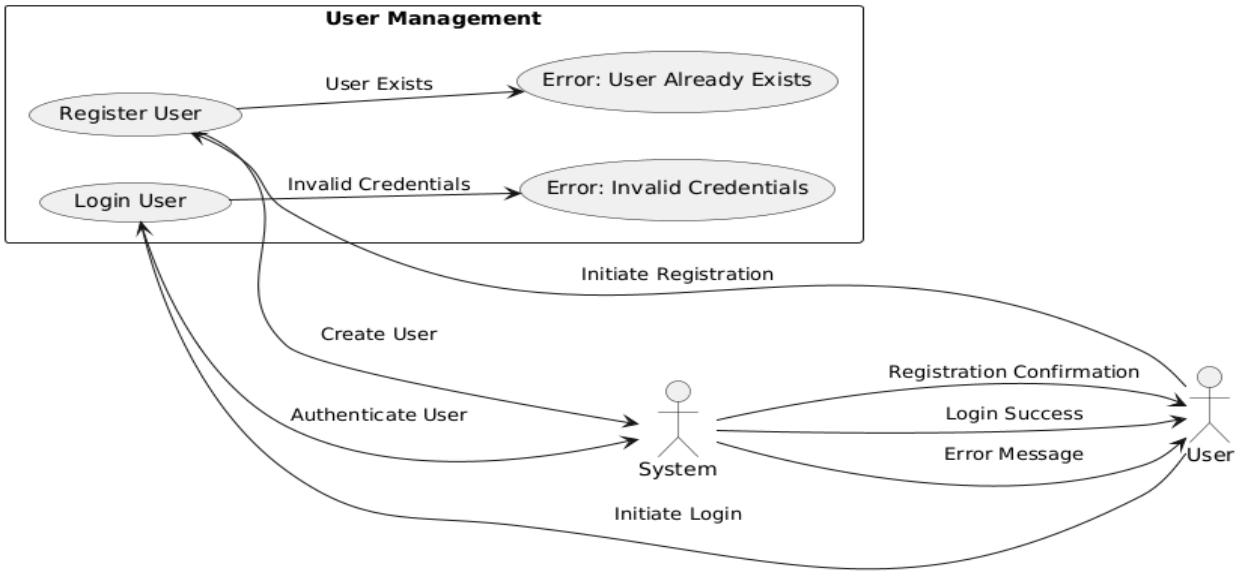


Use Case Diagram

Order Managemen



User Management



Test Cases

- Unit Test Cases for Order Management

Test Case: Create Order Successfully

- **Description**: Tests the ability to create a new order successfully.
- **Pre-requisites**:
  - User with email `"testuser@example.com"` must be registered.
  - The product with `id: 1`, `name: "Toy Car"`, and a price available for purchase.
- **Inputs**:
  - **user**: `"testuser@example.com"`
  - **orderItems**: `[{ id: 1, name: "Toy Car", quantity: 2, total: 30 }]`
  - **totalAmount**: `30`
- **Expected Output**:
  - **Status**: `201`
  - **Message**: `"Order placed successfully!"`

Test Case: Create Order with Missing Fields

- **Description**: Tests the response when creating an order without required fields.
- **Pre-requisites**:
  - No user registration or order required.

- **Inputs**:
  - **orderItems**: `[]` (empty array)
  - **totalAmount**: `0`
- **Expected Output**:
  - **Status**: `400`
  - **Message**: `"Failed to place order"`

Test Case: Fetch User Orders Successfully

- **Description**: Tests fetching orders for a specific user.
- **Pre-requisites**:
  - User with email `"testuser@example.com"` must have placed at least one order.
- **Inputs**:
  - **userEmail**: `"testuser@example.com"`
- **Expected Output**:
  - **Status**: `200`
  - **Message**: Array of orders for the user (e.g., `[{ orderId: "1", item: "Toy Car", quantity: 2, total: 30 }]`)

