Hackathon Day - 5 Bandage Marketplace Template Testing, Error Handling, and Backend Integration Refinement

Objective:
Thoroughly test, optimize, and refine "My Marketplace" for real-world deployment. Ensure seamless backend integrations, enhance performance and prepare the platform to handle customer traffic efficiently while delivering a smooth and error-free user experience.

Deployment Readiness Plan for "Bandage-Store"

Comprehensive Testing

- Validate all platform features and backend integrations.
- Perform functional, non-functional, and user acceptance testing to ensure system stability.

Error Handling

- Implement user-friendly error messages and fallback mechanisms.
- Gracefully manage API failures with fallback UI elements to maintain usability.

Performance Optimization:

- Improve platform speed, responsiveness, and overall reliability.
- Conduct thorough performance testing to identify and resolve bottlenecks.

Cross-Browser and Device Compatibility:

- Ensure seamless functionality across multiple browsers and devices through rigorous testing.

Security Testing

- Identify vulnerabilities and implement measures to secure sensitive user data.

Documentation

- Generate professional testing reports (CSV-based) summarizing results and resolutions.
- Prepare deployment-ready documentation, including best practices for future maintenance.

Key Learning Outcomes:

1. **Comprehensive Testing:**

- Ensure all platform features and backend integrations function as expected.
- Conduct thorough functional, non-functional, and user acceptance testing.

.2. Error Handling

- Implement clear, user-friendly error messages and fallback mechanisms.
- Handle API failures gracefully with fallback UI elements to maintain a smooth experience.

3. Performance Optimization

- Improve platform speed, responsiveness, and overall stability.
- Perform extensive performance testing to detect and resolve bottlenecks.

4. Cross-Browser and Device Compatibility

- Verify consistent functionality across multiple browsers and devices.

5. Security Testing

- Identify security vulnerabilities and implement measures to protect sensitive data.

Deployment Readiness Plan

1. Documentation:

- Generate professional testing reports (CSV-based) summarizing results and resolutions.
- Prepare comprehensive, deployment-ready documentation outlining best practices.

Testing & Optimization Plan

Step 1: Functional Testing:

Objective: Ensure the marketplace's core features function as expected, delivering a smooth and error-free experience.

Key Features to Test

Product Listing: Confirm that all products are displayed accurately with correct details.
Product Details Page: Verify that price, description, images, and availability are correctly shown.
Category Filters: Ensure filters return accurate and relevant results.
Dynamic Routing: Validate seamless navigation to individual product detail pages.
Add to Cart: Confirm that users can add, update, and remove items from the cart.
Add to Wishlist: Ensure products can be added to and managed in the wishlist.
Responsive Design: Test and validate UI adaptability across different screen sizes (mobile, tablet, desktop).
User Profile Management: Verify that users can update personal details, view order history, and manage saved addresses.

Step 2: Error Handling:

Objective: Implement robust error-handling mechanisms to maintain a smooth experience even when issues occur.

Key Areas to Address

Network Failures:

- Display user-friendly error messages when connectivity issues arise.
- Example: "Unable to connect to the server. Please check your internet connection."

API Errors:

- Implement fallback UI elements when API calls fail or return errors.
- Example: If product data fails to load, display a message like "Product details are currently unavailable. Please try again later."

Form Validation Errors:

- Ensure clear and helpful error messages for incorrect inputs (e.g., invalid email format, required fields missing).

Session Expiration:

- Notify users when their session has expired and prompt re-authentication.

Step 3: Advanced Error Handling

Objective: Ensure smooth user experience by handling missing data, server issues, and API failures gracefully.

Key Areas to Address

Invalid or Missing Data:

- Detect and handle cases where API responses return incomplete or incorrect data.
- **Example Message:** "Some information is missing. Please try again later."

