🚀 Day 5 - Testing, Error Handling, and Backend Integration Refinement

Objective:

Day 5 emphasizes preparing your marketplace for real-world deployment through rigorous testing, robust error handling, and seamless backend integration. It's all about optimizing performance, ensuring reliability, and delivering a user-friendly experience across all platforms.

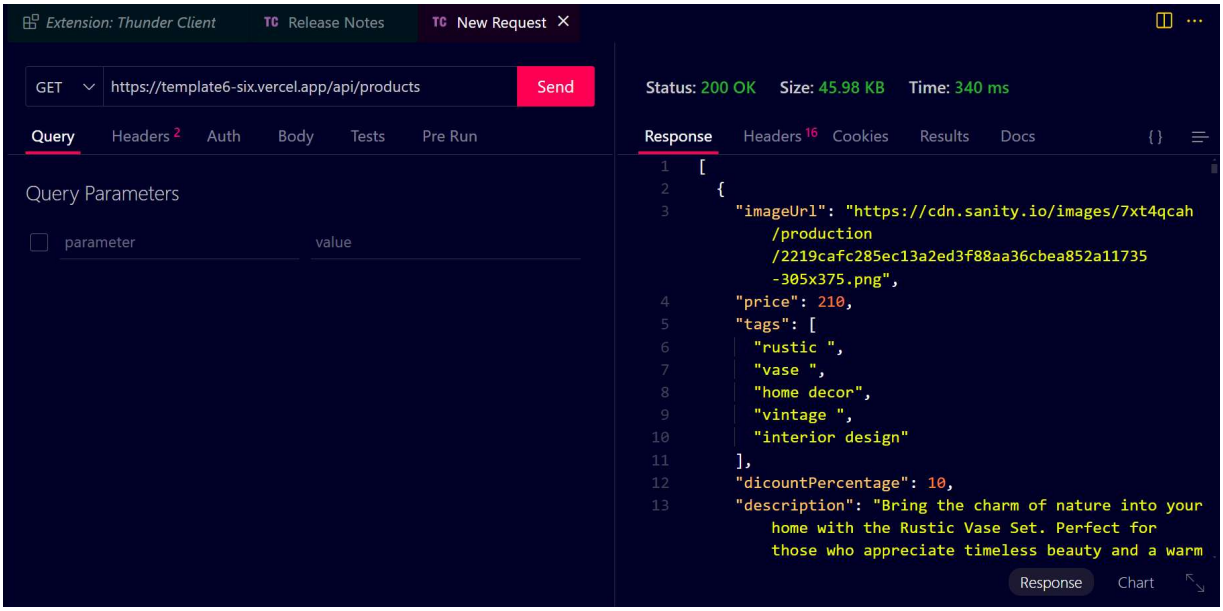
Key Learning Outcomes:

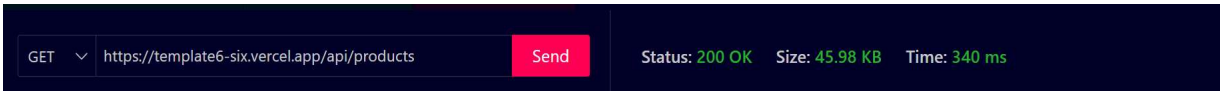
1️⃣ Comprehensive testing: Functional, non-functional, UAT, and security testing.
2️⃣ Implementing robust error handling with clear, user-friendly fallback messages.
3️⃣ Marketplace optimization for speed, responsiveness, and performance metrics.
4️⃣ Ensuring cross-browser and device compatibility.
5️⃣ Producing professional testing documentation, including CSV-based reports.
6️⃣ Gracefully handling API errors with fallback UI elements and detailed logs.

Key Areas of Focus

🔍 Functional Testing:

- Validate core functionalities like product listing, cart operations, and user profile management.
- Use tools like **Postman**, **React Testing Library**, and **Cypress** for API and component testing.