Test Case: Fetch Orders for Non-Existing User

- **Description**: Tests the response when fetching orders for a user that does not exist.
- **Pre-requisites**:
  - No user with email `"nonexistentuser@example.com"` in the system.
- **Inputs**:
  - **userEmail**: `"nonexistentuser@example.com"`
- **Expected Output**:
  - **Status**: `404`
  - **Message**: `"No orders found."`

Test Case: Return Item Successfully

- **Description**: Tests returning an item from an order.
- **Pre-requisites**:
  - A valid order with at least one item, e.g., `orderId: "validOrderId"`, and item `itemId: "validItemId"`.
- **Inputs**:

    - **orderId**: `"validOrderId"`
    - **itemId**: `"validItemId"`
- **Expected Output**:
    - **Status**: `200`
    - **Message**: `"Item returned successfully."`

Test Case: Return Non-Existent Item

- **Description**: Tests the response when attempting to return an item that does not exist in the order.
- **Pre-requisites**:
    - A valid order with at least one item, e.g., `orderId: "validOrderId"`, and a valid item `itemId: "validItemId"`.
- **Inputs**:
    - **orderId**: `"validOrderId"`
    - **itemId**: `"invalidItemId"`
- **Expected Output**:
    - **Status**: `404`
    - **Message**: `"Item not found in order."`

Unit Test Cases for User Management

Test Case: Register User Successfully

- **Description**: Tests successful registration of a new user.
- **Pre-requisites**:
    - No existing user with the email `"testuser@example.com"`.
- **Inputs**:
    - **username**: `"testuser"`
    - **email**: `"testuser@example.com"`
    - **password**: `"password123"`
- **Expected Output**:
    - **Status**: `201`
    - **Message**: `"User registered successfully!"`

Test Case: Register User with Existing Email

- **Description**: Tests the response when trying to register an email that already exists.
- **Pre-requisites**:
  - A user already exists with the email `"existinguser@example.com"`.
- **Inputs**:
  - **username**: `"testuser"`
  - **email**: `"existinguser@example.com"`
  - **password**: `"password123"`
- **Expected Output**:
  - **Status**: `400`
  - **Message**: `"User already exists."`

Test Case: Login User Successfully

- **Description**: Tests successful login of a registered user.
- **Pre-requisites**:
  - User must be registered with the email `"testuser@example.com"` and password `"password123"`.
- **Inputs**:
  - **email**: `"testuser@example.com"`
  - **password**: `"password123"`
- **Expected Output**:
  - **Status**: `200`
  - **Message**: Contains `email` and a valid JWT token (e.g., `"jwtToken": "valid_token_here"`)

Test Case: Login User with Invalid Credentials

- **Description**: Tests the response when attempting to log in with invalid credentials.
- **Pre-requisites**:
  - User must be registered with the email `"testuser@example.com"` and password `"password123"`.
- **Inputs**:
  - **email**: `"testuser@example.com"`
  - **password**: `"wrongpassword"`
- **Expected Output**:
  - **Status**: `400`
  - **Message**: `"Invalid credentials."`

Test Case: Login Non-Existing User

- **Description**: Tests the response when trying to log in with an email that is not registered.
- **Pre-requisites**:
  - No user with email `"nonexistinguser@example.com"` in the system.
- **Inputs**:
  - **email**: `"nonexistinguser@example.com"`
  - **password**: `"password123"`
- **Expected Output**:
  - **Status**: `400`
  - **Message**: `"Invalid credentials."`

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.

