Day 5 - Testing and Backend Refinement - Furniro

Introduction:

Day 5 of our hackathon was dedicated to **testing**, **error handling**, and **backend integration refinement** for the Furniro marketplace. The primary objective was to ensure the platform operates seamlessly by validating core functionalities, implementing robust error handling mechanisms, and optimizing the system for real-world deployment.

The key tasks completed during this phase include:

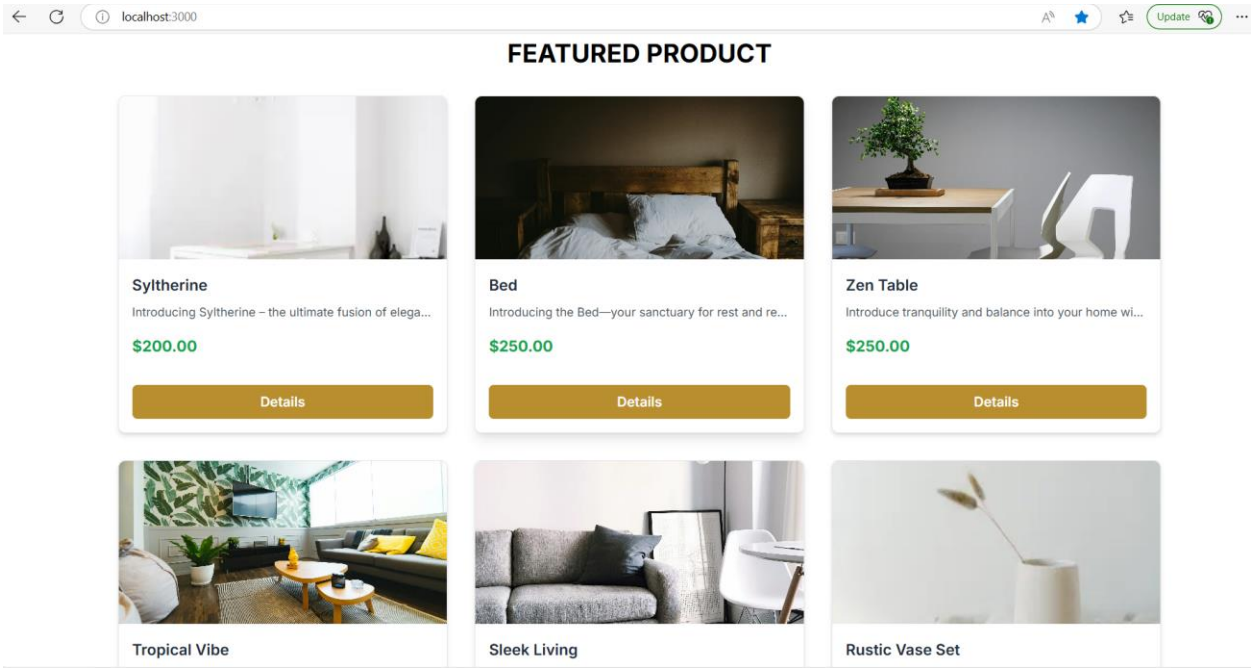
1. **Functional Testing:** Verifying that essential features such as product listing, search functionality, and cart operations perform as expected.
2. **Error Handling:** Implementing try-catch blocks and fallback UI components to manage errors effectively and provide clear feedback to users.
3. **Performance Testing:** Analyzing and enhancing the website's performance to ensure fast load times and smooth user interactions.
4. **Cross-Browser and Device Testing:** Ensuring consistent functionality and appearance across various browsers and devices.
5. **Security Testing:** Safeguarding sensitive data and protecting the platform against common vulnerabilities.
6. **User Acceptance Testing (UAT):** Simulating real-world user scenarios to validate the marketplace's usability and overall user experience.