⚠️ Error Handling:

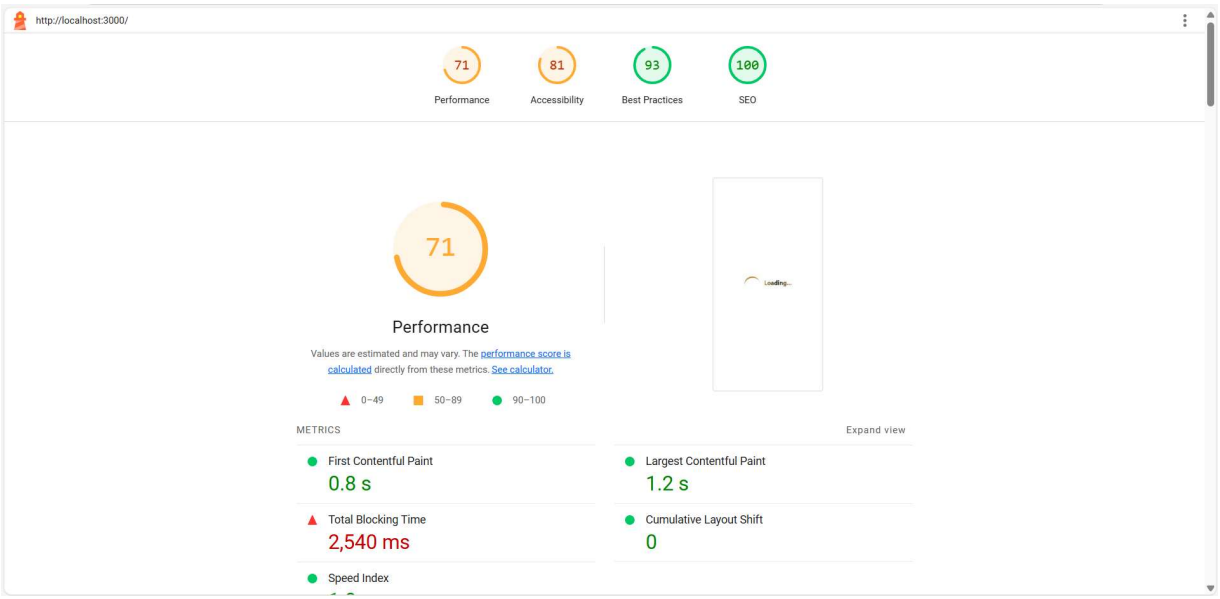
- Use `try-catch` blocks to manage API errors gracefully.

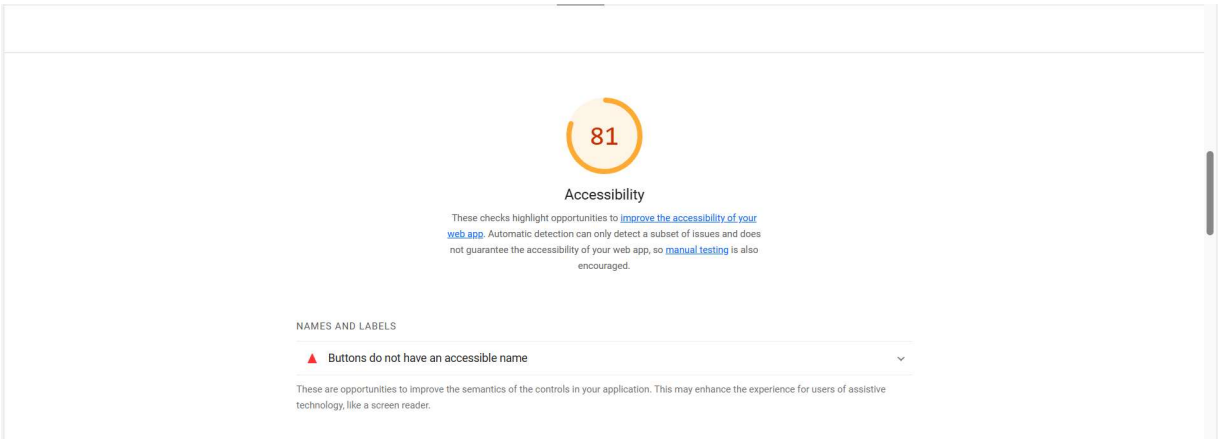
```
import { defineQuery } from "next-sanity";

// Fetch All Products

export const allproduct = async () => {
  try {
    const products = await defineQuery(`
      *[_type == "product"]{
        _id,
        title,
        description,
        "productImage": productImage.asset->url,
        price,
        tags,
        discountPercentage,
        isNew
      }
    `);
    return products;
  } catch (error) {
    console.error("Error fetching products:", error);
    throw new Error("Failed to fetch products. Please try again later.");
  }
};
```

