

Day 05: Marketplace Hackathon

MarketType: Q-commerce

Objective :

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

Key Objective:

Comprehensive Testing: 1. Validate all features and backend integrations. 2. Ensure functional, non-functional, and user acceptance testing is completed. Error Handling: 1. Implement user-friendly error messages and fallback mechanisms.

2. Gracefully handle API errors with fallback UI elements.

Performance Optimization: 1. Enhance speed, responsiveness, and reliability. 2. Conduct performance testing to identify and fix bottlenecks. Cross-Browser and Device Compatibility: 1. Test for seamless functionality across multiple browsers and devices.

Security Testing: 1. Identify vulnerabilities and secure sensitive data. Documentation: 1. Create professional testing reports (CSV-based) summarizing results and resolutions. 2. Prepare deployment-ready documentation with best practices.

Key Learning Outcome:

Comprehensive Testing:

1. Validate all features and backend integrations.

2. Ensure functional, non-functional, and user acceptance testing is completed.

Error Handling:

1. Implement user-friendly error messages and fallback mechanisms.

2. Gracefully handle API errors with fallback UI elements.

Performance Optimization:

1. Enhance speed, responsiveness, and reliability.

2. Conduct performance testing to identify and fix bottlenecks.

Cross-Browser and Device Compatibility:

1. Test for seamless functionality across multiple browsers and devices.

Security Testing: 2. 1. Identify vulnerabilities and secure sensitive data.

Documentation:

1. Create professional testing reports (CSV-based) summarizing results and resolutions.

2. Prepare deployment-ready documentation with best practices.

Step 01 Function Testing

Ensure the marketplace's key features operate as intended, delivering a smooth and error-free user experience.

Key Features to Test Product Listing: Verify products are displayed accurately with proper details.

Product Details: Ensure product detail pages show correct information like price, description, images, and availability.

Category Filters: Test that filters return precise results based on user selection.

Dynamic Routing: Validate seamless navigation to individual product detail pages.

Add to Cart: Confirm items can be added, updated, and removed from the cart.

Add to Wishlist: Ensure products can be added to and managed in the wishlist.

Responsive Design: Validate that all features and pages adapt seamlessly to various screen sizes (mobile, tablet, and desktop).

User Profile Management: Check that users can update personal information, view order history, and manage saved addresses.

Step 02: Error Handling:

Implement robust error handling mechanisms to ensure a smooth user experience, even when issues arise, by displaying clear and user-friendly fallback messages.

Key Areas to Address in Error Handling Network Failures:
Display a meaningful error message when there are connectivity issues

Example: "Unable to connect to the server. Please check your internet connection"

Invalid or Missing Data: Handle cases where API responses return incomplete or invalid data.

Example: "Some information is missing. Please try again later."
Unexpected Server Errors: Use generic fallback messages for unhandled server-side errors.

```
Tabnine | Edit | Test | Explain | Document
async function getData(): Promise<Product[]> {
  try {
    const fetchData = await client.fetch(`${_type} -- "product"`) {
      id,
      heading,
      subheading,
      image,
      price {
        originalPrice,
        discountedPrice
      },
    };
  } 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>
  );
}
```

Step 03 Performance Optimization .

Optimize the marketplace to improve load times, responsiveness, and overall performance. This step involves identifying bottlenecks and applying strategies to enhance the user experience.

Key Areas to Address

Optimize Assets:

Image Optimization: Compress images using tools like TinyPNG or ImageOptim to reduce size without losing quality.

Lazy Loading: Implement lazy loading for images and assets to defer loading until needed, improving initial load times.

Minimize JavaScript and CSS: Minification: Minify JavaScript (using Terser) and CSS (using CSSNano) to reduce file sizes for faster loading.

Remove Unused Code: Eliminate unused CSS and JavaScript to further reduce file sizes.

Implement Caching Strategies: Browser Caching: Store static assets in the user's browser to avoid repeated network requests

Service Workers: Cache resources and improve offline functionality using service workers.