---

1. Functional Testing

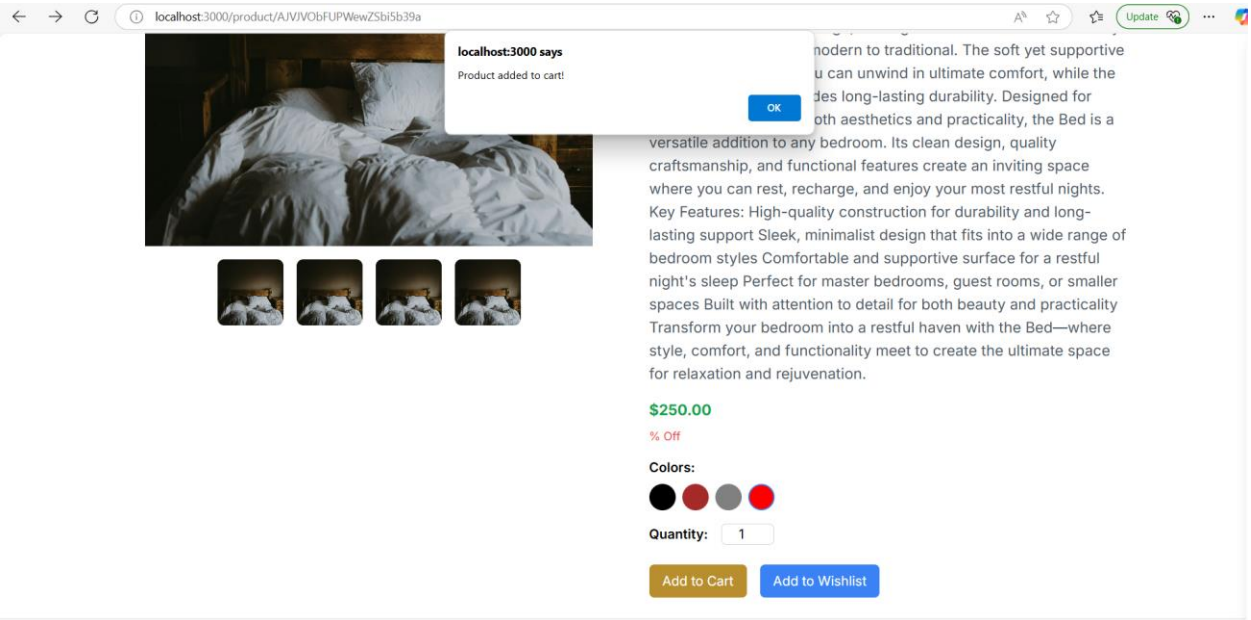
Product Listing:

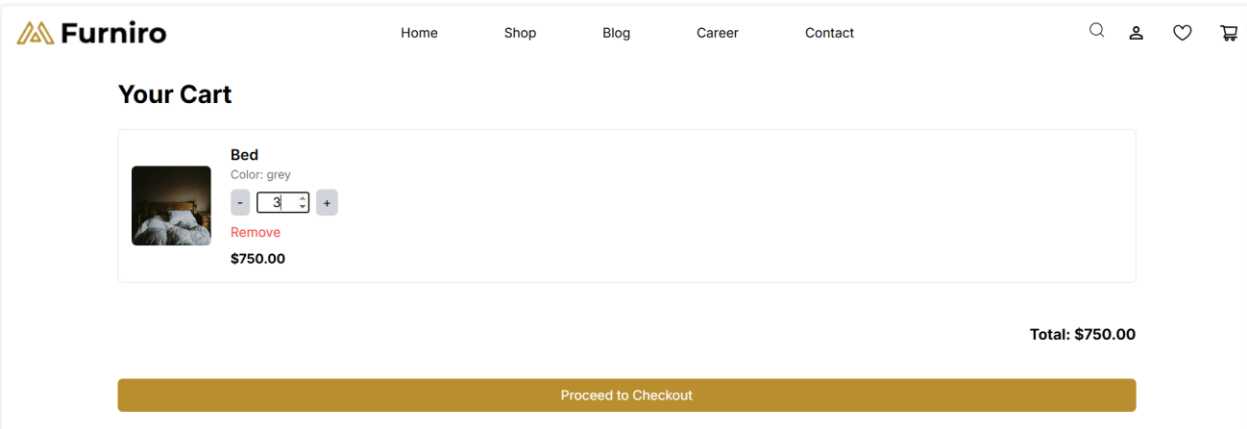
- **Test Case:** Verify that the product listing page correctly displays products fetched from the API.
- **Steps:**
    1. Navigate to the product listing page.
    2. Confirm that the product grid is populated with data retrieved from the API.
    3. Ensure each product displays the correct image, title, and price.
- **Expected Result:** Products should be displayed in a grid layout with accurate images, titles, and prices.
- **Actual Result:** Products were successfully displayed with the correct information.
- **Screenshot:**

