

HACKATHON-3-Day 5

Testing, Error Handling, and Backend Integration Refinement

Objective:

Preparing your marketplace for real-world deployment by ensuring all components are thoroughly tested, optimized for performance, and ready to handle customer-facing traffic.

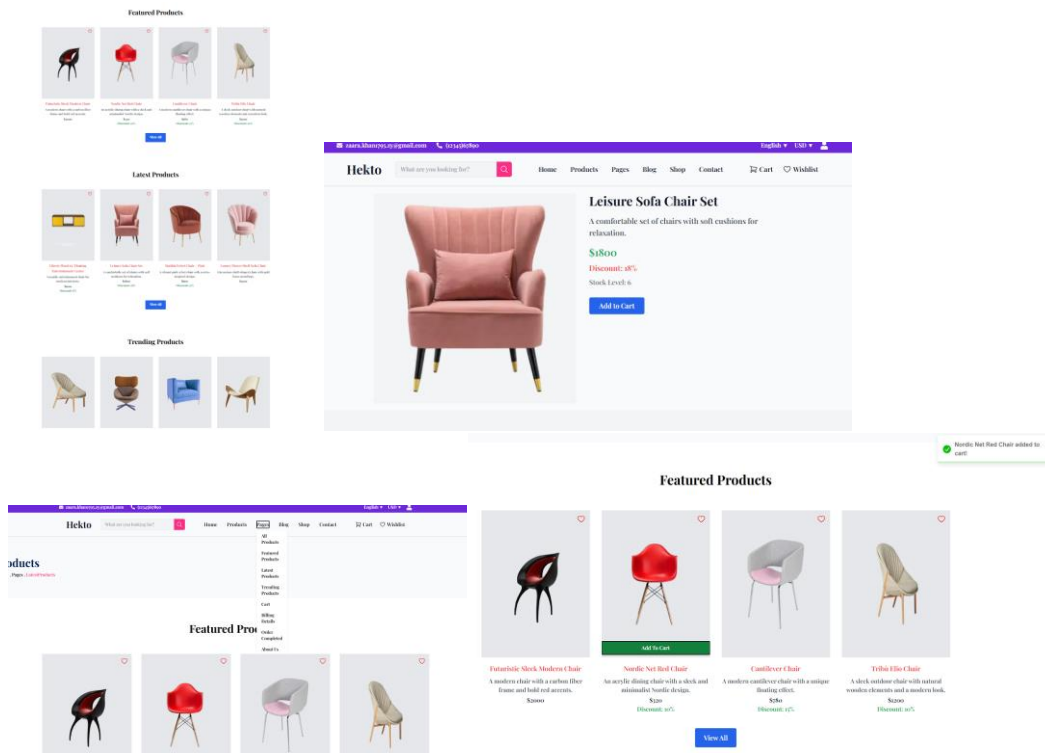
The emphasis will be on testing

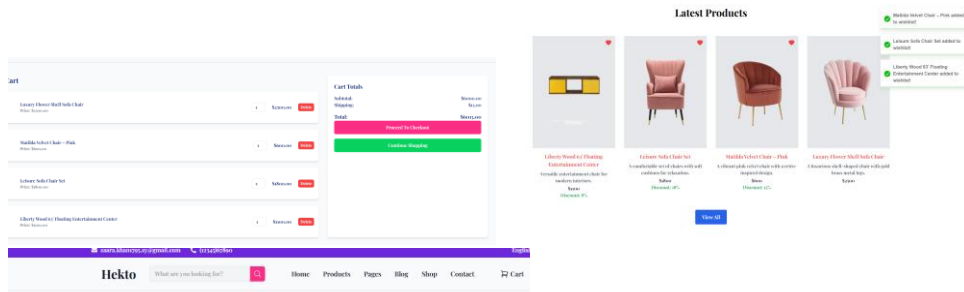
- Backend integrations,
- Implementing error handling, and
- Refining the user experience.