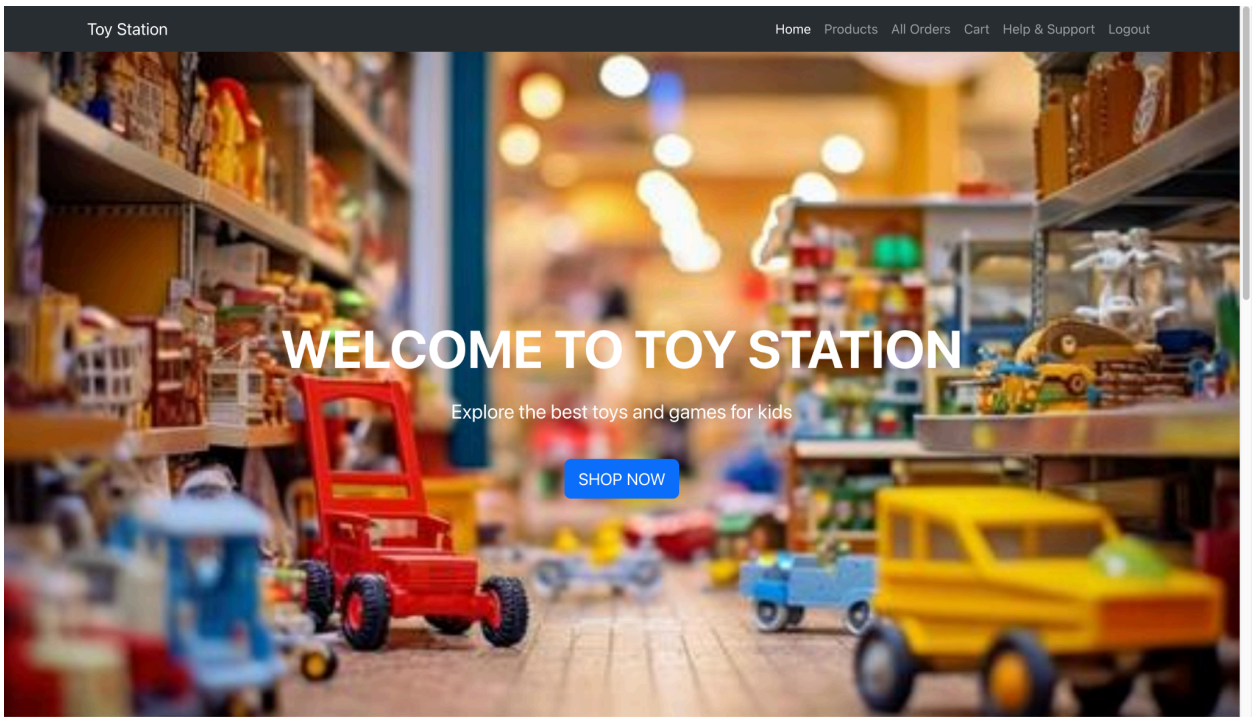
Table of Contents

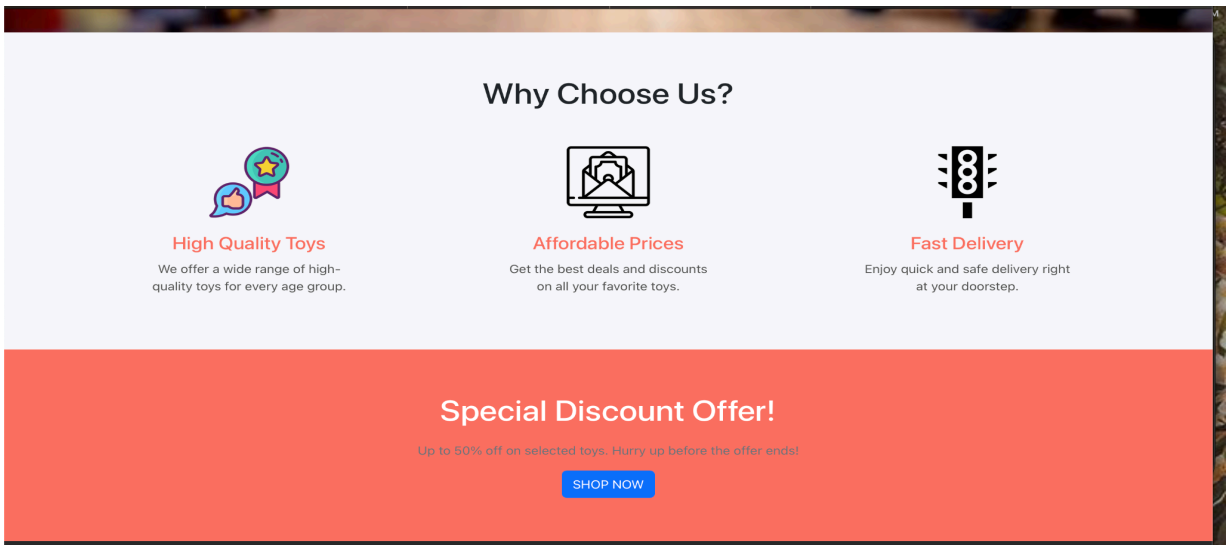
Getting Started

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

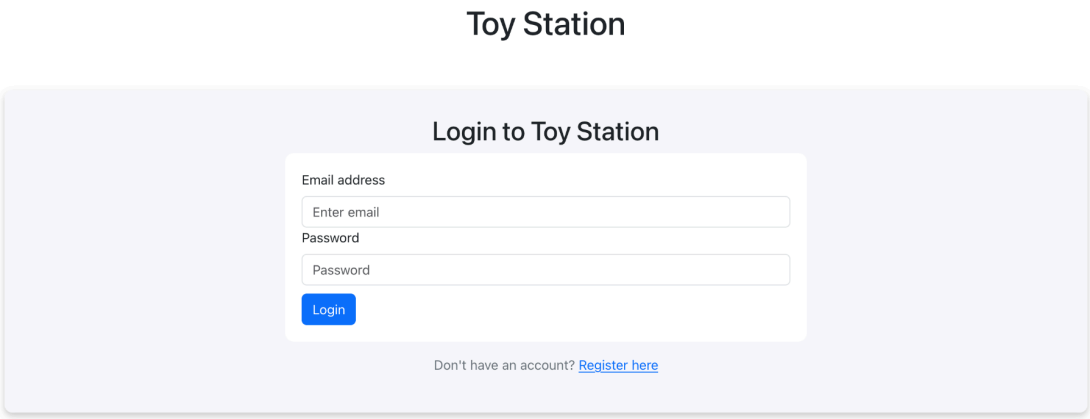