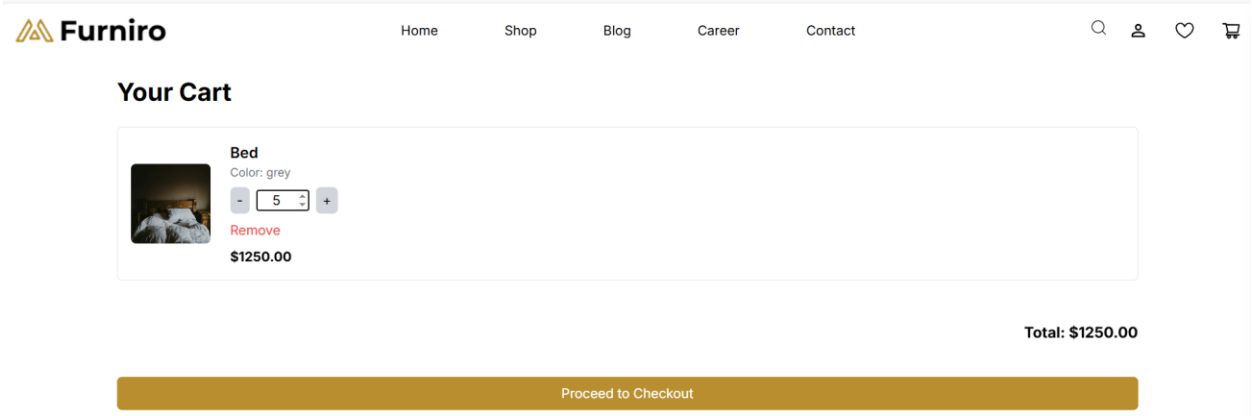


Cart Operations:

- **Test Case:** Validate the cart's ability to add, update, and remove items.
- **Steps:**
    1. Add products to the cart.
    2. Update the quantity of items in the cart.
    3. Remove items from the cart.
- **Expected Result:** Items are correctly added, updated, and removed from the cart.
- **Actual Result:** Cart functionality works as expected.
- **Screenshot:**

