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

Overview

This document outlines the requirements for testing, error handling, and backend integration refinement as part of Day 5 tasks. The goal is to ensure your application is functional, secure, performant, and user-friendly, meeting industry standards for real-world deployment.

Key Learning Outcomes

1. Perform comprehensive testing, including functional, non-functional, user acceptance, and security testing.
2. Implement robust error handling mechanisms with user-friendly fallback messages.
3. Optimize application performance for speed and responsiveness.
4. Verify cross-browser compatibility and device responsiveness.
5. Develop professional testing documentation, including a CSV-based test report.
6. Gracefully handle API errors and include fallback UI elements.
7. Prepare detailed documentation for testing results, optimizations, and resolutions.

Key Testing Areas

1. Functional Testing

- Validate all application features, including:

    - Product listing and detail pages.
    - Search and filtering functionalities.
    - Cart operations (add, update, remove).
- Tools: Postman, React Testing Library, Cypress.

2. Error Handling

- Ensure proper error messages for:
    - Network failures.
    - Invalid or missing data.
    - Server errors.
- Implement fallback UI elements (e.g., "No products available" when data is unavailable).

3. Performance Testing

- Identify bottlenecks using tools like Lighthouse, GTmetrix, or WebPageTest.
- Optimize images, CSS, JavaScript, and caching strategies.

4. Cross-Browser and Device Testing

- Test functionality and responsiveness on:
    - Browsers: Chrome, Firefox, Safari, Edge.
    - Devices: Desktop, tablet, mobile.
- Tools: BrowserStack, LambdaTest.

5. Security Testing

- Validate input fields to prevent injection attacks.
- Ensure API calls are secured over HTTPS.
- Tools: OWASP ZAP, Burp Suite.

6. User Acceptance Testing (UAT)

- Simulate real-world scenarios to ensure workflows like browsing, searching, and checkout are error-free.
- Gather feedback and make necessary adjustments.

Steps for Implementation

Step 1: Functional Testing

- Write test cases for all features.
- Simulate user interactions and validate outputs.

Step 2: Error Handling

- Use try-catch blocks to handle API errors gracefully.
- Provide user-friendly error messages and fallback UI.

Step 3: Performance Optimization

- Compress and lazy-load images.
- Analyze performance metrics using Lighthouse.
- Reduce unused CSS and JS files.

Step 4: Cross-Browser and Device Testing

- Test across multiple browsers and devices using tools.
- Manually verify key functionalities on at least one physical mobile device.

Step 5: Security Testing

- Sanitize inputs using regular expressions.
- Ensure sensitive API keys are stored securely in environment variables.

Step 6: User Acceptance Testing (UAT)

- Perform end-to-end tests simulating real user actions.
- Collect feedback from peers or mentors.

Step 7: Documentation Updates

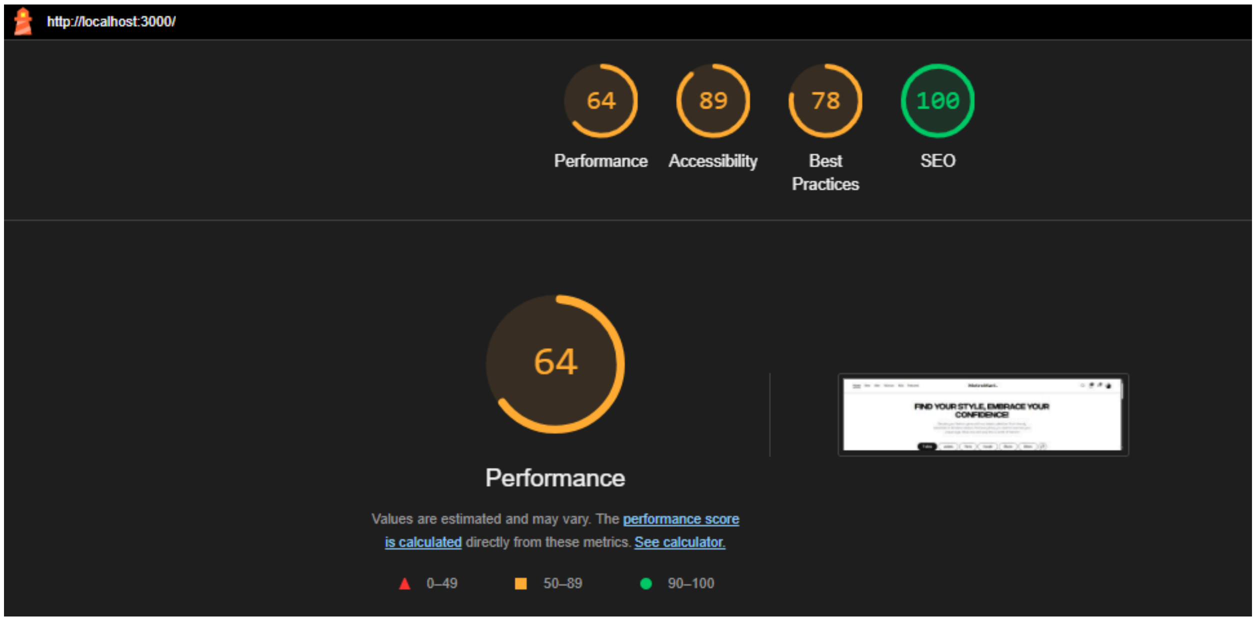
- Document test cases, results, and optimizations.
- Submit a professional report summarizing testing efforts.

Submission Requirements

1. **Functional Deliverables**:
   - Screenshots or recordings of functional and responsive components.
   - Logs or reports from testing tools.
2. **Testing Report (CSV Format)**:
   Include the following columns:
   - Test Case ID
   - Description
   - Test Steps
   - Expected Result
   - Actual Result
   - Status (Passed/Failed/Skipped)
   - Severity Level (High/Medium/Low)
   - Assigned To
   - Remarks
3. **Documentation**:
   - Summarize all testing, optimization steps, and fixes.
   - Acceptable formats: PDF or Markdown.
4. **Repository Submission**:
   - Push updated files, including the testing report, to the designated GitHub repository.
   - Include a clear folder hierarchy and a README file.

Expected Output

1. Fully tested and functional application components.
2. Clear and user-friendly error handling mechanisms.
3. Optimized performance with faster load times.
4. Responsive design verified across browsers and devices.
5. Comprehensive CSV-based testing report.
6. Detailed documentation summarizing all efforts.



http://localhost:3000/

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

▲ 0–49 ■ 50–89 ● 90–100

FAQs

1. **What tools can I use for functional testing?**
   - Cypress, Postman, and React Testing Library.
2. **How do I handle API failures gracefully?**
   - Use try-catch blocks and fallback UI elements (e.g., "No products available").

3. **What should my CSV-based testing report include?**
    - Test Case ID, Description, Steps, Expected/Actual Results, Status, Severity Level, Assigned To, and Remarks.
4. **How do I secure sensitive API keys?**
    - Store API keys in environment variables and ensure the use of HTTPS.
5. **How can I optimize load times?**
    - Compress images, minimize CSS/JS files, and implement caching.