1. **Navigate to the Login Page**: Click on the "Login" link at the top right corner of the homepage.

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.

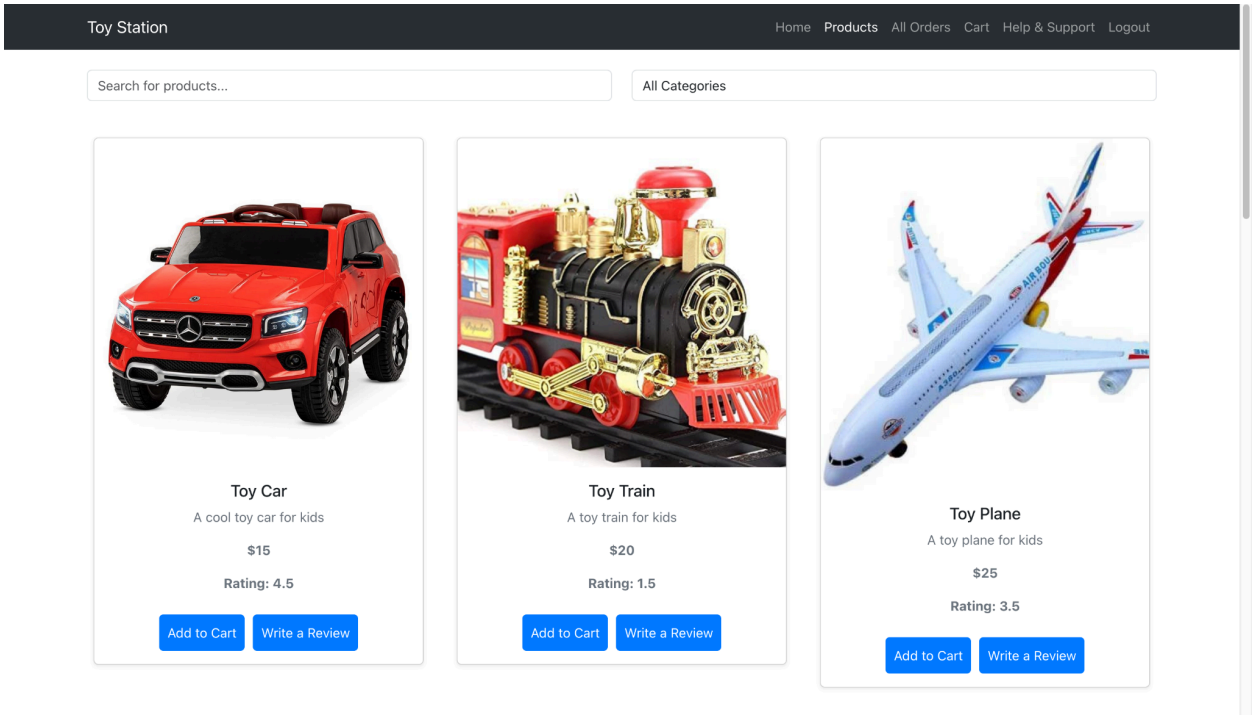
3. **Add to Cart**:

    - 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.

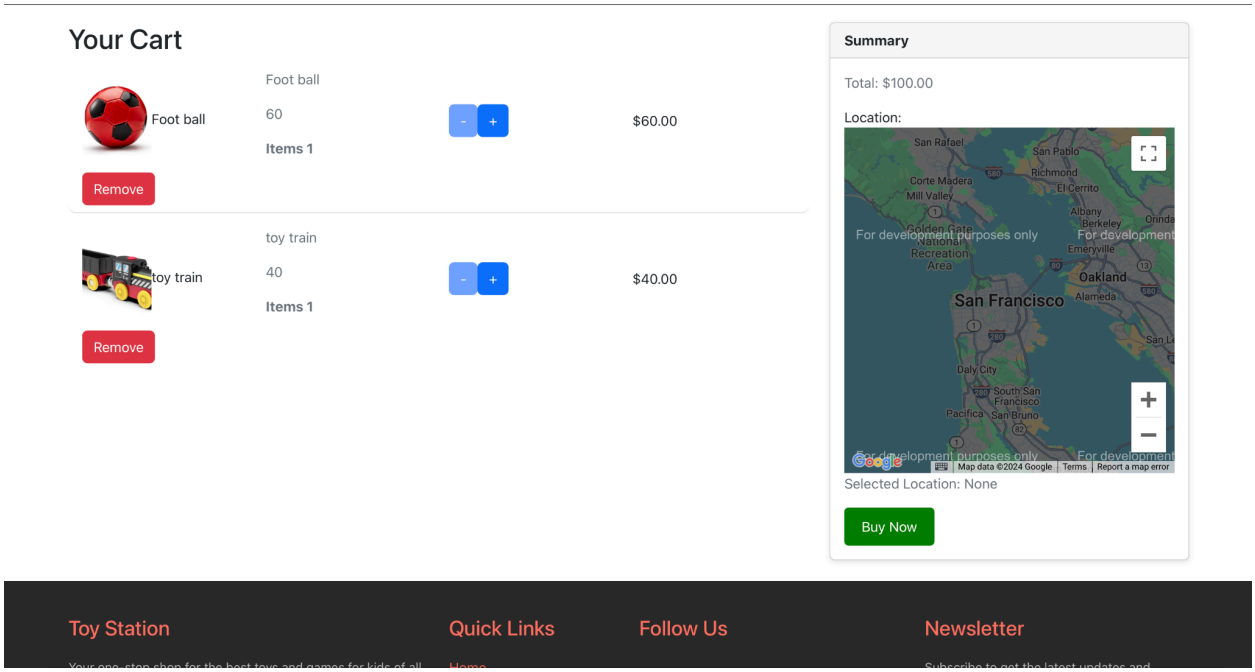
=



Google Maps Integration

The user manual effectively explains how to use the Google Maps integration for delivery directions. It clearly outlines the process from clicking the "Get Directions" button to following the route in Google Maps. The manual highlights key benefits such as efficiency, accuracy, and ease of use, while ensuring users are informed with loading indicators and error handling.

Suggestions for improvement: adding visuals (e.g., screenshots) and a troubleshooting section for common issues would further enhance clarity. Overall, it’s a clear and user-friendly guide.