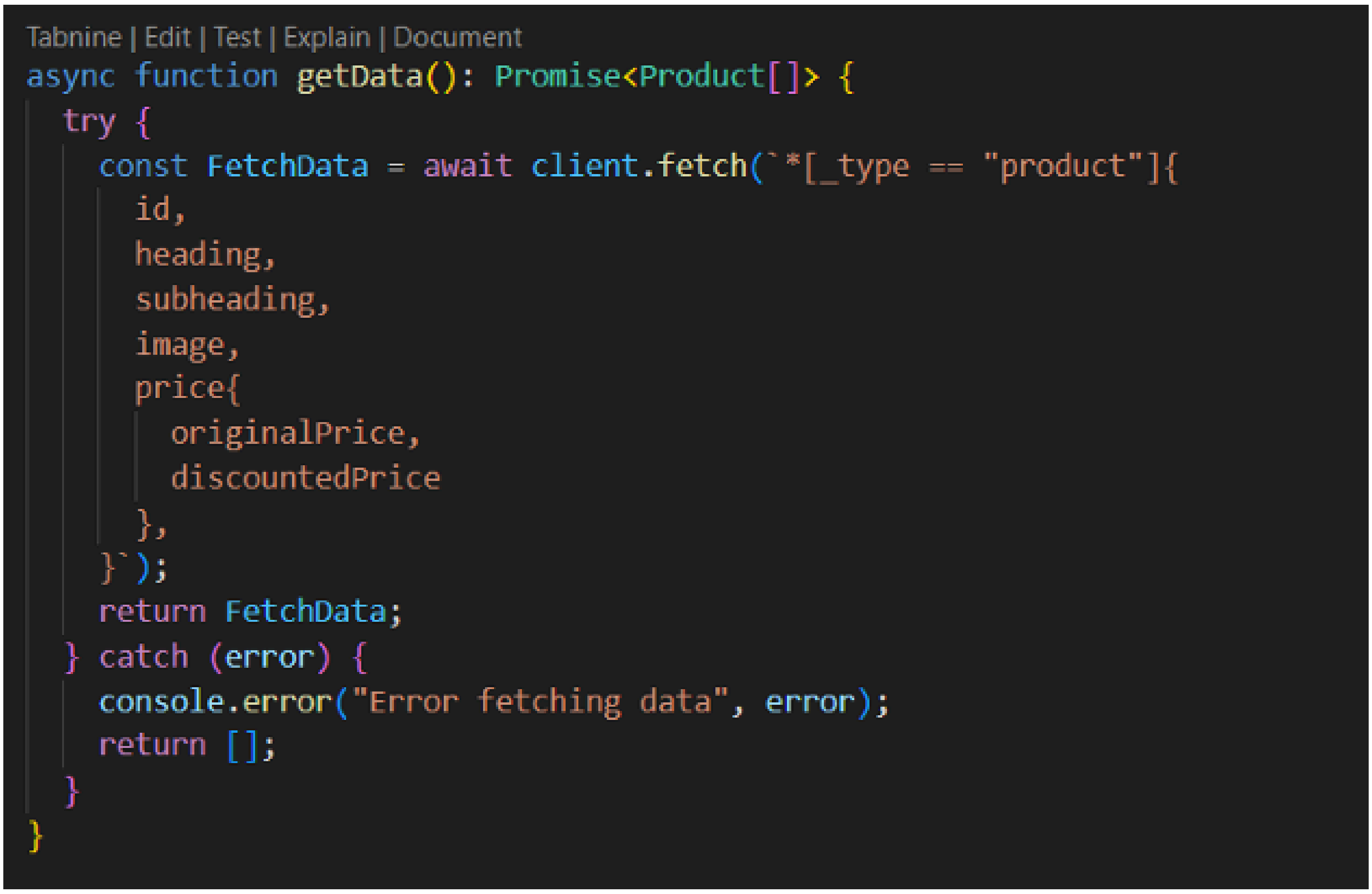
Unexpected Server Errors:

- Provide user-friendly fallback messages for server-side failures.

- **Example Message:** "Something went wrong on our end. Please try again later."

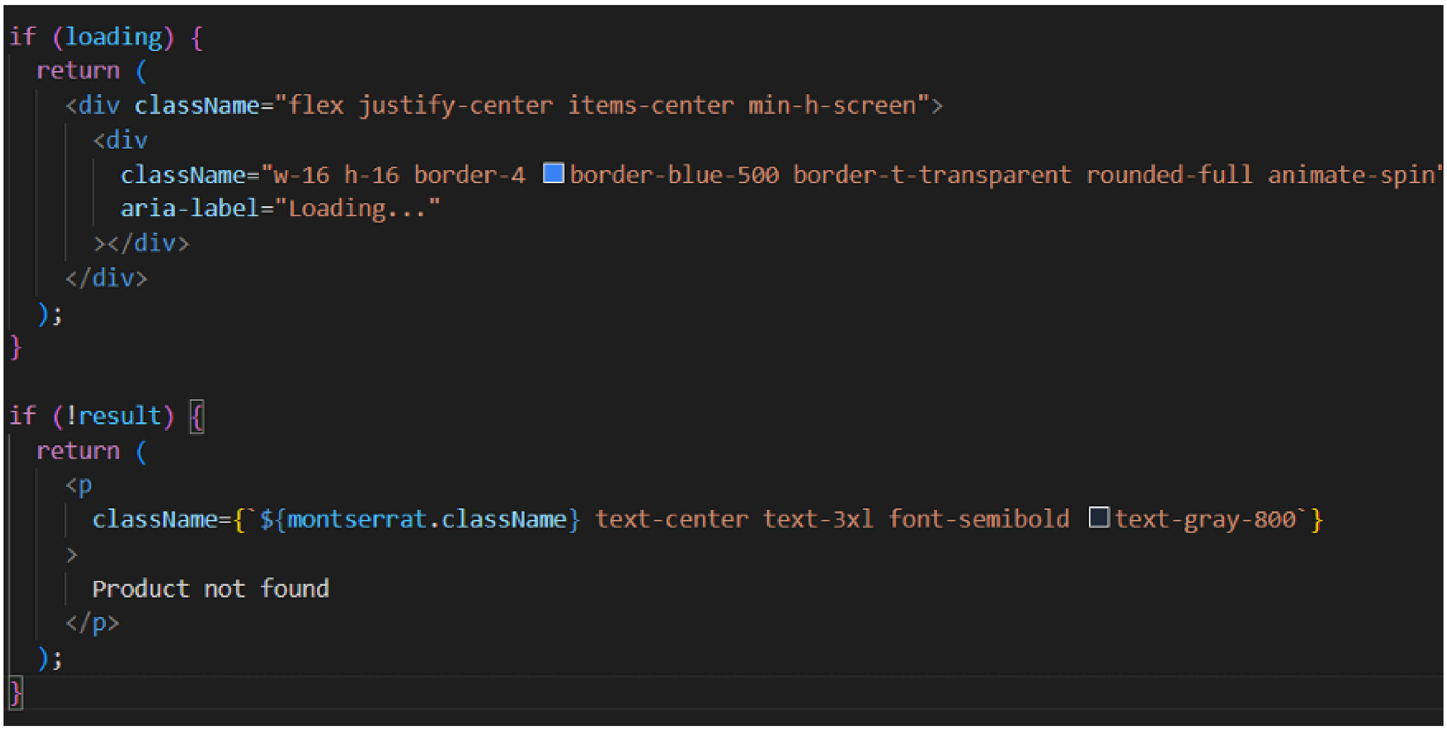
API Error Handling:

- Use try-catch blocks in API calls to prevent crashes and improve error resilience.
- Implement default values or cached data when API responses fail.



```
async function getData(): Promise<Product[]> {
  try {
    const FetchData = await client.fetch(`*[_type == "product"]{
      id,
      heading,
      subheading,
      image,
      price{
        originalPrice,
        discountedPrice
      },
    }`);
    return FetchData;
  } catch (error) {
    console.error("Error fetching data", error);
    return [];
  }
}
```

Fallback UI Elements: Provide alternative content when data is unavailable. Example: Display a "No products available" message or a placeholder image when the product list is empty.

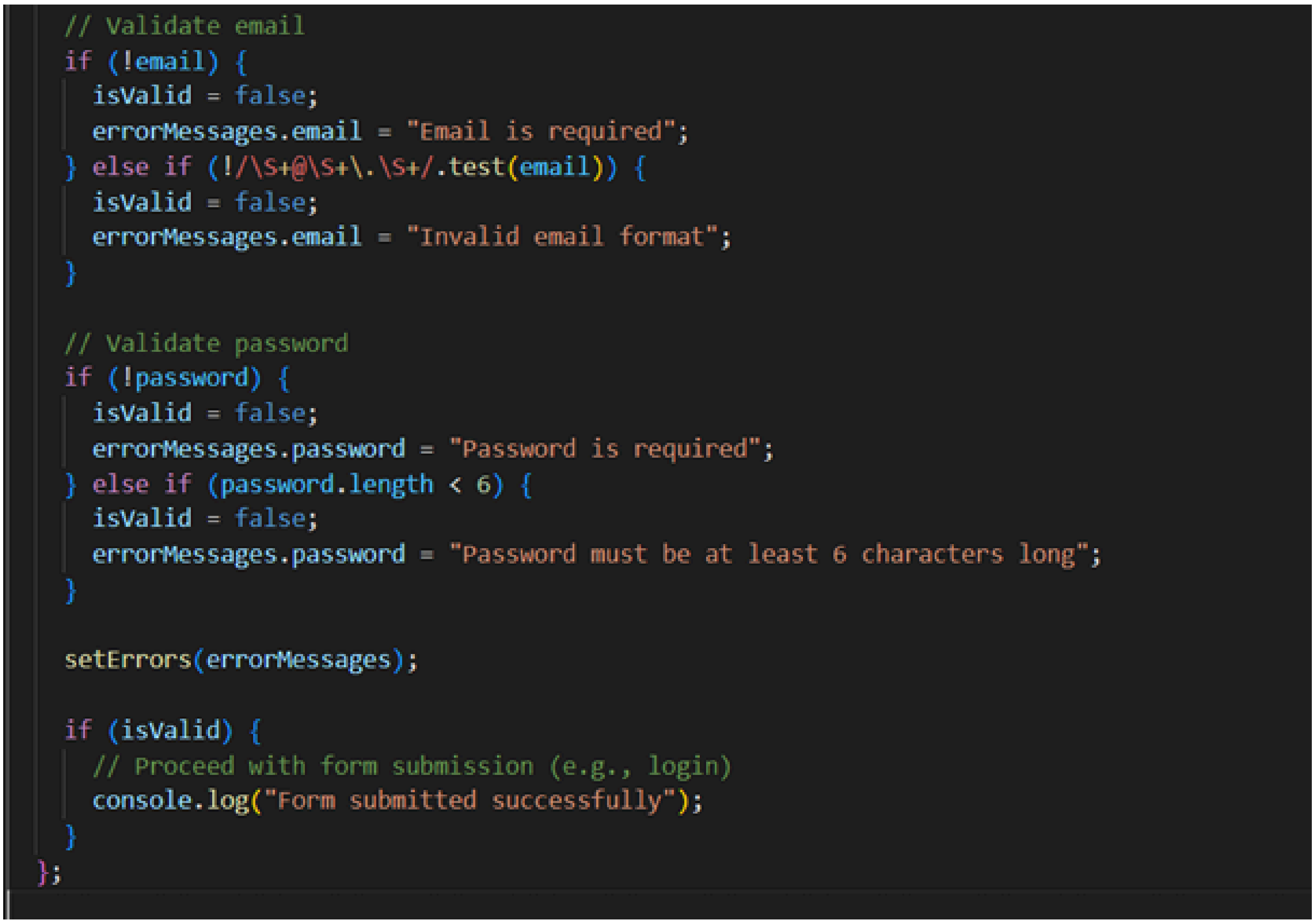


```
if (loading) {
  return (
    <div className="flex justify-center items-center min-h-screen">
      <div
        className="w-16 h-16 border-4 border-blue-500 border-t-transparent rounded-full animate-spin"
        aria-label="Loading..."
      ></div>
    </div>
  );
}

if (!result) {
  return (
    <p
      className={`${montserrat.className} text-center text-3xl font-semibold text-gray-800`}
    >
      Product not found
    </p>
  );
}
```

Form Validation Errors: Validate user inputs on both the front end and backend to avoid invalid data submissions.