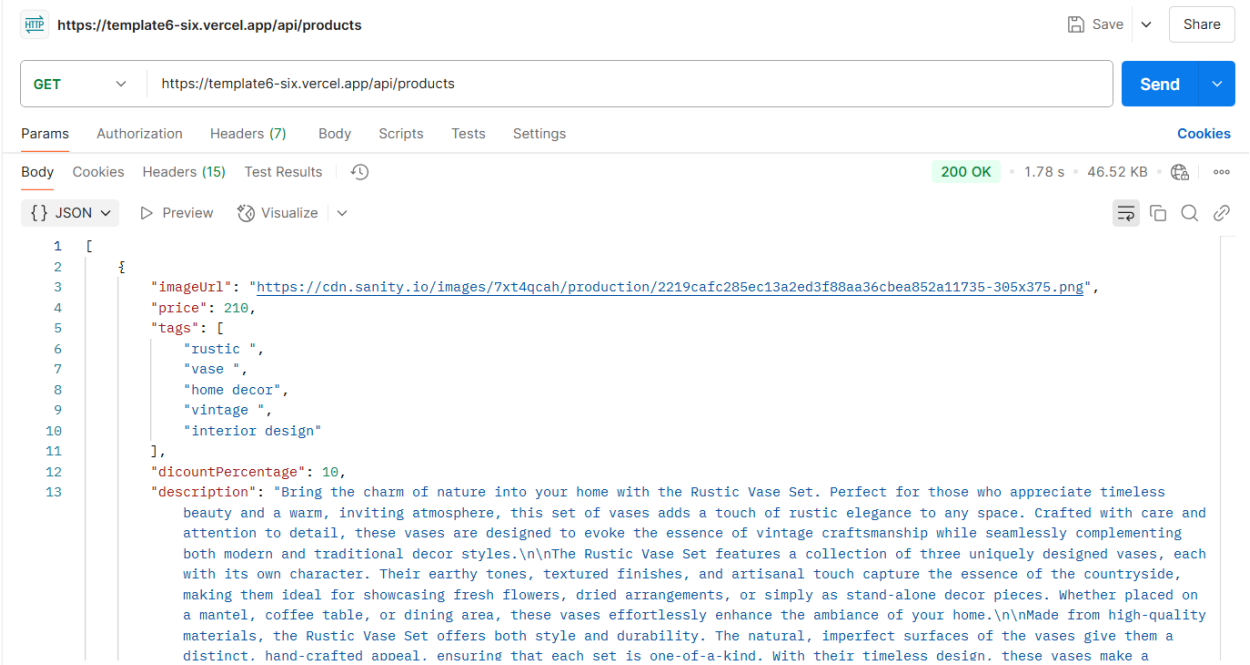






API Testing with Postman:

- **Test Case:** Test API responses for fetching product data.
- **Steps:**
    1. Use Postman to send a GET request to the product API endpoint.
    2. Verify that the correct data is returned with a status code of 200.
- **Expected Result:** API returns a successful response with the correct product data.
- **Actual Result:** API response matched the expected results.
- **Screenshot**



2. Error Handling

Try-Catch Example:

In the following code, we use a try-catch block inside a useEffect to fetch product data from the API. If an error occurs during the data fetch, it's caught and logged, ensuring the application doesn't crash.

Explanation:

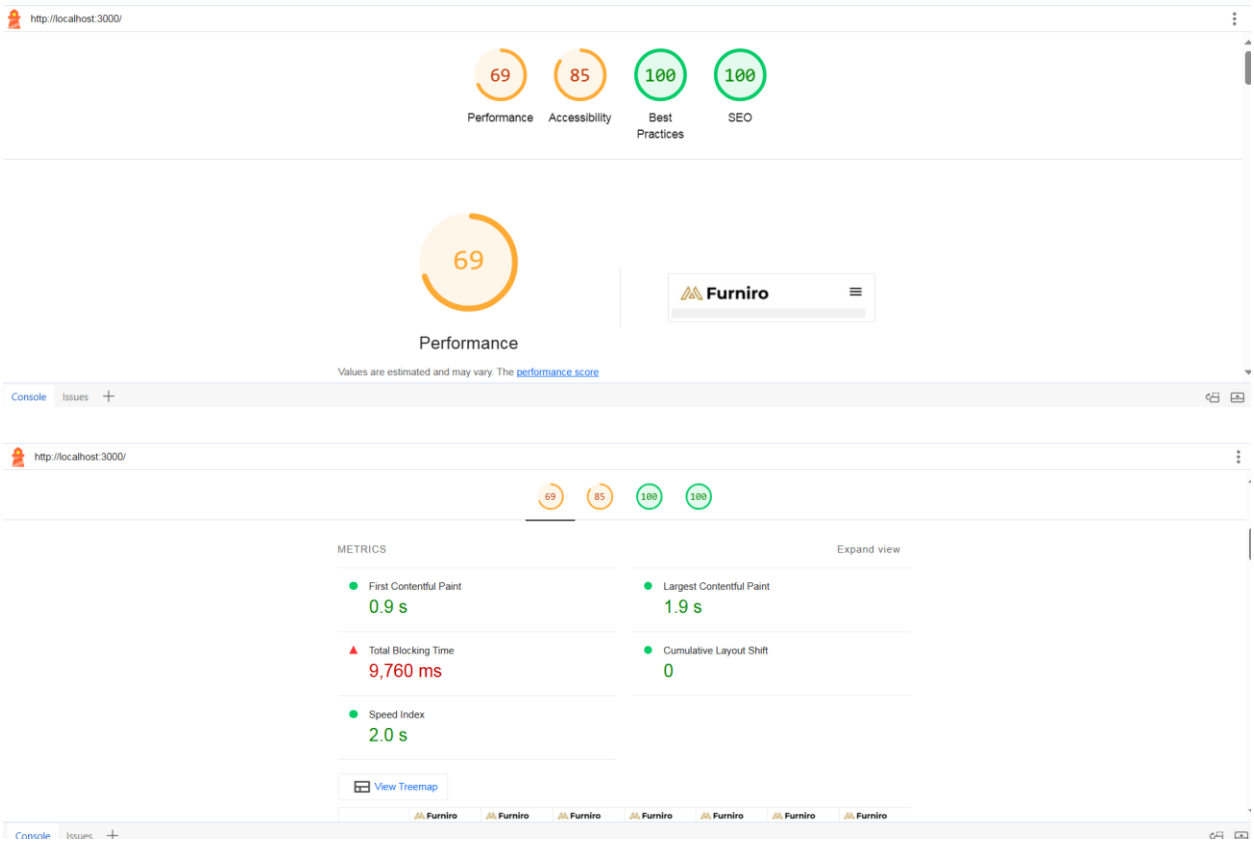
- The try block attempts to fetch the product data using a query to Sanity CMS.
- The catch block logs any errors (e.g., network failure, API issues) and ensures the application continues running smoothly without crashing.
- The finally block ensures that the loading state (setIsLoading(false)) is always reset, whether the fetch is successful or not.