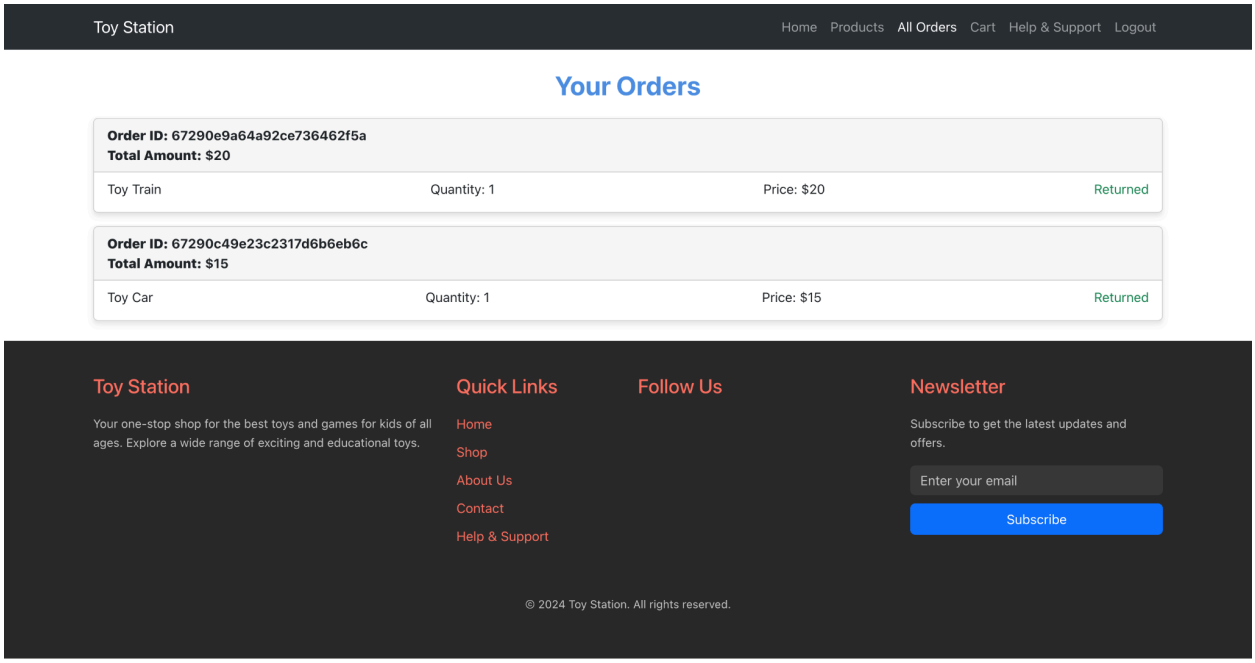


Viewing Orders

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

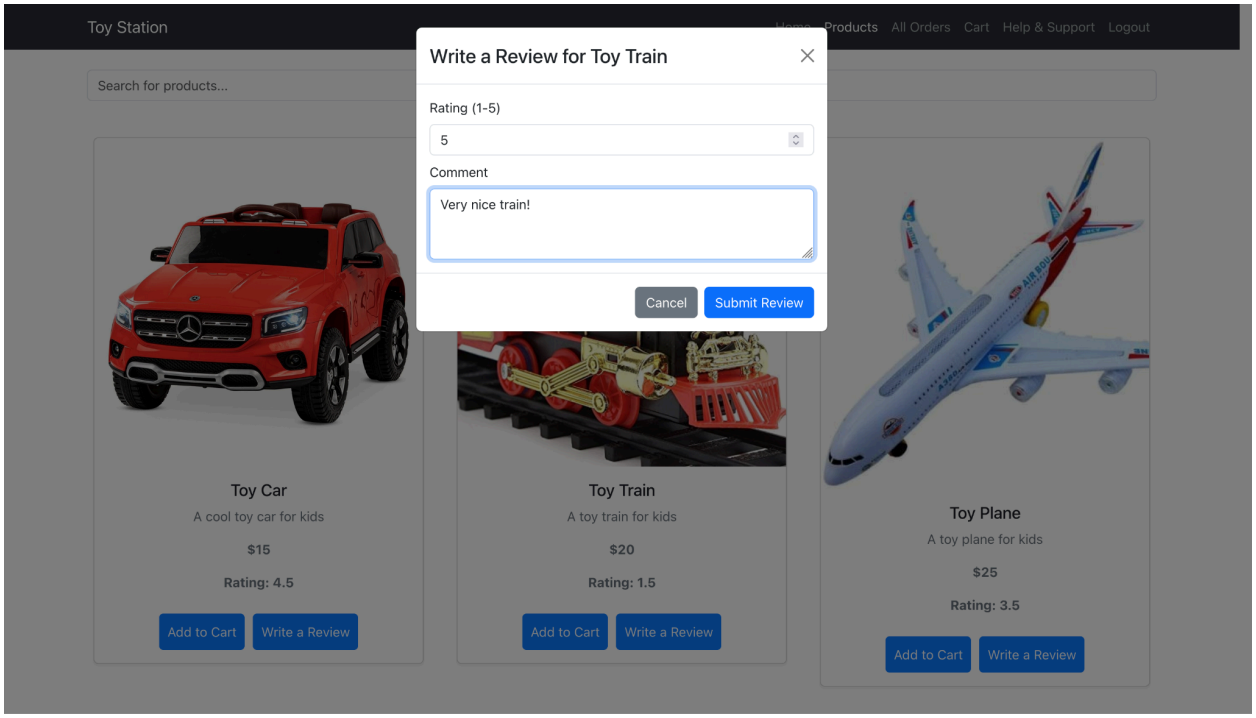
2. **Order List**: You will see a list of your past orders, including details such as order ID, total amount, and order items.

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



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**: tejanagendrasirigineedi@my.unt
- **Phone**: +1 940 597 3113

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

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.

Feature Ending Summary:

Successfully Implemented Features

User Authentication:

Secure registration and login functionality ensures only authorized access to the platform.

Provides a seamless onboarding experience with real-time validation and error handling.

Order Management System:

Allows users to place, view, and manage their orders.

Includes features for tracking order status and processing item returns.

Payment Gateway Integration (Stripe):

Enables secure and seamless transactions using Stripe's API.

Incorporates real-time error handling and feedback for a user-friendly checkout experience.

Google Maps Integration for Delivery Directions:

Provides accurate and efficient routing for delivery personnel.

Ensures cross-platform compatibility with desktop and mobile devices.

Feedback and Documentation Enhancements:

Comprehensive documentation and refactoring based on code inspection feedback.

Improved modularity and readability for future scalability.

Known Limitations:

Recommendation Engine

Issue: Lack of sufficient user data and backend support for implementing advanced recommendation algorithms.

Future Scope: Integration of machine learning or third-party APIs to enhance personalization.

Customer Support System