Example: Show specific error messages like "Email is required" or "Invalid phone number format."



```
  // Validate email
  if (!email) {
    isValid = false;
    errorMessages.email = "Email is required";
  } else if (!/\S+@\S+\.\S+/.test(email)) {
    isValid = false;
    errorMessages.email = "Invalid email format";
  }

  // Validate password
  if (!password) {
    isValid = false;
    errorMessages.password = "Password is required";
  } else if (password.length < 6) {
    isValid = false;
    errorMessages.password = "Password must be at least 6 characters long";
  }

  setErrors(errorMessages);

  if (isValid) {
    // Proceed with form submission (e.g., login)
    console.log("Form submitted successfully");
  }
};
```

Step 4: Performance Optimization

Objective: Improve load times, responsiveness, and overall efficiency by eliminating bottlenecks and enhancing user experience.

Key Areas to Address

Optimize Assets

- **Image Compression:** Use tools like **TinyPNG** or **ImageOptim** to reduce file sizes without sacrificing quality.
- **Lazy Loading:** Defer loading of off-screen images and assets to enhance initial page speed.

Minimize JavaScript and CSS

- **Minification:** Reduce JavaScript (via **Terser**) and CSS (via **CSSNano**) file sizes for faster rendering.
- **Remove Unused Code:** Eliminate unnecessary CSS and JavaScript to improve page speed.

Implement Caching Strategies

- **Browser Caching:** Store static assets locally in the user's browser to reduce redundant network requests.

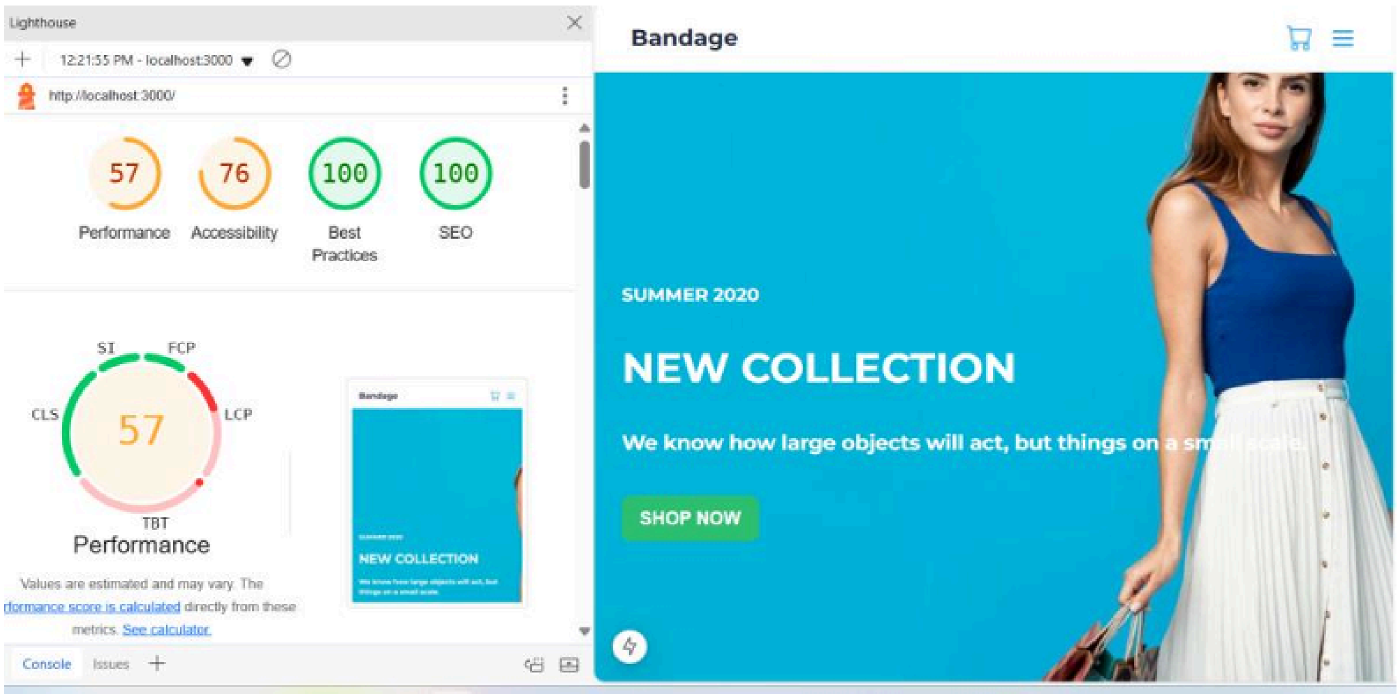
Performance Optimization (Continued)

