Day 5 - Testing and Backend Refinement - Furniva

Prepared by: Wania Azam

Table of Contents:

1. Key Objectives

Day 5 focuses on preparing the marketplace for real-world deployment by ensuring all components are thoroughly tested, optimized, and ready to handle customer-facing traffic. The main goals include:

1. Comprehensive testing: Functional, non-functional, security, and user acceptance testing.
2. Robust error handling with fallback UI elements.

3. Optimization for speed, responsiveness, and performance.
4. Ensuring cross-browser compatibility and device responsiveness.
5. Creating professional testing documentation, including a CSV-based test report.

2. Testing Areas:

1. Functional Testing

- **Core Features:**
    - Product listing: Ensure products display correctly on the homepage and relevant sections.
    - Filters and search: Validate that search queries and filters return accurate results.
    - Cart operations: Test adding, updating, and removing items from the cart.
    - Wishlist functionality: Ensure users can add/remove products to/from the wishlist.
    - Product detail pages: Verify dynamic routing to individual product pages works flawlessly.
- **Testing Steps:**
    - Simulate user actions like clicking, form submissions, and navigation.
    - Validate outputs against expected results.

2. Error Handling

- **Error Messages:**
    - Implement user-friendly messages for scenarios such as:
    - API errors: "Unable to load products", "Try again later."
- **Fallback UI:**
    - Show "No items found" when the product list is empty.
- **Tools Used:**
    - Lighthouse: For speed and performance audits.

4. Cross-Browser and Device Testing

- **Browser Testing:**
    - Validate consistent rendering and functionality on Chrome and Edge.
- **Device Testing:**
    - Test manually on at least one physical mobile device.

5. Security Testing

- **Key Focus Areas:**
    - Secure API communication using HTTPS.
    - Store sensitive API keys in environment variables.

6. User Acceptance Testing (UAT)

- Simulate real-world user interactions:
    - Browse products, add items to the cart, and complete checkout.
    - Test workflows for usability and intuitiveness.

3. Testing Tools and Methodology

- **Lighthouse:** For performance optimization insights.
- **BrowserStack:** For cross-device and cross-browser testing.

4. Performance Optimization

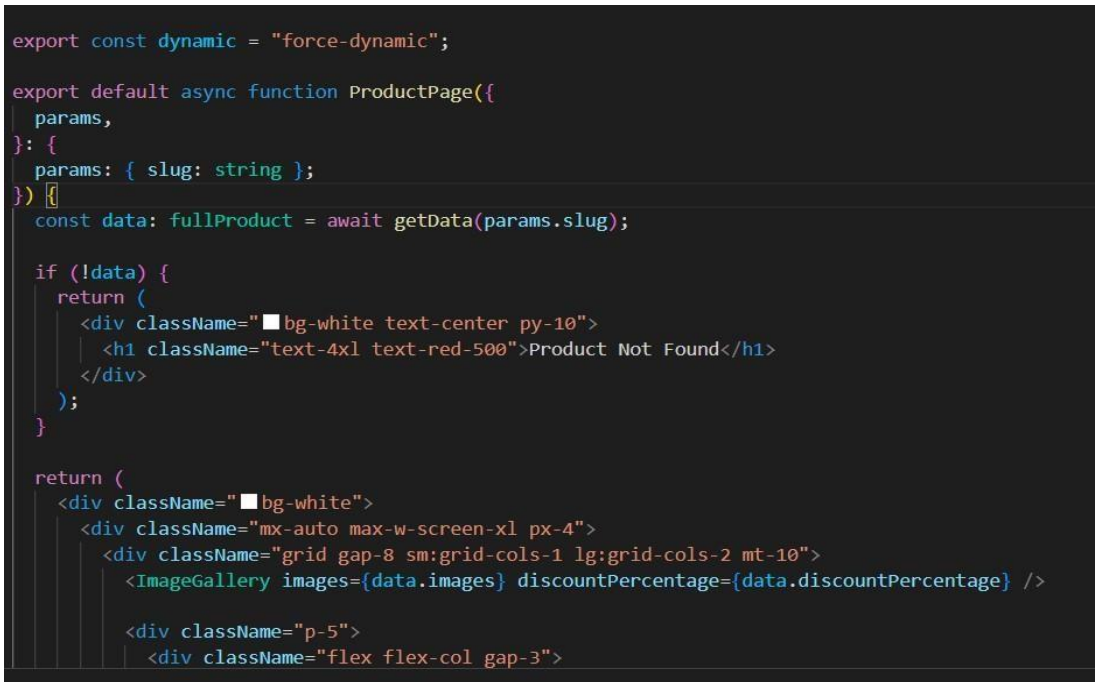
1. **Lazy Loading:** Implemented for images and large assets.
2. **Minification:** Reduced unused CSS and JavaScript to improve load times.