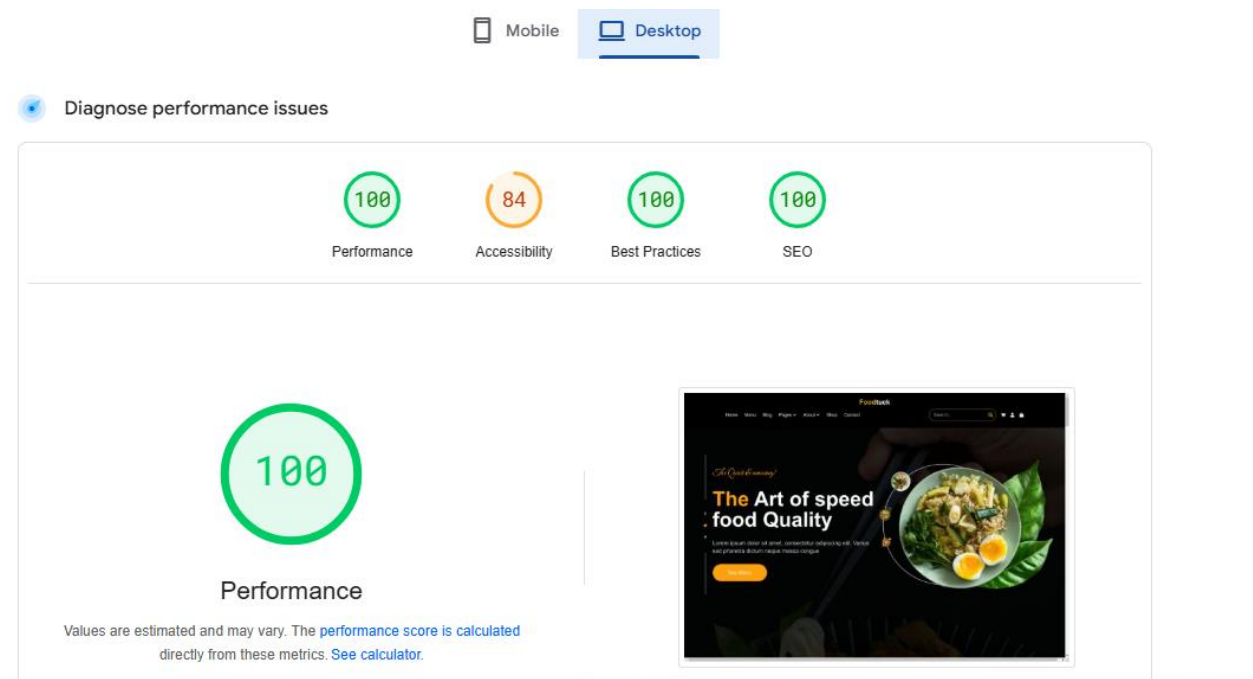
Analyze Performance: Google Lighthouse: Use Lighthouse to audit the website's performance and identify areas to improve, such as image optimization and resource loading.

WebPageTest & GTmetrix: Use these tools to identify performance bottlenecks and improve page load speeds

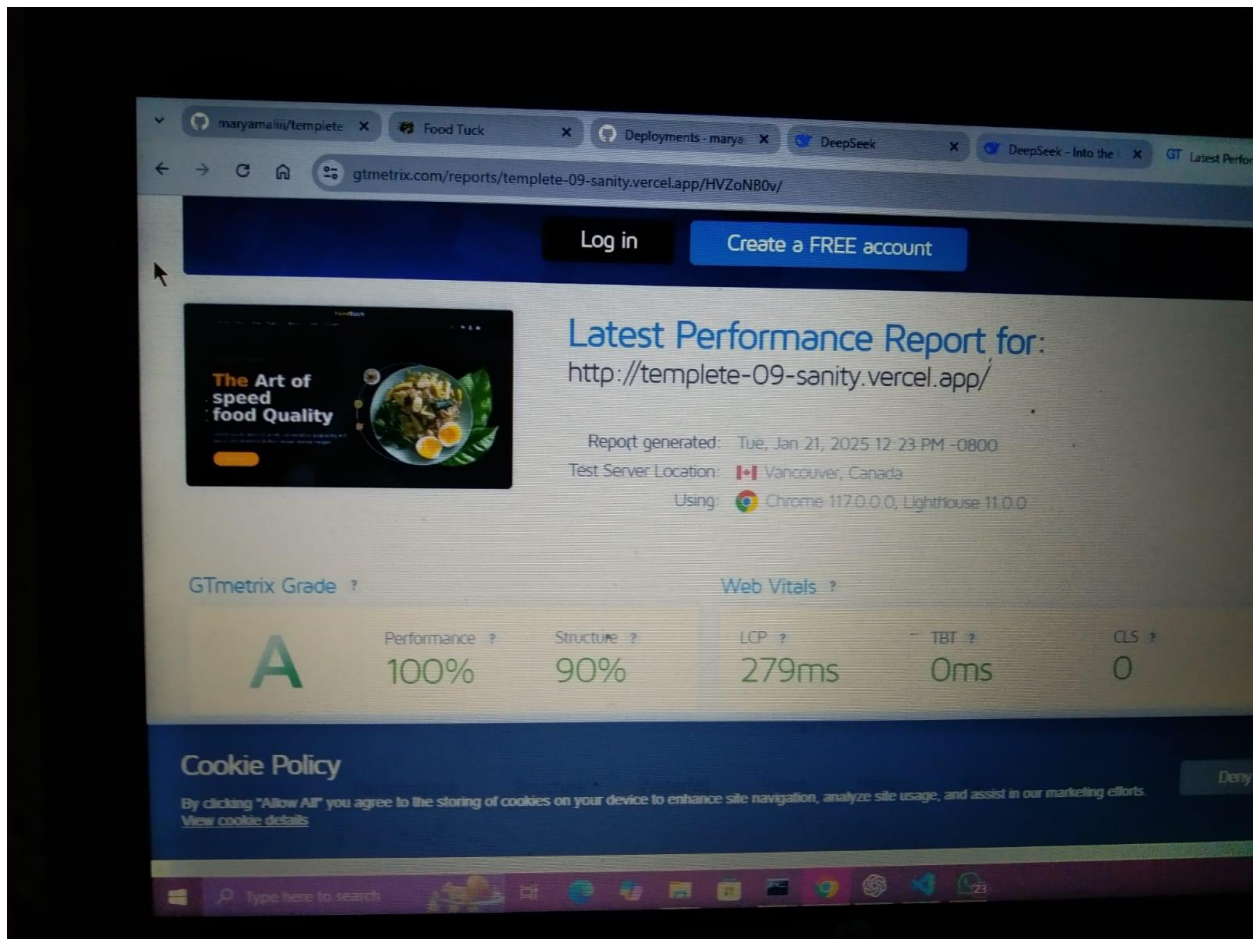
Step 04 Analyze Performance Testing:

Desktop.

Google lighthouse.



Gtmetrix.

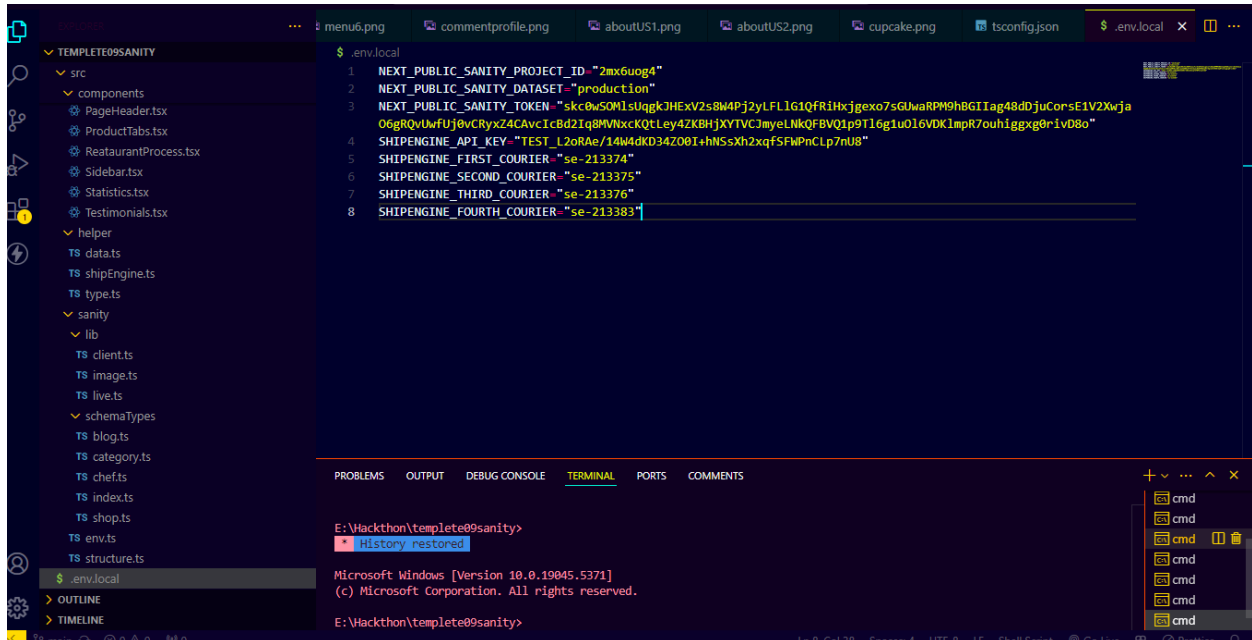


Step 05 Security testing .