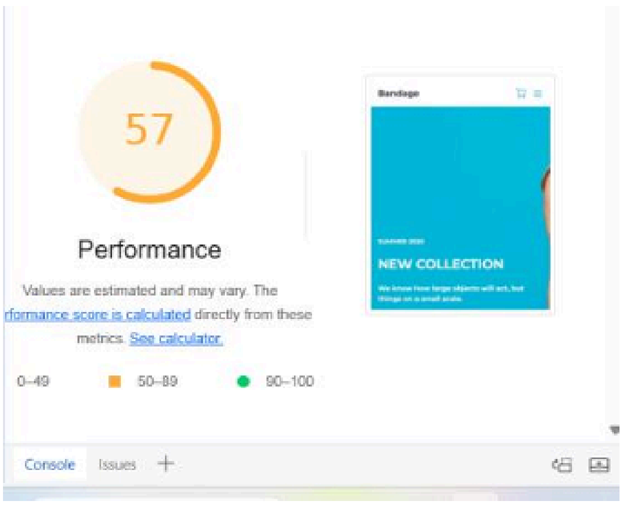
Service Workers:

- **Cache Resources:** Utilize **service workers** to store frequently accessed assets, reducing load times.
- **Offline Functionality:** Enable limited browsing even when the user is offline.

Analyze Performance:

- **Google Lighthouse:** Conduct audits to pinpoint areas for improvement, including image optimization and resource loading.
- **WebPageTest & GTmetrix:** Leverage these tools to detect bottlenecks and enhance page load speeds.

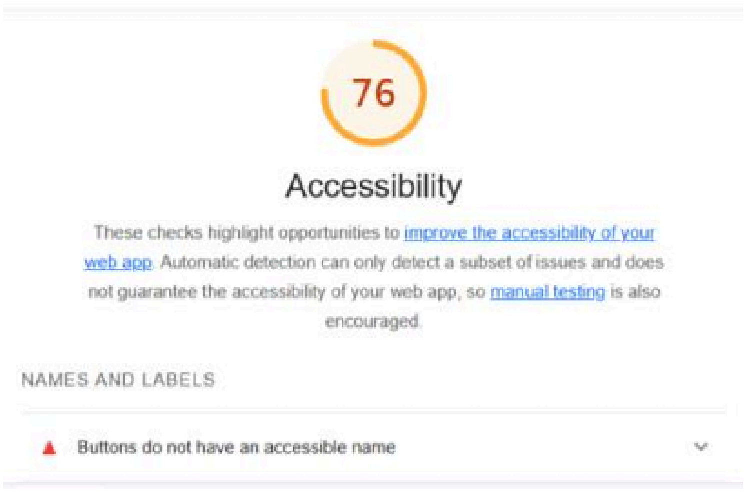

Lighthouse
12:21:55 PM - localhost:3000
http://localhost:3000/
57
Performance
76
Accessibility
100
Best Practices
100
SEO
SI
FCP
CLS
LCP
57
TBT
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
Console
Issues
Bandage
SUMMER 2020
NEW COLLECTION
We know how large objects will act, but things on a small scale.
SHOP NOW


57

Performance

Values are estimated and may vary. The rformance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

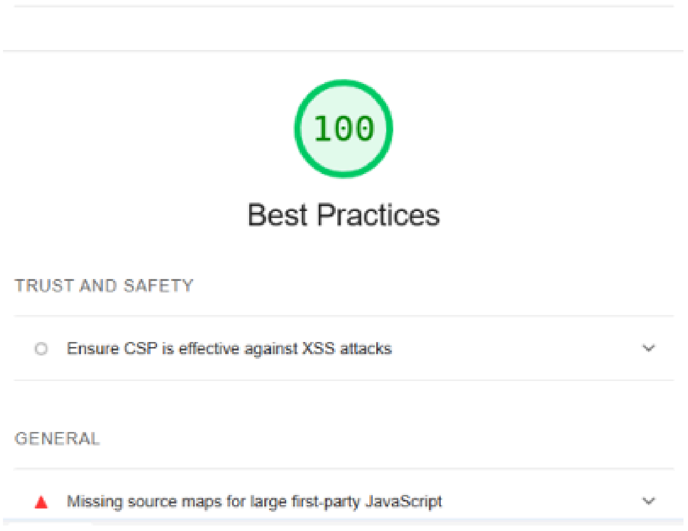

76

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


100

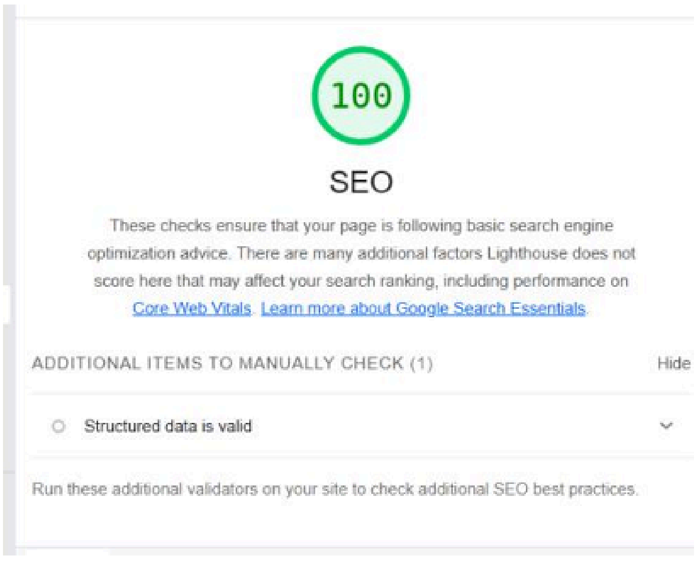
Best Practices

TRUST AND SAFETY

○ Ensure CSP is effective against XSS attacks

GENERAL

▲ Missing source maps for large first-party JavaScript


100

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more about Google Search Essentials.