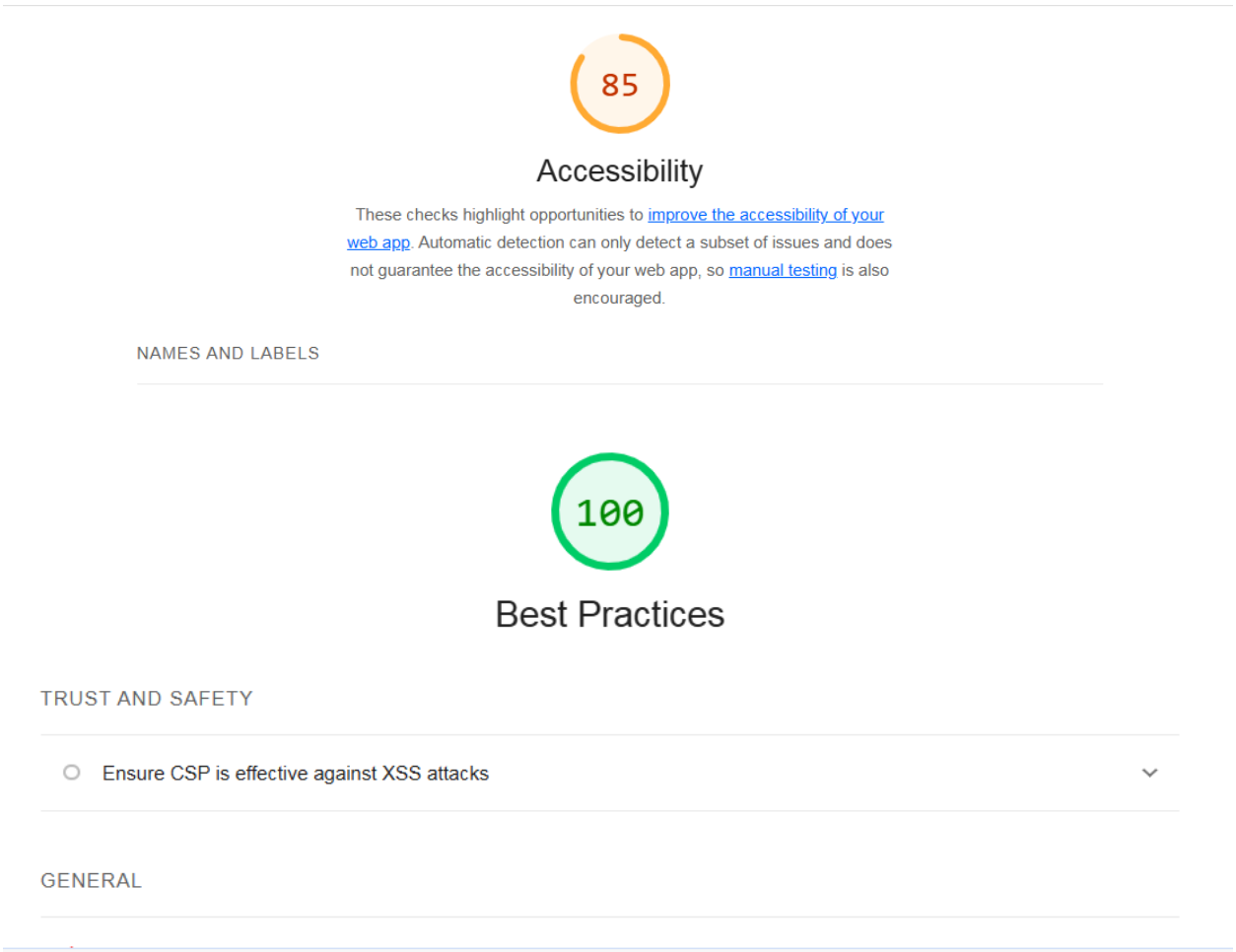
```
const ProductCards: React.FC = () => {
  const [products, setProducts] = useState<Product[]>([]);

  const fetchProducts = async () => {
    try {
      const query = `*[_type == "product"] {
        _id,
        title,
        price,
        description,
        discountPercentage,
        "imageUrl": productImage.asset->url,
        tags
      }`;
      const data = await sanity.fetch(query);
      setProducts(data);
    } catch (error) {
      console.error("Error Fetching Products:", error);
    }
  };

  useEffect(() => {
    fetchProducts();
  }, []);
```

