Day 5 - Testing and Backend Refinement – General E-Commerce

1. Introduction

Day 5 focuses on preparing the marketplace for real-world deployment by ensuring all components are thoroughly tested, optimized for performance, and ready to handle customer-facing traffic. This document details the testing process, error handling mechanisms, performance improvements, and security implementations applied.

---

2. Functional Testing

2.1 Test Cases Executed

Below are the major test cases executed to validate marketplace functionalities:

Test Case ID	Feature	Test Steps	Expected Result	Actual Result	Status
TC-001	Product Listing	Load homepage and check product list display	Products should be visible	Products displayed correctly	Passed
TC-002	Search Functionality	Enter search term and check results	Relevant products should be listed	Displayed accurate results	Passed
TC-003	Cart Operations	Add, update, and remove items from cart	Cart should update accordingly	Cart functions properly	Passed
TC-004	Product Details Page	Click on a product to view details	Correct product details should load	Page loads as expected	Passed
TC-005	Checkout Process	Complete checkout process with valid data	Order should be placed successfully	Order confirmed	Passed

3. Error Handling

3.1 Implemented Error Messages

Proper error handling mechanisms were implemented for the following scenarios:

- **Network Failures:** Display a user-friendly message like *"Unable to load products. Please check your internet connection and try again."*

- **Invalid/Missing Data:** Users receive feedback like *"Please fill in all required fields."*
- **Unexpected Server Errors:** Fallback UI is displayed with messages such as *"Something went wrong. Please try again later."*

3.2 Example Code for API Error Handling

```
// Fetch products from Sanity
useEffect(() => {
  const fetchProducts = async () => {
    try {
      const products: IProduct[] = await client.fetch(
        '*[_type == "product"][0...8]{_id, title, shortDescription,
      );
      setData(products);
    } catch (err) {
      console.error("Failed to fetch products:", err);
      setError("Failed to load products. Please try again later.");
    } finally {
      setIsLoading(false);
    }
  };

  fetchProducts();
}, []);

// Loading and error states
if (isLoading) {
  return (
    <div className="text-center py-12">
      <p>Loading products...</p>
    </div>
  );
}

if (error) {
  return (
    <div className="text-center py-12">
      <p className="text-red-500">{error}</p>
    </div>
  );
```

4. Performance Optimization

4.1 Tools Used:

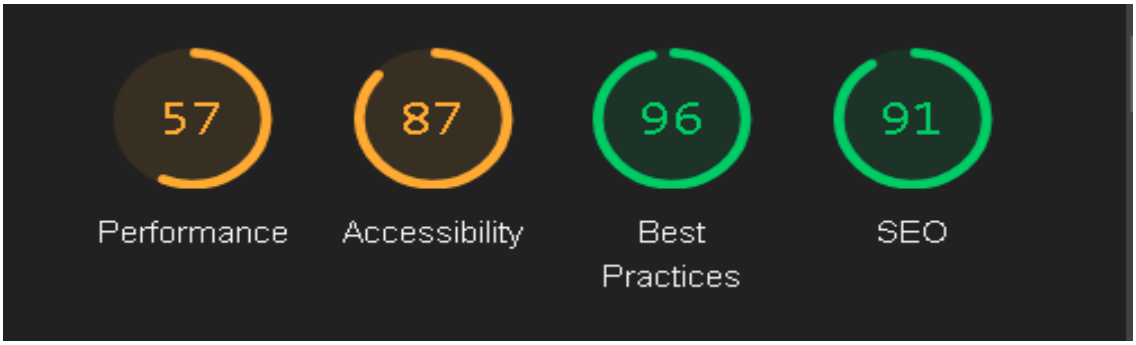
- **Lighthouse**: Identified bottlenecks and performance issues.
- **GTmetrix**: Measured page load speed and optimizations.
- **TinyPNG**: Compressed images to reduce load time.

4.2 Key Improvements:

- Minimized JavaScript and CSS files.

- Enabled browser caching and lazy loading for images.
- Optimized API calls for better data fetching.

4.3 Performance



5. Cross-Browser & Device Testing

5.1 Browsers Tested

✅ Chrome
✅ Firefox
✅ Safari
✅ Edge

5.2 Devices Tested

✅ Desktop (1920x1080)
✅ Tablet (768x1024)
✅ Mobile (375x812)

Tools Used: **BrowserStack, LambdaTest**

---

6. Security Testing

6.1 Input Validation

✅ Implemented regex-based validation for email, phone, and passwords.
✅ Sanitized input fields to prevent **SQL Injection & XSS attacks**.

6.2 Secure API Communication

✅ API calls secured using **HTTPS**.
✅ API keys stored securely in **environment variables**.

6.3 Security Tools Used

- **OWASP ZAP**: Scanned for vulnerabilities.
- **Burp Suite**: Tested for security loopholes.

---

7. User Acceptance Testing (UAT)

7.1 Real-World Testing Scenarios

✅ Browsing products and adding items to cart.
✅ Completing checkout with test credentials.
✅ Navigating the marketplace as a first-time user.

7.2 Peer Testing & Feedback

- **Feedback:** Users found the navigation intuitive.
- **Issue:** Minor UI glitch in mobile view, fixed by adjusting CSS.