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Hide

○ Structured data is valid

Run these additional validators on your site to check additional SEO best practices.

Step 5: Security Testing

Ensure Marketplace Security:

Identify vulnerabilities and apply robust security measures to **protect sensitive data** and **prevent cyber threats**.

Key Security Measures

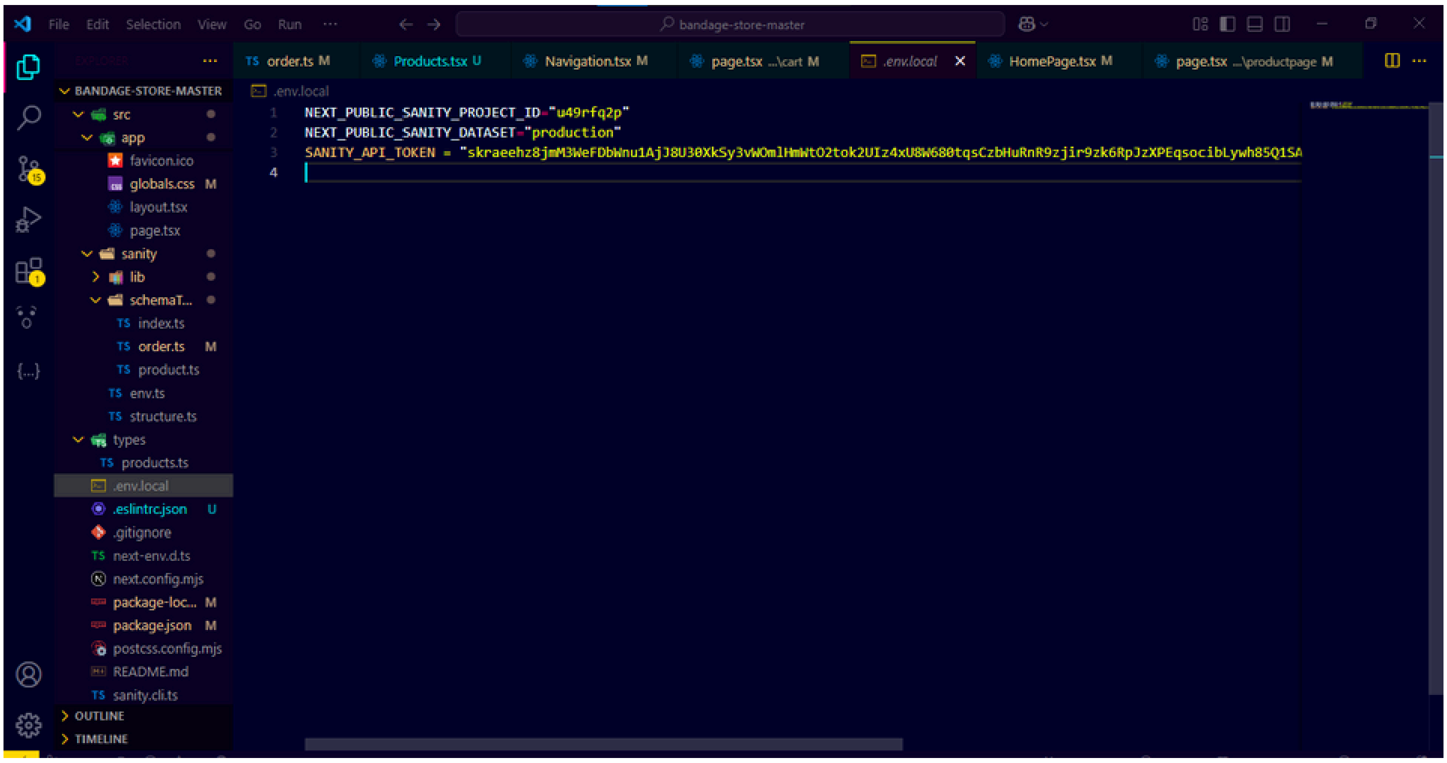
Input Validation

- **Sanitize User Inputs:** Prevent SQL injection, **Cross-Site Scripting (XSS)**, and **other attacks** by validating input fields.
- **Regular Expressions:** Use regex to **validate** user data (emails, phone numbers, etc.).

Secure API Communication

- **Enforce HTTPS:** Encrypt all **API requests** to secure data transmission.

- **Protect Sensitive Data:** Store API keys in **environment variables**, not in the front end.



```
NEXT_PUBLIC_SANITY_PROJECT_ID="u49rfq2p"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKEN = "skraeehz8jmM3WeFDbWnu1AjJ8U30XkSy3vWOmlHmWtO2tok2UIz4xU8W680tqsCzbHuRnR9zjir9zk6RpJzXPEqsocibLywh85Q1SA
```

Authentication & Authorization

- **JWT & OAuth**: Implement secure user authentication using **JSON Web Tokens (JWT)** or **OAuth** for safe user sessions.
- **Access Control**: Enforce **role-based access control (RBAC)** to restrict access to admin-only or user-specific features.