Step 1: Functional Testing

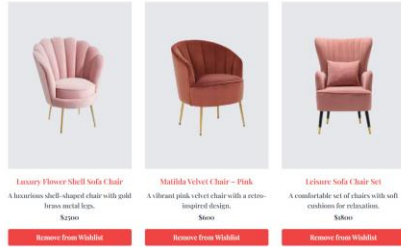
1. Test Core Features:

- Product listing
- Cart operations: Add, update, and remove items from the cart.
- Dynamic routing: Verify individual product detail pages load correctly.





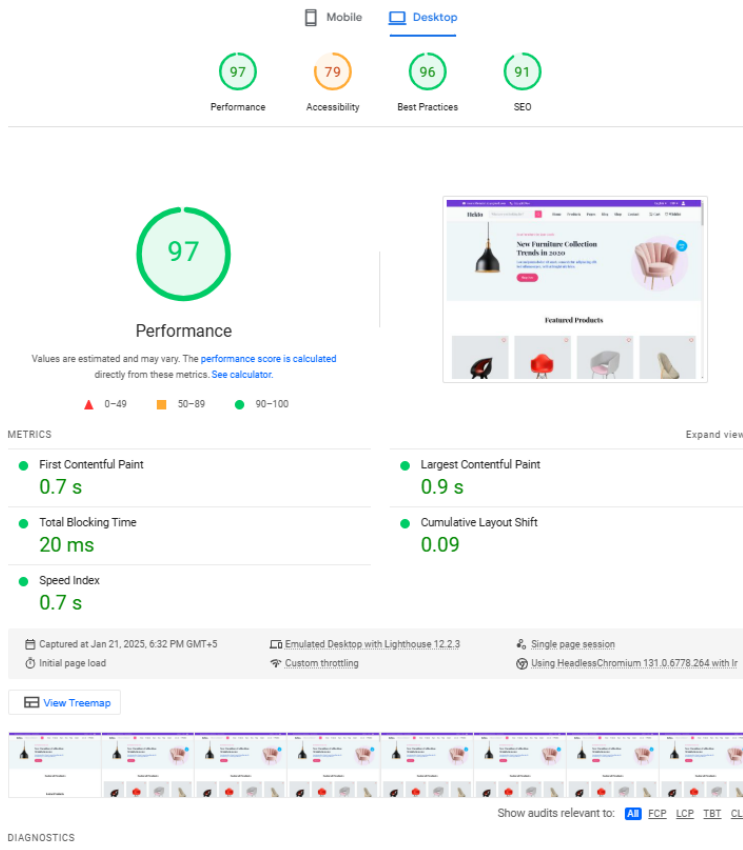
Your Wishlist



2. Testing Tools:

PAGESPEED INSIGHTS

DESKTOP VIEW



MOBILE VIEW

MobileDesktop

Discover what your real users are experiencingNo Data

Diagnose performance issues

81799691

PerformanceAccessibilityBest PracticesSDD

81

Performance

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

▲ 0-49

■ 50-89

● 90-100

Expand view

METRICS

First Contentful Paint2.6 s

Total Blocking Time60 ms

Speed Index2.8 s

Largest Contentful Paint4.4 s

Cumulative Layout Shift0.003

Captured at Jan 21, 2023, 8:32 PM GMT+5Initial page loadSimulated Mobile Power with Lighthouse 12.2.2Show settingsSingle page sessionUltra HeadlessChromium 121.0.8778.265 with 100% CPU throttling