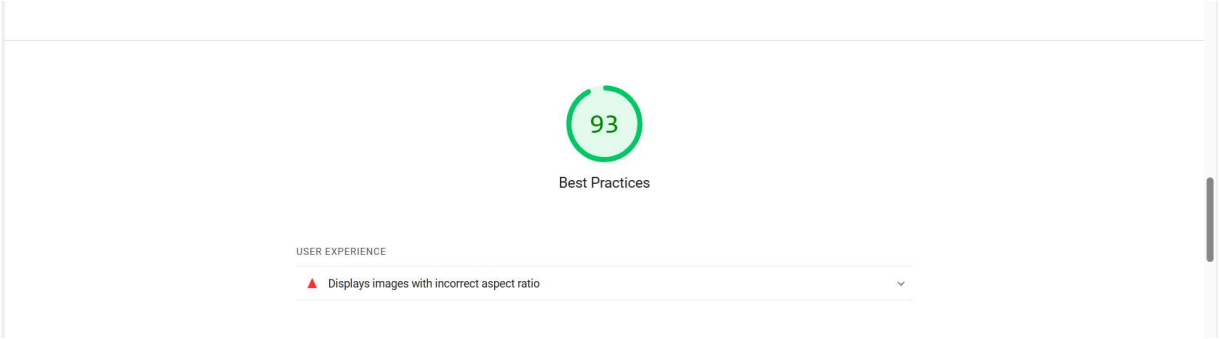
- Display fallback messages like "No items found" for empty product lists.

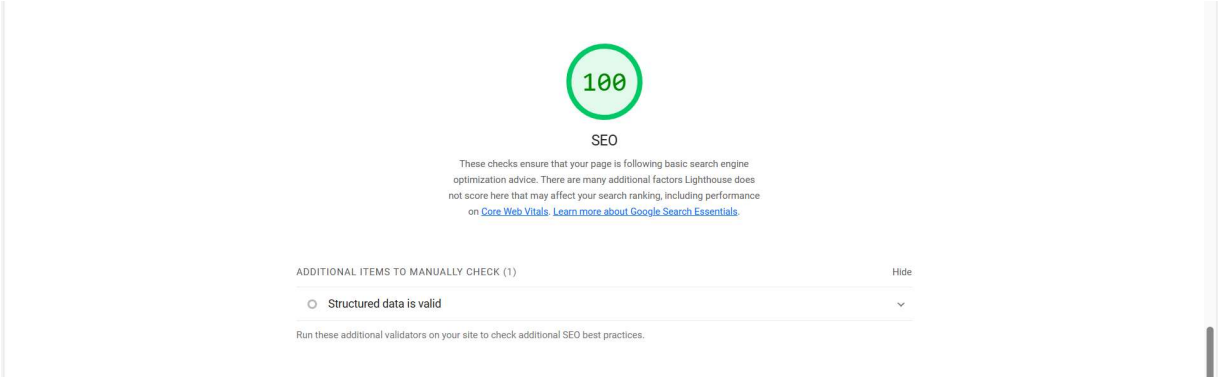
⚡ Performance Testing:

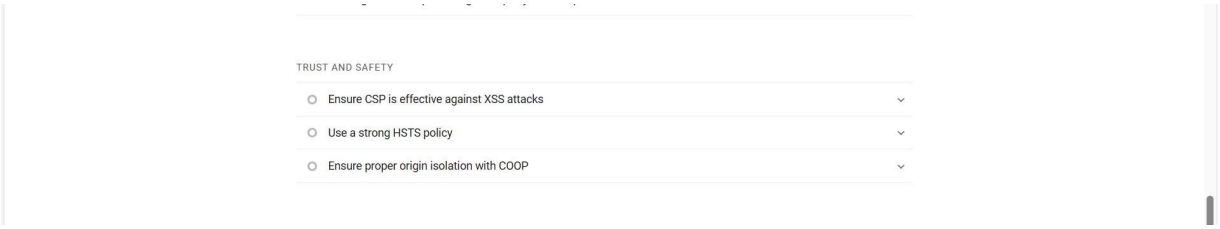
- Optimize assets (compress images, lazy loading).
- Analyze performance with **Lighthouse**, **GTmetrix**, and similar tools.


http://localhost:3000/
71
Performance
81
Accessibility
93
Best Practices
100
SEO
71
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
0–49
50–89
90–100
METRICS
Expand view
First Contentful Paint
0.8 s
Largest Contentful Paint
1.2 s
Total Blocking Time
2,540 ms
Cumulative Layout Shift
0
Speed Index


81
Accessibility
These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.
NAMES AND LABELS
Buttons do not have an accessible name
These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.


93
Best Practices
USER EXPERIENCE
Displays images with incorrect aspect ratio





- Target load times under **2 seconds**.

🌐 **Cross-Browser & Device Testing:**

- Test on Chrome, Firefox, Safari, and Edge using tools like **BrowserStack**.
- Ensure responsive design across devices (desktop, tablet, mobile).

🔒 **Security Testing:**

- Validate inputs to prevent SQL injection or XSS attacks.
- Secure API calls using **HTTPS** and environment variables.
- Use tools like **OWASP ZAP** for vulnerability scans.

👥 **User Acceptance Testing (UAT):**

- Simulate real-world scenarios to validate intuitive workflows.
- Gather feedback for improvements.

Steps for Implementation

✅ **Step 1: Functional Testing**

- Test features like product listing, filters, and cart operations.

- Simulate user actions and validate outcomes.

✅ **Step 2: Error Handling**

- Handle API errors using `try-catch` and display meaningful error messages.

```
import { defineQuery } from "next-sanity";

// Fetch All Products

export const allproduct = async () => {
  try {
    const products = await defineQuery(`
      *[_type == "product"]{
        _id,
        title,
        description,
        "productImage": productImage.asset->url,
        price,
        tags,
        discountPercentage,
        isNew
      }
    `);
    return products;
  } catch (error) {
    console.error("Error fetching products:", error);
    throw new Error("Failed to fetch products. Please try again later.");
  }
};
```

✅ **Step 3: Performance Optimization**

- Compress assets, implement caching, and reduce unused CSS/JS.
- Conduct load time analysis and implement fixes.

✅ **Step 4: Cross-Browser and Device Testing**

- Ensure consistent rendering and functionality across major browsers and devices.

✅ **Step 5: Security Testing**

- Validate inputs and secure sensitive data.

✅ **Step 6: UAT**

- Simulate real-world interactions and gather feedback.

✅ Step 7: Documentation

- Include test results, optimization steps, and security measures.

Expected Output:

- Fully tested, optimized, and responsive marketplace.
- Clear error handling and fallback mechanisms.
- Comprehensive documentation and CSV-based testing reports.

CSV:

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Assigned To	Remarks
TC001	Validate product listing page	Open product page > Verify products	Products displayed correctly	Products displayed correctly	Passed	Low	-	No issues found
TC002	Test API error handling	Disconnect API > Refresh page	Show fallback UI with error message	Error message shown	Passed	Medium	-	Handled gracefully
TC003	Check cart functionality	Add product to cart > Verify cart contents	Cart updates with added product	Cart updates as expected	Passed	High	-	Works as expected
TC004	Ensure responsiveness on mobile	Resize browser window > Check layout	Layout adjusts properly to screen size	Responsive layout working as intended	Passed	Medium	-	Test successful