CSRF Protection

- **Anti-CSRF Tokens**: Prevent **Cross-Site Request Forgery (CSRF)** attacks by implementing **tokens** to validate user requests.

Security Headers

- **Content-Security-Policy (CSP)**: Protect against **XSS** by restricting allowed content sources.
- **X-Content-Type-Options**: Prevent MIME-type sniffing vulnerabilities.
- **Strict-Transport-Security (HSTS)**: Enforce HTTPS to **prevent downgrade attacks**.

Vulnerability Scanning

- Use **OWASP ZAP** or **Burp Suite** to scan for common vulnerabilities, including **injections, misconfigurations, and exposure risks**.

Step 6: User Acceptance Testing (UAT)

Goal: Ensure the marketplace meets **user expectations** before final deployment by providing a **seamless, intuitive experience**.

Key Areas to Address: Conduct testing on the marketplace by simulating typical user actions, such as browsing products, searching, adding items to the cart, and completing a purchase. This helps identify usability issues and ensures that all features function as intended in real-world scenarios.
User Feedback: Engage peers, stakeholders, or potential users in testing to collect valuable insights. This will help pinpoint pain points or areas for improvement, ensuring the product aligns with user needs.

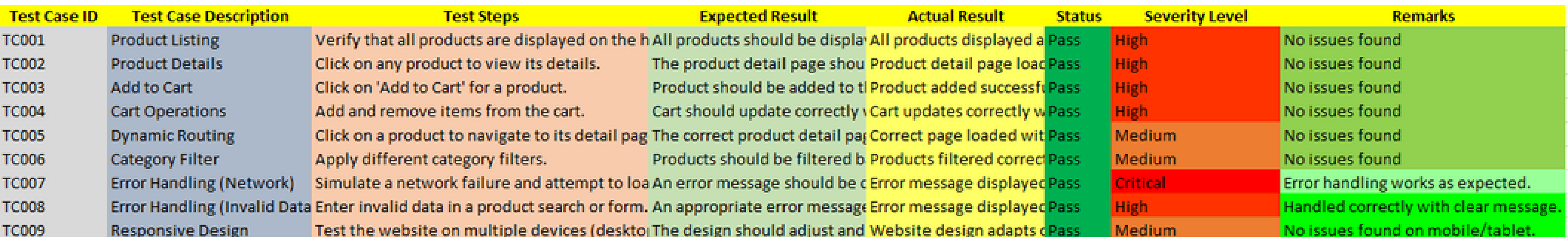
Usability Testing: Confirm that the user interface is intuitive and easy to navigate. Ensure all workflows, such as signing up, logging in, and making a purchase, are simple and error-free.

Edge Cases: Test edge cases, like handling large product inventories, unusual search queries, or user interactions such as failed transactions or login attempts, to ensure the marketplace operates as expected.
Device and Browser Compatibility: Ensure that the user experience remains consistent across various devices (desktop, tablet, mobile) and browsers (Chrome, Firefox, Safari, Edge), promoting accessibility for all users.

CSV Format

Test Case ID,Test Case Description,Test Steps,Expected Result,Actual Result,Status,Severity Level,Remarks TC001,Product Listing,Verify that all products are displayed on the homepage.,All products should be displayed correctly.,All products displayed as expected.,Pass,High,No issues found TC002,Product Details,Click on any product to view its details.,The product detail page should load correctly.,Product detail page loaded without issues.,Pass,High,No issues found TC003,Add to Cart,Click on 'Add to Cart' for a product.,Product should be added to the cart.,Product added successfully to the cart.,Pass,High,No issues found TC004,Cart Operations,Add and remove items from the cart.,Cart should update correctly with the added/removed items.,Cart updates correctly when items are added or removed.,Pass,High,No issues found TC005,Dynamic Routing,Click on a product to navigate to its detail page.,The correct product detail page should load.,Correct page loaded with the right details.,Pass,Medium,No issues found TC006,Category Filter,Apply different category filters.,Products should be filtered based on selected category.,Products filtered correctly based on the selected category.,Pass,Medium,No issues found TC007,Error Handling (Network),Simulate a network failure and attempt to load a product.,An error message should be displayed, indicating a failure.,Error message displayed as expected.,Pass,Critical,Error handling works as expected. TC008,Error Handling (Invalid Data),Enter invalid data in a product search or form.,An appropriate error message should be displayed.,Error message displayed for invalid input.,Pass,High,Handled correctly with clear message. TC009,Responsive Design,Test the website on multiple devices (desktop, tablet, mobile).,The design should adjust and be responsive on all devices.,Website design adapts correctly to various screen sizes.,Pass,Medium,No issues found on mobile/tablet.