View Treemap

Show audits relevant to:CLSFCPLCPFIDTBTCLS

DIAGNOSTICS

▲ Largest Contentful Paint element — 4,410 ms

▲ Eliminate render-blocking resources — Potential savings of 1,620 ms

▲ Largest Contentful Paint image was lazily loaded

▲ Reduce unused JavaScript — Potential savings of 24 KiB

■ Serve images in next-gen formats — Potential savings of 460 KiB

○ Avoid large layout shifts — 1 layout shift found

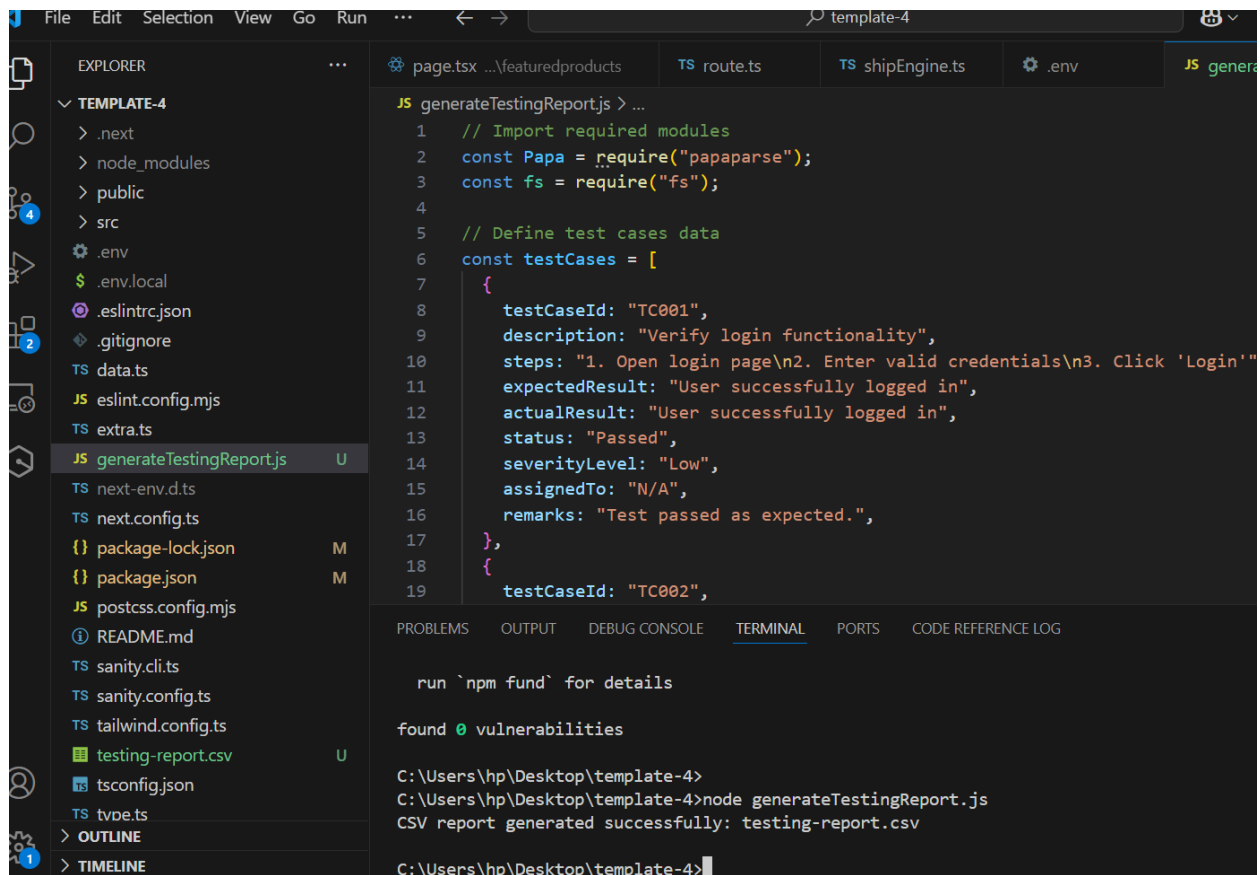
○ JavaScript execution time — 0.3 s

○ Minimize main-thread work — 0.6 s

○ Avoid long main-thread tasks — 1 long task found

○ Initial server response time was short — Root document took 30 ms

3:FUNCTIONAL TESTING BY TEST CASES



The screenshot shows a Visual Studio Code editor with a project named 'template-4'. The Explorer sidebar on the left lists files including 'generateTestingReport.js'. The main editor displays the content of 'generateTestingReport.js', which is a TypeScript file. It imports 'papaparse' and 'fs', and defines an array of test cases. The first test case, 'TC001', describes a login functionality test with steps, expected and actual results, status, severity level, assigned to, and remarks. The second test case, 'TC002', is partially visible. Below the code editor, the TERMINAL tab is active, showing the command 'run `npm fund` for details' and the output 'found 0 vulnerabilities'. The command prompt shows the execution of 'node generateTestingReport.js', resulting in the message 'CSV report generated successfully: testing-report.csv'.

```
JS generateTestingReport.js > ...
1  // Import required modules
2  const Papa = require("papaparse");
3  const fs = require("fs");
4
5  // Define test cases data
6  const testCases = [
7    {
8      testCaseId: "TC001",
9      description: "Verify login functionality",
10     steps: "1. Open login page\n2. Enter valid credentials\n3. Click 'Login'",
11     expectedResult: "User successfully logged in",
12     actualResult: "User successfully logged in",
13     status: "Passed",
14     severityLevel: "Low",
15     assignedTo: "N/A",
16     remarks: "Test passed as expected.",
17   },
18   {
19     testCaseId: "TC002",
```

run `npm fund` for details

found 0 vulnerabilities

C:\Users\hp\Desktop\template-4>
C:\Users\hp\Desktop\template-4>node generateTestingReport.js
CSV report generated successfully: testing-report.csv

C:\Users\hp\Desktop\template-4>

4:Structure of the Report

- **Test Case Description:** A brief explanation of the test.
- **Test Steps:** Steps to replicate the test scenario.
- **Expected Result:** The outcome you expect.
- **Actual Result:** The outcome observed during testing.
- **Status:** The result of the test: "Passed," "Failed," or "Skipped."
