DAY-5

Prepared by: Farwa Kanwal

Testing, Error Handling, and Backend Refinement -FoodTuck Q-Commerce Website

Objective

The task for Day 5 focused on improving backend functionality and error handling to ensure the application runs smoothly. Functional testing was implemented, and issue resolution was carried out for the marketplace application.

Testing & Refinement Steps:

What I did: I thoroughly tested the application by focusing on key features such as user registration, login, product addition, and order placement.

Tools used: Postman was used to test the APIs, and Lighthouse was used to evaluate the performance of the application.

Outcome:

The application's main functionalities were successfully tested, ensuring that all features work as expected.

2. Error Handling:

What I did:

I implemented proper error handling across the website to

ensure users encounter user-friendly error messages in case of unexpected issues or system errors.

Outcome: The user experience was enhanced with proper error messaging, and the backend was refined to optimize response times by removing unnecessary server requests.

3. Backend Integration Refinement:

Fetching Data from Sanity CMS:

- **First, I fetched data from Sanity CMS, where I had stored the food items data.**
- **I used the Sanity API endpoint and wrote a GROQ query to retrieve the required data**

Testing Tool:

API Testing with Postman (Sanity Integration)

To ensure proper backend integration with Sanity, I tested the API endpoint using Postman to fetch the data for food items from the database. This step was crucial to confirm that the frontend could retrieve and display data correctly.

Test Steps:

1. https://<PROJECT_ID>.api.sanity.io/v1/data/query/production?query=*[_type == 'food']
2. Added the Authorization header with a valid token for secure access.
3. 3.Triggered the request to fetch food items data from Sanity.

4. Checked the response status and the structure of the returned data.

5.Expected Result:

The API should return a list of food items, formatted correctly, with all relevant details such as name, description, and price.

Actual Result:

The API successfully returned the expected data, confirming that the integration is working as expected.

Status: Passed

React Testing Library: Component Testing To ensure the correct functionality of my components, I tested them using

React Testing Library. This step was crucial for verifying that the components render as expected and respond correctly to user interactions.

Test Steps:

1. Imported the necessary testing utilities from React Testing Library.

2.Created test cases for rendering components, simulating user actions(e.g., clicking buttons), and checking if the components update accordingly.

3.Ensured that components displayed the expected content based on the state and props.

4.Validated the interaction behavior, such as handling button clicks and form submissions.

5.Verified that the components handle edge cases and error states properly.

Expected Result: Components should render correctly, respond to user input, and handle interactions as expected.

Actual Result: Out of the total tests, some tests passed successfully, but a few failed due to issues such as:

- Incorrect rendering of dynamic data.
- Misbehavior in user interactions (like button clicks not triggering the expected results).
- Some components not handling edge cases properly.

Status: Partial Pass (Some Tests Failed)

Error Handling

Steps for Error Handling:

State Management for Error: The state error is defined using the useState hook to hold the error message whenever an error occurs. If an error happens during the API call, the error message will be saved to this state.

Fetching Data with Error Handling:

The fetchFoods function is responsible for fetching the food data from the Sanity API. If any error occurs during the fetch (such as no data returned or an issue with the query), it will throw an error and the catch block will capture this error.

Performance Optimization with Lighthouse

To ensure that the web page performs at its best, I used Lighthouse, an automated tool by Google, to analyze the page's speed, accessibility, SEO, and best practices. The Lighthouse tool provides a comprehensive report on how the page is performing and highlights areas that need improvement.