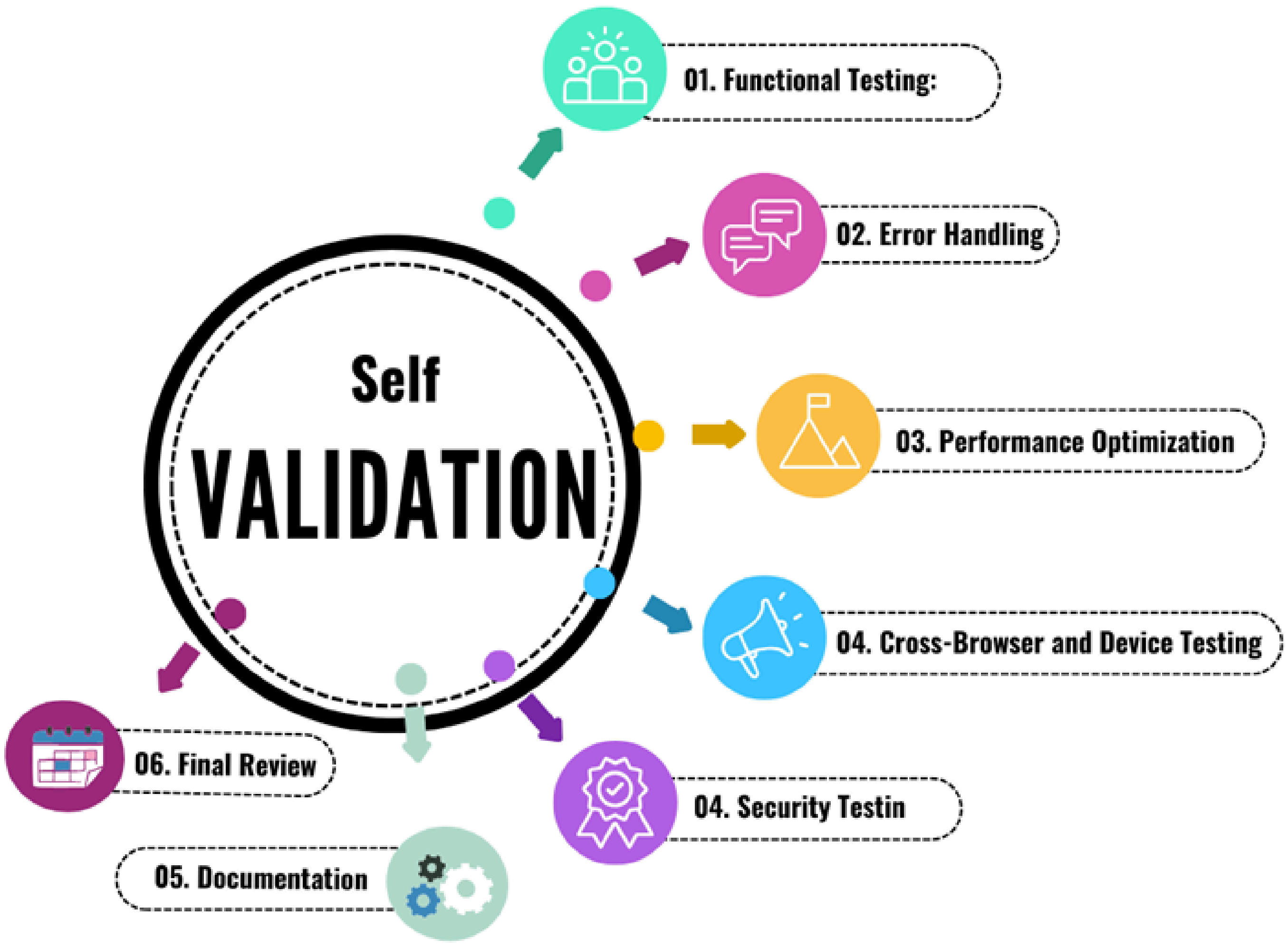
CSV Table



Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Remarks
TC001	Product Listing	Verify that all products are displayed on the h	All products should be displa	All products displayed a	Pass	High	No issues found
TC002	Product Details	Click on any product to view its details.	The product detail page shou	Product detail page loac	Pass	High	No issues found
TC003	Add to Cart	Click on 'Add to Cart' for a product.	Product should be added to tl	Product added successfu	Pass	High	No issues found
TC004	Cart Operations	Add and remove items from the cart.	Cart should update correctly	Cart updates correctly w	Pass	High	No issues found
TC005	Dynamic Routing	Click on a product to navigate to its detail pag	The correct product detail pa	Correct page loaded wit	Pass	Medium	No issues found
TC006	Category Filter	Apply different category filters.	Products should be filtered b	Products filtered correc	Pass	Medium	No issues found
TC007	Error Handling (Network)	Simulate a network failure and attempt to loa	An error message should be c	Error message displayec	Pass	Critical	Error handling works as expected.
TC008	Error Handling (Invalid Data	Enter invalid data in a product search or form.	An appropriate error messag	Error message displayec	Pass	High	Handled correctly with clear message.
TC009	Responsive Design	Test the website on multiple devices (deskto	The design should adjust and	Website design adapts c	Pass	Medium	No issues found on mobile/tablet.

Conclusion:

Day 5 was dedicated to preparing the marketplace for deployment by thoroughly testing, optimizing, and refining all components. Comprehensive functional, error handling and performance tests were conducted to validate critical features such as product listings, cart operations, and dynamic routing. Robust error-handling mechanisms and fallback UI elements were implemented to improve reliability. Performance optimization efforts led to faster load times and better responsiveness across different devices and browsers. The team also documented all findings and resolutions in a professional format, ensuring the marketplace is fully prepared for real-world deployment.


Self
VALIDATION
01. Functional Testing:
02. Error Handling
03. Performance Optimization
04. Cross-Browser and Device Testing
04. Security Testin
05. Documentation
06. Final Review