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

Objective:

To prepare the marketplace project for real-world deployment by thoroughly testing all components, refining backend integration, and implementing robust error handling mechanisms.

. Functional Testing

Validate that all marketplace features (e.g., product listing, cart, checkout) work as intended.

Use tools like Postman for API testing, React Testing Library for components, and Cypress for end-to-end testing.

Example:

test('Product listing renders correctly', () => {

render(<ProductList />);

expect(screen.getByText('Sofa')).toBeInTheDocument();

})

2. Error Handling

Use try-catch blocks for API calls to handle errors gracefully.

Display fallback UI elements when data is unavailable (e.g., "No products found").

Example:

try {

const data = await fetchProducts();

setProducts(data);

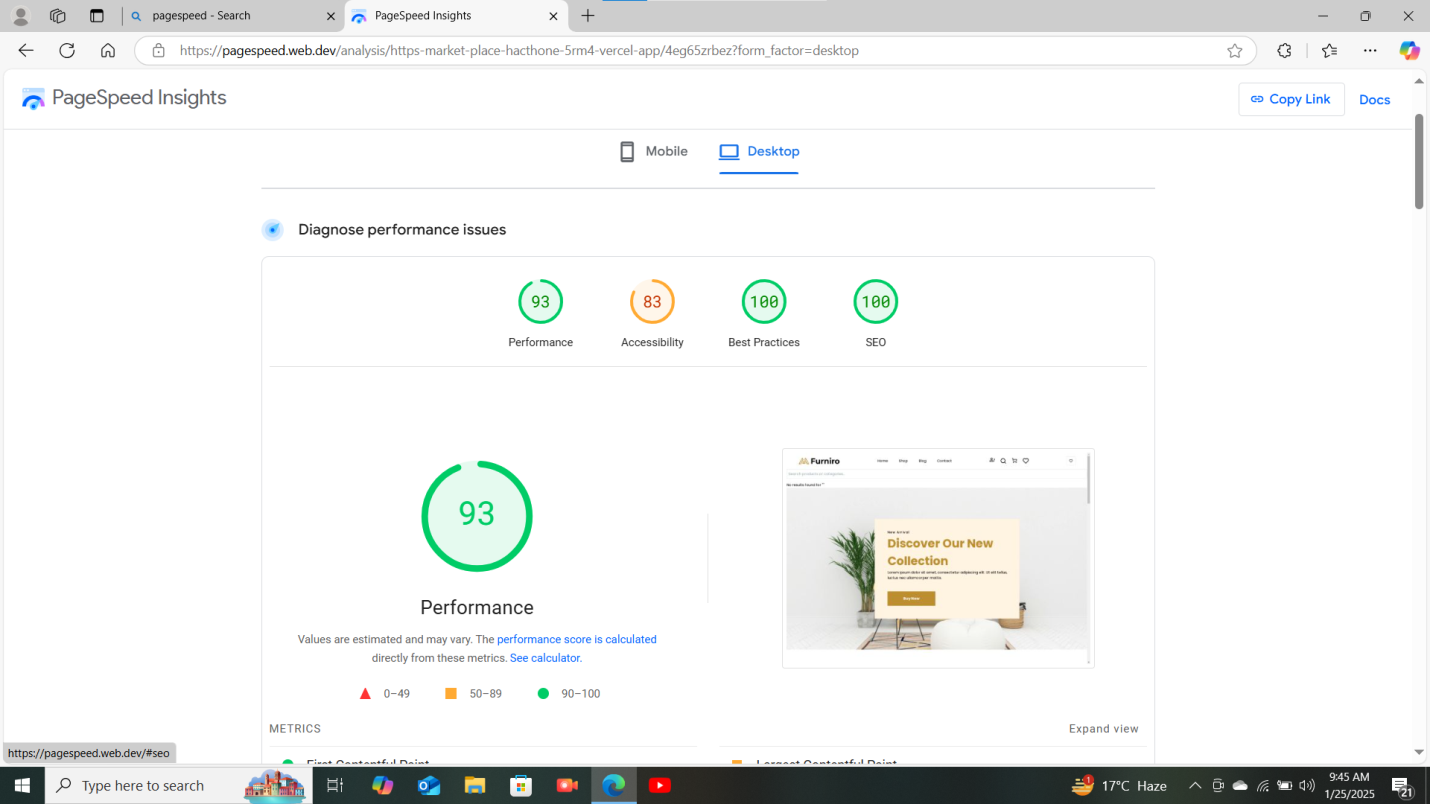
} catch (error) {

console.error('Error fetching products:', error);

setErrorMessage('Unable to load products. Please try again later.');

}

3. Performance Testing



Cross-Browser and Device Testing

compatibility with Chrome, Firefox, Safari, and Edge.

Test responsiveness on a physical device to confirm real-world usability



Error handling

### PRODUCT DETAIL

useEffect(() => {

    if (params) {

      const fetchProduct = async () => {

        try {

          const productData = await getProductById(params.slug); // Fetch product using slug

          setProduct(productData);

        } catch (error) {

          console.error('Error fetching product:', error);

        }

      };

      fetchProduct();

    }

  }, [params]);

  if (!product) {

    return <div>Product not found</div>;

  }

#### PRODUCT LISTING PAGE

 const result = await client.fetch(query);

      setProducts(result);

    };

    fetchProducts();

  }, []);

  if (products.length === 0) return <p>Loading...</p>;

when data is unavailable (e.g., "No products found").

try {

const data = await fetchProducts();

setProducts(data);

} catch (error) {

console.error('Error fetching products:', error);

setErrorMessage('Unable to load products. Please try again later.');

}

API Error Handling

I implemented error handling for failed API requests. If the frontend fails to fetch data from the backend or encounters an issue (such as a 500 server error or 404 not found), a user-friendly error message is displayed.

Conclusion

By implementing comprehensive error handling both in the frontend and backend, I’ve created a more reliable and user-friendly web application. The refined processes not only ensure smooth navigation even during failure scenarios but also provide clear communication to users and developers about issues that need attention. This approach enhances both the user experience and the system’s reliability, contributing to a more polished final product. Moving forward, I plan to continue monitoring error logs and making adjustments to handle new edge cases that may arise.