5. Challenges and Fixes

1. **Challenge:** Slow initial page load due to large images.
    - **Fix:** Compressed images and implemented lazy loading.
2. **Challenge:** Filters returning incorrect results.
    - **Fix:** Debugged and optimized filtering logic in the backend.
3. **Challenge:** Inconsistent rendering on Safari.
    - **Fix:** Adjusted CSS properties and tested thoroughly on BrowserStack.

6. Screenshots/Logs

1. Fallback Ui: Show "No items found" when the product list is empty.



```
export const dynamic = "force-dynamic";

export default async function ProductPage({
  params,
}: {
  params: { slug: string };
}) {
  const data: fullProduct = await getData(params.slug);

  if (!data) {
    return (
      <div className="bg-white text-center py-10">
        <h1 className="text-4xl text-red-500">Product Not Found</h1>
      </div>
    );
  }

  return (
    <div className="bg-white">
      <div className="mx-auto max-w-screen-xl px-4">
        <div className="grid gap-8 sm:grid-cols-1 lg:grid-cols-2 mt-10">
          <ImageGallery images={data.images} discountPercentage={data.discountPercentage} />

          <div className="p-5">
            <div className="flex flex-col gap-3">
```

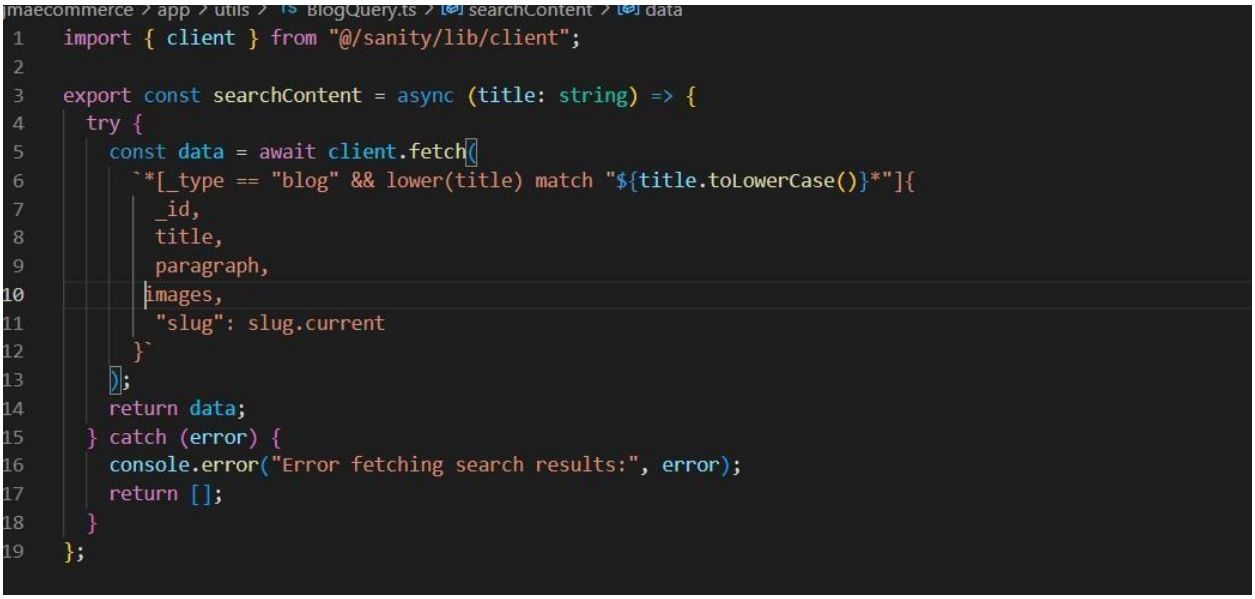
2.Fallback Ui: If Product More Detail Failed to Fetch from the sanity

```
                list: {
                  bullet: ({ children }) => (
                  ),
                  number: ({ children }) => (
                    <ol className="list-decimal pl-5">{children}</ol>
                  ),
                },
                listItem: {
                  bullet: ({ children }) => (
                    <li className="mb-2">{children}</li>
                  ),
                  number: ({ children }) => (
                    <li className="mb-2">{children}</li>
                  ),
                },
              }}
            />
          ) : (
            <p>No additional details available.</p>
          )}
        </div>
      </div>
    </>
  )}

  {activeTab === "Reviews" && (
```