- **Severity Level:** The importance of the issue: "High," "Medium," or "Low."
- **Assigned To:** Name of the person responsible for resolving issues.
- **Remarks:** Any additional notes.

```

TestCaseId,Description,Steps,ExpectedResult,ActualResult,Status,SeverityLevel,AssignedTo,Remarks,,
TC001,Verify login functionality,"1. Open login page
2. Enter valid credentials
3. Click 'Login'",User successfully logged in,User successfully logged in,Passed,Low,N/A,Test passed as expected.,,
TC002,Test invalid credentials,"1. Open login page
2. Enter invalid credentials
3. Click 'Login'",,"Show Error Message,Invalid Credentials",Error message displayed,Passed,Low,John Doe,Works as Expected,,
TC003,Validate cart functionality,"1. Add product to cart
2. View cart",Product appears in the cart,Product appears in the cart,Passed,Low,N/A,Works as expected.,,
TC004,Test checkout flow,"1. Add product to cart
2. Proceed to checkout
3. Enter payment details",Order placed successfully,Order placed successfully,Passed,Low,Jane Smith,Work as expected,,

```

5: Testing Report (CSV Format):

TestCaseld	Description	Steps	ExpectedResult	ActualResult	Status	SeverityLevel	AssignedTo	Remarks
TC001	Verify login functionality	1. Open login page 2. Enter valid credentials 3. Click 'Login'	User successfully logged in	User successfully logged in	Passed	Low	N/A	Test passed as expected.
TC002	Test invalid credentials	1. Open login page 2. Enter invalid credentials 3. Click 'Login'	Show Error Message,Invalid Credentials	Error message displayed	Passed	Low	John Doe	Works as Expected
TC003	Validate cart functionality	1. Add product to cart 2. View cart	Product appears in the cart	Product appears in the cart	Passed	Low	N/A	Works as expected.
TC004	Test checkout flow	1. Add product to cart 2. Proceed to checkout 3. Enter payment details	Order placed successfully	Order placed successfully	Passed	Low	Jane Smith	Work as expected

Step 2: Error Handling

```
function LatestProducts() {  
  
  // Fetch products from Sanity  
  useEffect(() => {  
    const fetchProducts = async () => {  
      const query = `*[_type == "product"] {  
        _id,  
        name,  
        "imageUrl": image.asset->url,  
        price,  
        description,  
        discountPercentage,  
        stockLevel,  
        category  
      }`;  
  
      try {  
        const sanityProducts = await client.fetch(query);  
        console.log('Fetched Products:', sanityProducts); // Debugging step  
        setProducts(sanityProducts);  
      } catch (error) {  
        console.error('Failed to fetch products:', error);  
      }  
    }  
  });  
}
```

RESPONSIVE DESIGN