Ensure the security of your marketplace by identifying vulnerabilities and implementing measures to protect sensitive data and prevent attacks.

Key Areas to Address Input Validation: Sanitize all user inputs to prevent SQL injection, cross-site scripting (XSS), and other injection attacks. Use regular expressions to validate fields like email addresses, phone numbers, and other critical inputs.

Secure API Communication: Ensure all API calls are made over HTTPS to encrypt data in transit. Avoid exposing sensitive information, such as API keys, in the frontend code. Store them securely in environment variables.



The screenshot shows a VS Code editor with a file explorer on the left displaying the project structure for 'template09sanity'. The main editor area shows the contents of the '.env.local' file, which contains the following environment variables:

```
1 NEXT_PUBLIC_SANITY_PROJECT_ID="2mx6uog4"  
2 NEXT_PUBLIC_SANITY_DATASET="production"  
3 NEXT_PUBLIC_SANITY_TOKEN="skc0wSOM1sUagk3HEXv2s8W4Pj2yLFL1G1QfRiHxjgexo7sGJwaRPM9hBGITag48dDjuCorsE1V2Xwja  
  O6gRQvUwFuj0vCRyxZ4CAvcIcBd21q8MVNxcKQtley4ZKBHjXYTVC3myeLNkQFBVQ1p9Tl6g1u016VDK1mpR7ouhigxg8r1vD8o"  
4 SHIPENGINE_API_KEY="TEST_L2oRAe/14W4dKD34Z00I+hMSsXh2xqf5FwPnClp7nU8"  
5 SHIPENGINE_FIRST_COURIER="se-213374"  
6 SHIPENGINE_SECOND_COURIER="se-213375"  
7 SHIPENGINE_THIRD_COURIER="se-213376"  
8 SHIPENGINE_FOURTH_COURIER="se-213383"
```

Below the editor, the 'TERMINAL' panel is visible, showing the command prompt at 'E:\Hackthon\template09sanity>' and a message 'History restored'.

Authentication & Authorization:

Implement secure user authentication mechanisms like JWT (JSON Web Tokens) and OAuth. Ensure proper authorization checks are in place to restrict access to certain features or resources.

Cross-Site Request Forgery (CSRF) Protection: Implement anti-CSRF tokens to prevent unauthorized actions from being executed on behalf of an authenticated user

Security Headers: Set HTTP security headers like Content-Security-Policy (CSP), X-Content-Type-Options, and Strict-Transport-Security (HSTS) to enhance security.

Vulnerability Scanning: Use tools like OWASP ZAP or Burp Suite to scan your marketplace for common security vulnerabilities and fix identified issues.

6. User Acceptance Testing (UAT) Objective: Ensure that the marketplace meets user expectations and provides a seamless, intuitive experience before final deployment

Key area to Address.

Test the marketplace by performing typical user tasks, such as browsing products, searching, adding items to the cart, and completing a purchase. This will help identify usability issues and ensure that all features work as expected in real-world scenarios.

User Feedback: Involve peers, stakeholders, or potential users in testing to gather valuable feedback. This will help identify pain points or areas for improvement, ensuring the product meets user needs.

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

Edge Cases: Test edge cases such as handling large product inventories, unusual search queries, and user interactions like failed transactions or login

attempts to ensure the marketplace behaves as expected.

CSV format

```
Test Case ID,Test Case Description,Test Steps,Expected Result,Actual Result>Status,Severity Level,Assigned To,Remarks
TC001,Product Listing,Verify products on homepage,Products displayed correctly,Displayed correctly,Pass,High,Team A,No issues found
TC002,Product Details,Click on product,Product page loads,Loaded successfully,Pass,High,Team B,No issues found
TC003,Add to Cart,Click 'Add to Cart',Product added to cart,Added successfully,Pass,High,Team A,No issues found
TC004,Cart Operations,Add and remove items from cart,Cart updates correctly,Updated correctly,Pass,High,Team C,No issues found
TC005,Dynamic Routing,Click on product,Product page loads dynamically,Loaded correctly,Pass,Medium,Team B,No issues found
TC006,Category Filter,Apply filter,Products filtered correctly,Filtered correctly,Pass,Medium,Team C,No issues found
TC007,Error Handling (Network),Simulate network failure,Error message displayed,Message displayed,Pass,Critical,Team A,Handled correctly
TC008,Error Handling (Invalid Data),Enter invalid data,Error message displayed,Message displayed,Pass,High,Team A,Handled correctly
TC009,Responsive Design,Test on multiple devices,Responsive design works,Works on all devices,Pass,Medium,Team D,No issues found
```