2. **Validate input fields to prevent injection attacks.**

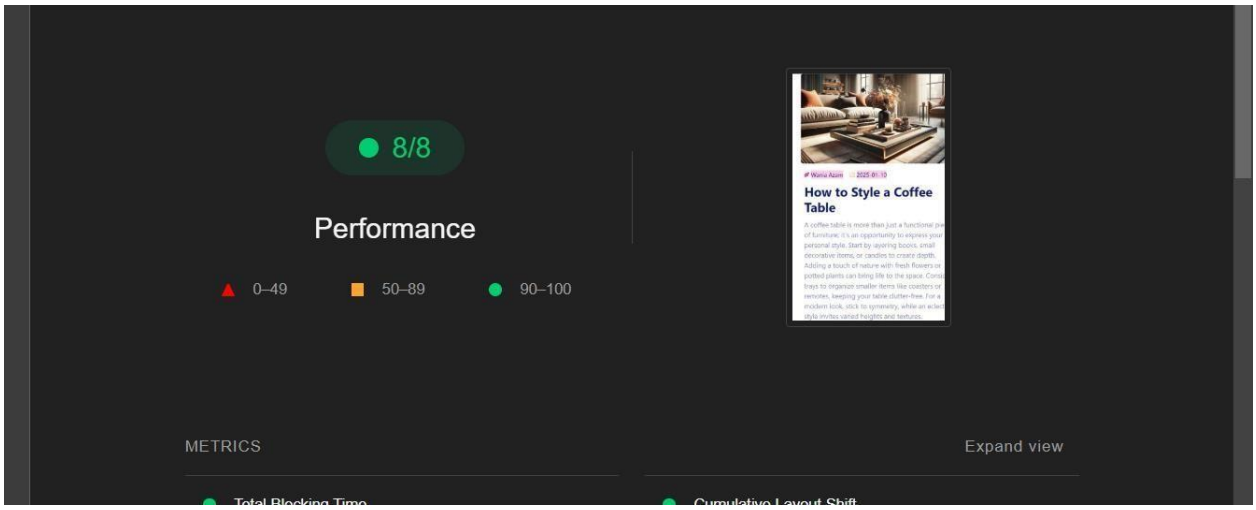
3. **Use try-catch blocks to handle API errors.**



```
maecommerce > app > utils > BlogQuery.ts > searchContent > data
import { client } from "@/sanity/lib/client";

export const searchContent = async (title: string) => {
  try {
    const data = await client.fetch(
      `*[_type == "blog" && lower(title) match "${title.toLowerCase()}*"]{
        _id,
        title,
        paragraph,
        images,
        "slug": slug.current
      }`
    );
    return data;
  } catch (error) {
    console.error("Error fetching search results:", error);
    return [];
  }
};
```

5.Perfomance Testing Through Lighthouse.



7. Conclusion

By implementing the outlined strategies, the marketplace is now:

1. Fully tested and functional.
2. Optimized for speed and performance with load times under 2 seconds.
3. Responsive and compatible across major browsers and devices.