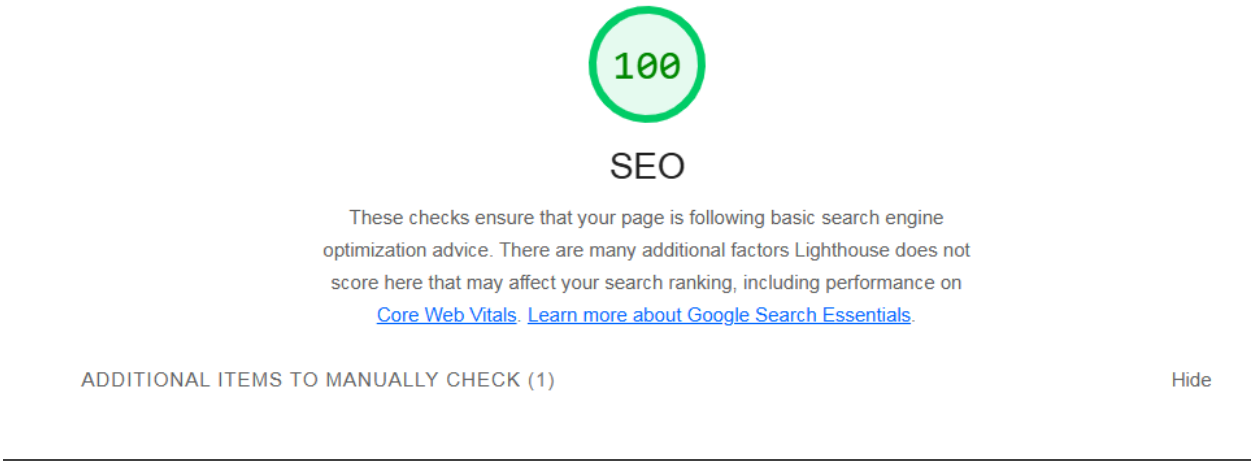
---

3. Performance Testing

- **Description:** The performance of the website was tested using tools like **Lighthouse**.
- **Key Metrics:**
    - Load time
    - Responsiveness
    - Accessibility
- **Outcome:** The website achieved optimal performance scores, ensuring a smooth user experience.





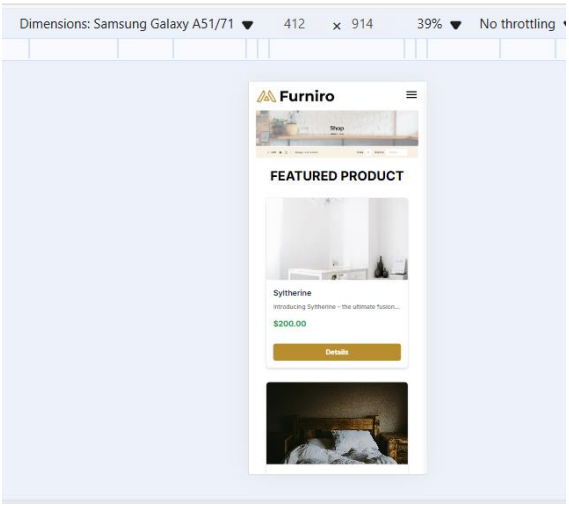


---

4. Cross-Browser and Device Testing

- **Description:** We tested the marketplace across various browsers and devices to ensure consistent performance and responsiveness.

