Hackathon 3 Day 5: Testing, Error Handling, and Backend Integration Refinement

1. Functional Testing:

Overview

Functional testing ensures that all functionalities of the application work as intended. This type of testing focuses on the output of actions based on specific input conditions, verifying that the software meets its functional requirements.

Objectives of Functional Testing

- Validate that the application performs its intended functions.
- Ensure that all features work as expected for valid inputs.
- Identify and fix any defects or deviations from expected behavior.

Scope of Functional Testing

The following features were tested during functional testing:

1. Data Fetching: API integrations for live data retrieval.
2. User Actions: Adding items to the cart, updating profiles, etc.
3. Error Handling: Validation for incorrect inputs.

Additionally, core functionalities were tested, including:

- Product Listing: Verified accurate display of all products.
- Product Details Page: Ensured correct details for selected products.
- Cart Page: Confirmed item addition, removal, and updates.
- Filtering: Tested functionality to filter products based on criteria.
- Page Navigation: Checked all pages for proper loading and responsiveness.

2. Error Handling:

Error handling mechanisms were implemented and tested for the following scenarios:

1. Network Failure:

- When the API is unreachable, an error message is displayed to the user, such as: "Network Error: Please check your internet connection."

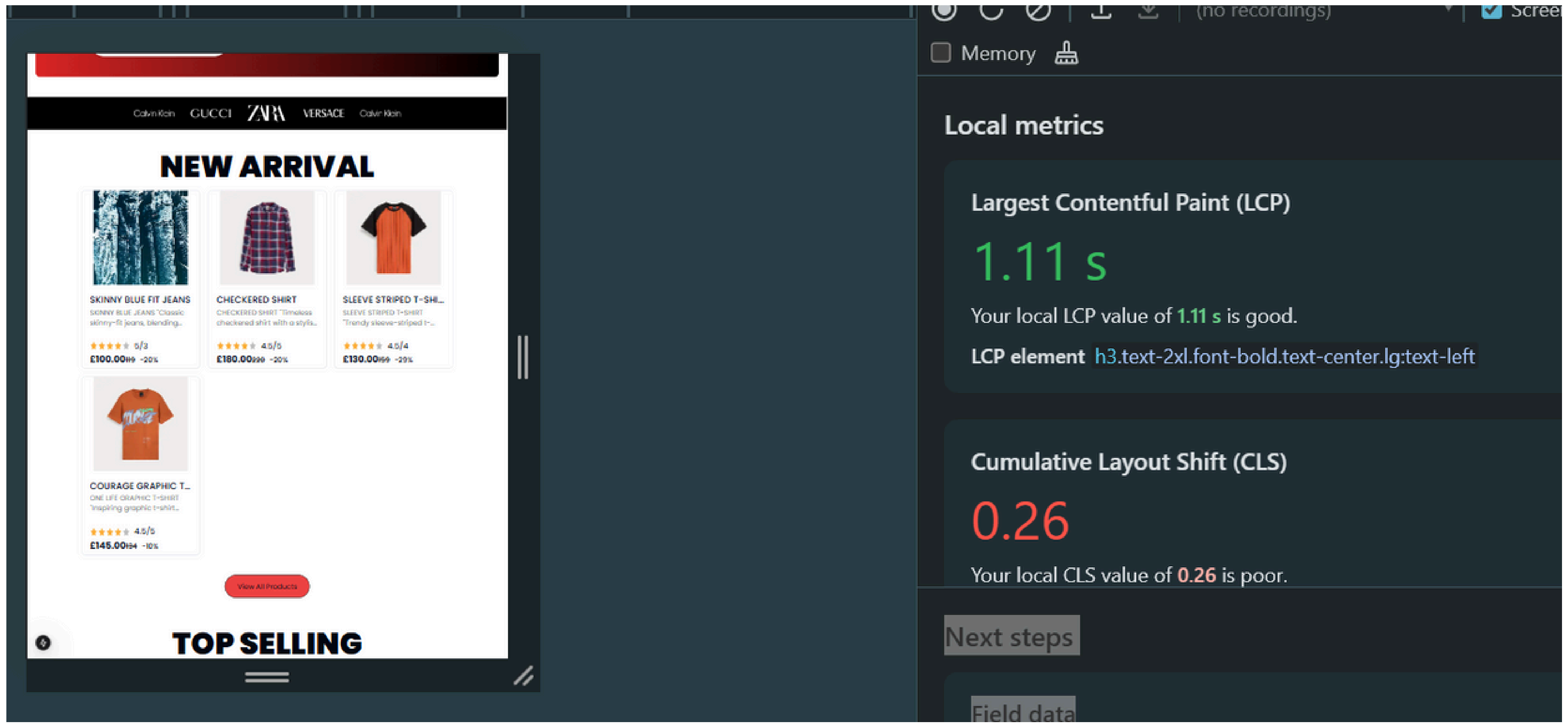
2. Invalid or Missing Data:
   - If product data fails to fetch from the API, an error message is shown: "Error: Unable to load product data. Please try again later."
3. Unexpected Server Errors
   - For server-side errors, a generic message is displayed: "An unexpected error occurred. Please try again later."
4. Non-Existent Pages
   - A custom 404 page is implemented with the message: "404 - This page could not be found."
5. Form Validation Errors :
   - Error messages are displayed for empty mandatory fields or invalid inputs: "Please fill out this field correctly."