Issue: Limited backend infrastructure and absence of a notification system.

Future Scope: Incorporate third-party tools for scalable customer support.

Product Reviews and Ratings

Issue: Dynamic UI/UX and backend moderation capabilities were deprioritized.

Future Scope: Enhance engagement with user-generated content moderation.

Returns and Refunds

Issue: Complex backend workflows for validating return requests and payment integration.

Future Scope: Streamline logistics and payment coordination for seamless post-purchase management.

Live Chat Implementation

Issue: Time and resource constraints prevented the integration of a robust real-time messaging system.

Future Scope: Explore solutions like Firebase or Socket.IO for future iterations.

Feedback Received During Code Inspection

During the code inspection session, several constructive feedback points were raised by peers and mentors. Key areas of focus included code readability, modularity, and documentation.

1. **Code Readability**: Some team members suggested enhancing the readability of the code by using more descriptive variable names and adhering to a consistent coding style. In response, we refactored several functions and components to improve clarity and added comments to explain complex logic.

2. **Modularity**: The inspection highlighted the need for better modularization of components to promote reusability and maintainability. We took action by breaking down larger components into smaller, reusable ones, ensuring that each component has a single responsibility.

3. **Documentation**: It was noted that documentation for certain functions and classes was lacking. To address this, we prioritized writing comprehensive documentation for all major components and functions, providing clearer guidelines for future developers.