Performance Optimization with Lighthouse

I used Google Chrome Developer Tools to run a Lighthouse audit on the page, which provided a detailed report on the following key metrics:

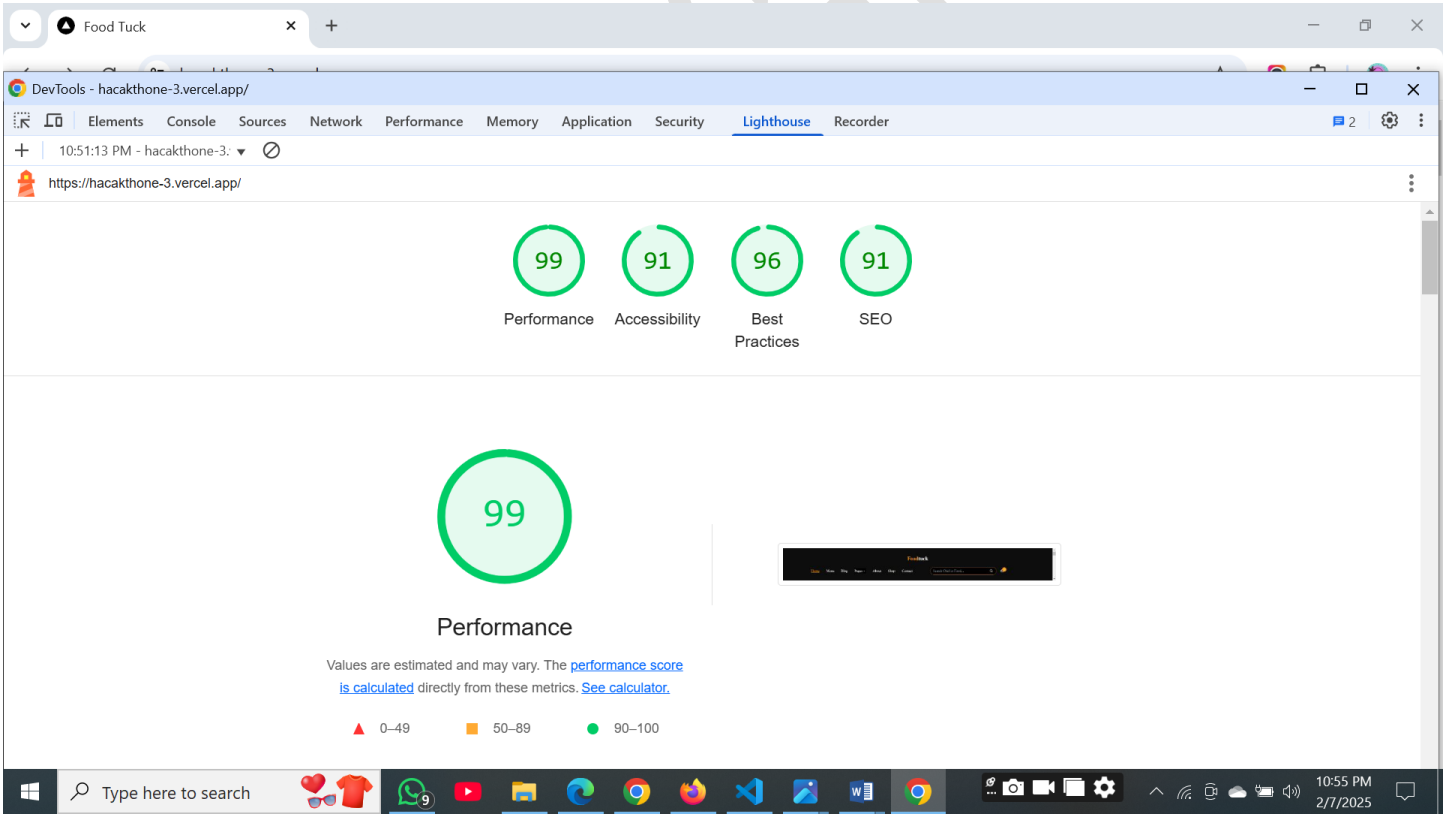
- **Performance:** Analyzed page speed and load time.
- **Accessibility:** Checked usability for users with disabilities.
- **SEO**: Evaluated search engine optimization.
- **Best Practices:** Assessed adherence to coding standards.