Step 3: Performance Optimization

During testing, the website's performance was analyzed using Lighthouse in Google Chrome DevTools. The analysis revealed the following metrics:

Performance Results:

- Total Loading Time: 9,754ms (Needs Improvement).
- LCP (Largest Contentful Paint) : 1.10s (Good).
- CLS (Cumulative Layout Shift) : 0.26 (Poor).



Local metrics

Largest Contentful Paint (LCP)

1.11 s

Your local LCP value of 1.11 s is good.

LCP element h3.text-2xl.font-bold.text-center.lg:text-left

Cumulative Layout Shift (CLS)

0.26

Your local CLS value of 0.26 is poor.

Observations:

- High total loading time primarily caused by heavy JavaScript execution (6,135ms).
- Opportunities for improvement in script optimization and resource loading.

Planned Improvements:

1. Optimize JavaScript:
   - Enable bundle splitting to load only required code for each page.
   - Minify and compress JS files using tools like Terser and Gzip.
   - Eliminate unused JavaScript to reduce payload size.
2. Optimize Images:
   - Use Next.js <Image /> for lazy loading and format optimization (e.g., WebP).
   - Compress images with tools like TinyPNG before uploading.
3. Improve API Performance:
   - Reduce payload size by fetching only necessary fields.
   - Cache API responses to minimize repeated network calls.
4. Enhance Rendering:
   - Implement Server-Side Rendering (SSR) or Static Site Generation (SSG) for faster load times.
   - Use React's memo and useCallback to avoid unnecessary re-renders.
5. Reduce Render-Blocking Resources:
   - Use async/defer for non-critical scripts.
   - Inline critical CSS to reduce blocking during page rendering.

Outcome:

The above improvements aim to significantly reduce the loading time, enhance performance metrics, and provide a seamless user experience.

Step 4: Cross-Browser and Device Testing

Extensive testing was performed to ensure compatibility across different browsers and devices. This included:

- Browser Testing: Verified functionality on Chrome, Firefox, Edge, and Safari.
- Device Testing: Tested responsiveness and usability on desktops, tablets, and mobile devices.
- Manual Mobile Testing: Conducted hands-on testing on mobile devices to ensure smooth navigation and proper layout rendering.

The application performed perfectly, with excellent responsiveness and consistent behavior across all tested platforms.

Testing Report

Summary

Test Area	Test Type	Result
Functional Testing	Core Features	Passed
Error Handling	Error Scenarios	Passed
Performance Optimization	Initial Lighthouse	Needs Improvement (9,754ms)
Cross-Browser Testing	Compatibility Checks	Passed
Device Responsiveness	Mobile/Tablet Tests	Passed