These actions have contributed to a more robust codebase and facilitated easier collaboration among team members.

Reflection on Accomplishments

Throughout this project, we have successfully implemented an Order Management System that meets the specified requirements. Key accomplishments include:

- **User Authentication**: The implementation of user registration and login functionality allows for secure access to the system.
- **Order Processing**: Users can place orders, view their order history, and return items seamlessly.
- **Product Reviews**: We added a feature that allows users to leave reviews for products, enhancing user engagement and providing valuable feedback.

What Went Well

- **Collaboration**: The team worked well together, effectively communicating and supporting one another throughout the development process.

- **Adherence to Deadlines**: We successfully met all major milestones, ensuring that the project remained on schedule.

Areas for Improvement

- **Testing Coverage**: While we implemented unit tests, there is room to expand our test coverage to include more edge cases and integration tests.
- **Feedback Implementation**: Some feedback from the code inspection was implemented post-deadline. Establishing a more proactive approach to incorporate feedback throughout the development cycle could enhance future projects.

Member Name	Role	Task assigned	Report Description	Overall Contribution
Vaishnavi Shastrula	Project ManagementLead	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, versioncontrol, environment setup, demo preparation	Contributed to the system architecture and risk management sections	16%

Sai Sowjanya Edupuganti	Design Lead	UI/UXdesign, 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%
Gayatri Vutla	Implementation Lead for back end	Backend development, server-side logic,database management	Contributed to back end and database design sections	12%
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%

Member Contribution Table **Total Contribution: 100%**

Meeting Minutes:

Meeting 1:

Date: November 12th, 2024

Time: 3:00 PM – 4:00 PM

Location: Google Meet

Attendees: All team members

Agenda:

- Start of Deliverable 5 requirements.
- Assign work regarding final development:
- Fix errors and improve features.
- Complete the frontend and backend integration.
- Talk about the user manual and documentation updates.

Action Items:

- Assign team members to work on frontend enhancements and bug fixes.
- Write the final report, concentrating on the most updates.
- Examine the database queries and backend API performance.

Meeting 2:

Date: November 18th, 2024

Time: 3:00 PM – 4:00 PM

Location: Teams

Attendees: All team members

Agenda:

Development progress:

- Frontend improvements to improve user experience.
- Backend upgrades for reliable performance and integration.
- Evaluate the testing process:
- Individual modules unit tests.
- System integration tests.
- Talk about the demo's presentation setup.

Action Items:

- Finalize the backend and frontend features by the next meeting.
- Finish the newly integrated modules testing
- Make an outline for the demonstration.

Meeting 3:

Date: November 24th, 2024

Time: 4:00 PM – 5:00 PM

Location: Google Meet

Attendees: All team members

Agenda:

- Perform system-wide and integration testing.
- Resolve serious issues found during testing.
- Examine and finalize the updates related to deployment guide and user manual.

Action Items:

- Update all documentation after final test results.
- Make sure bug are fixed and tested .
- Create a demo script that highlights all of the functionality.

Meeting 4:

Date: November 28th, 2024

Time: 3:00 PM – 4:00 PM

Location: Teams

Attendees: All team members

Agenda:

Complete the project deliverables:

- Make sure that GitHub has the most recent versions of the source code, test cases, and outcomes.
- Finalize the report, deployment guide, and user manual.

Setting up for demo:

- Practice your demo presentation.
- Verify functionality on various devices and under various conditions.

Action Items:

- Resolve any small errors that were found during the demo rehearsal.
- Upload all the updated documents into the project repository.

Meeting 5:

Date: December 2nd, 2024

Time: 3:00 PM – 4:00 PM

Location: Teams

Attendees: All team members

Agenda:

Final preparations for submission:

· Verify that all the reports, documents, source code have all been finished and uploaded to GitHub.

· Talk about any last-minute adjustments and make sure the presentation video is good.

Action Items:

· Deliverable 5 must be submitted by December 2, 2024.

· Make sure that everything is uploaded to Canvas, including the presentation video.

· Get ready for the last review and comments.