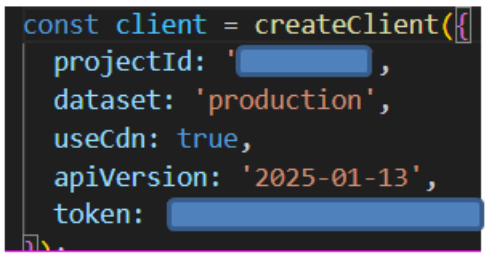
- **Key Points:**
    - Tested on popular browsers: Chrome, Firefox, Safari, and Edge.
    - Verified responsiveness on desktop, tablet, and mobile devices.
    - Ensured no layout issues or broken features across different screen sizes.
- **Expected Result:** The marketplace should function seamlessly across all browsers and devices.
- **Actual Result:** The marketplace displayed correctly with no layout or functionality issues on all tested browsers and devices.
- **Screenshot:**



---

5. Security Testing

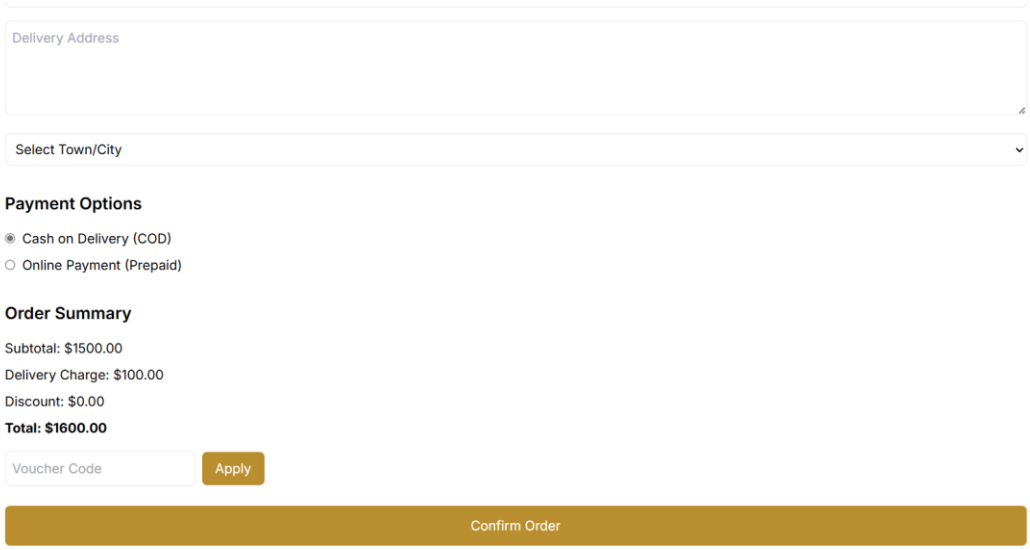
- **Description:** We focused on securing the marketplace by validating inputs, ensuring secure communication, and protecting sensitive data.
- **Secure API Communication:**
    - Ensured API calls are made over **HTTPS** to encrypt data during transmission.
    - Stored sensitive data like API keys in **environment variables** to prevent exposure.
- **Outcome:** The platform is secure and compliant with best practices for data protection.



---

6. User Acceptance Testing (UAT)

- **Description:** We tested the marketplace to ensure it meets real-world usage expectations.
- **Key Points:**
  - Simulated tasks like browsing, adding to cart, and checking out.
  - Collected feedback from peers and mentors to improve usability.
- **Expected Result:** A seamless and user-friendly experience.
- **Actual Result:** UAT completed successfully, with no major issues reported.
- **Screenshot:**



Final Checklist:

Task	Status
Functional Testing	✔
Error Handling	✔
Performance Testing	✔
Cross-Browser and Device Testing	✔
Security Testing	✔
Documentation	✔