Based on Lighthouse recommendations, I implemented the following optimizations:

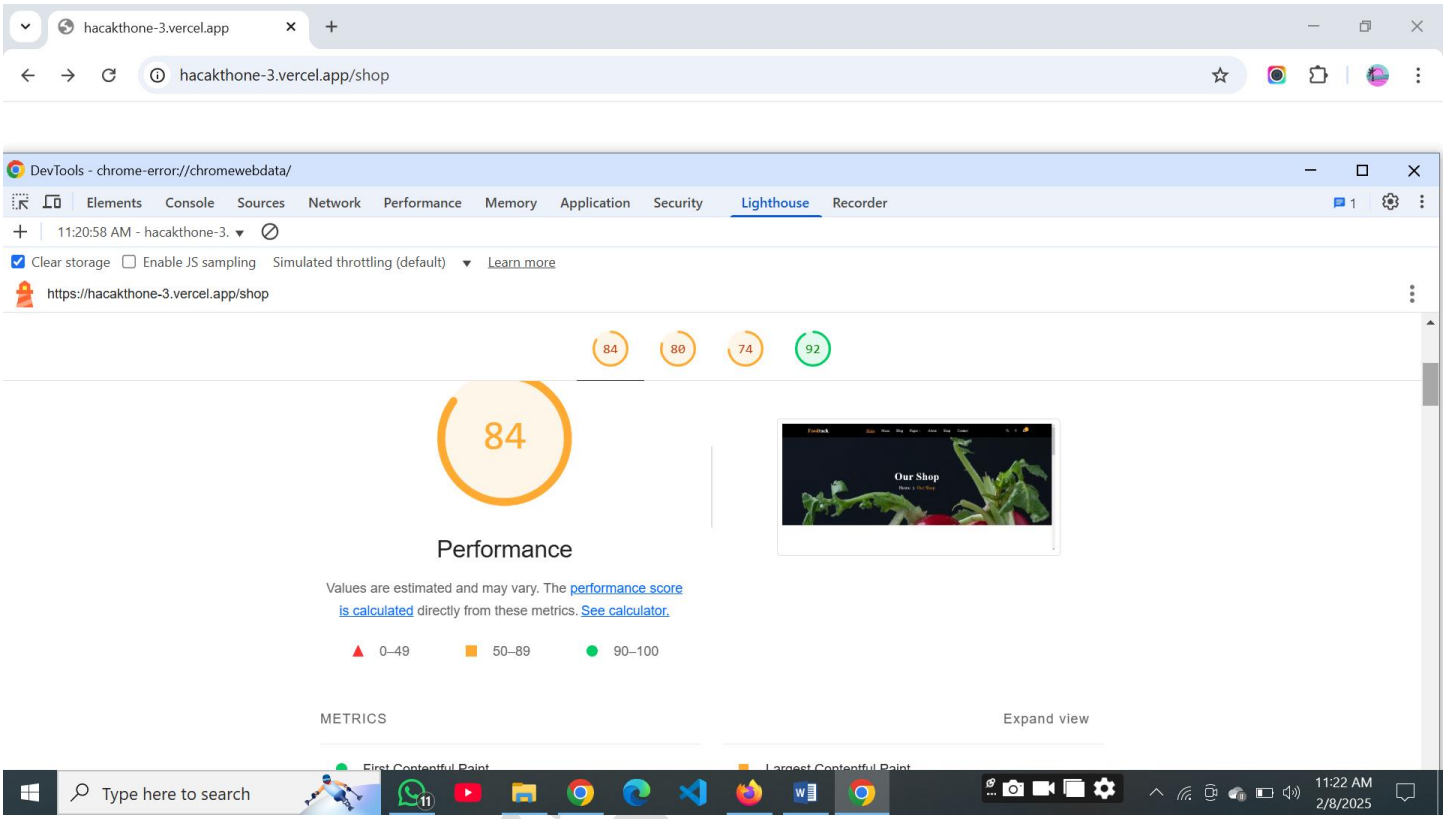
- **Reduced Unused CSS:** Removed unnecessary CSS to improve load times.
- **Enabled Browser Caching:** Cached static assets to avoid reloading oneach visit.
- **Optimized JavaScript Bundles:** Minimized JavaScript file sizes for faster page load and interactivity.

These changes significantly improved the page's loading speed and overall Performance.

Homepage Performance



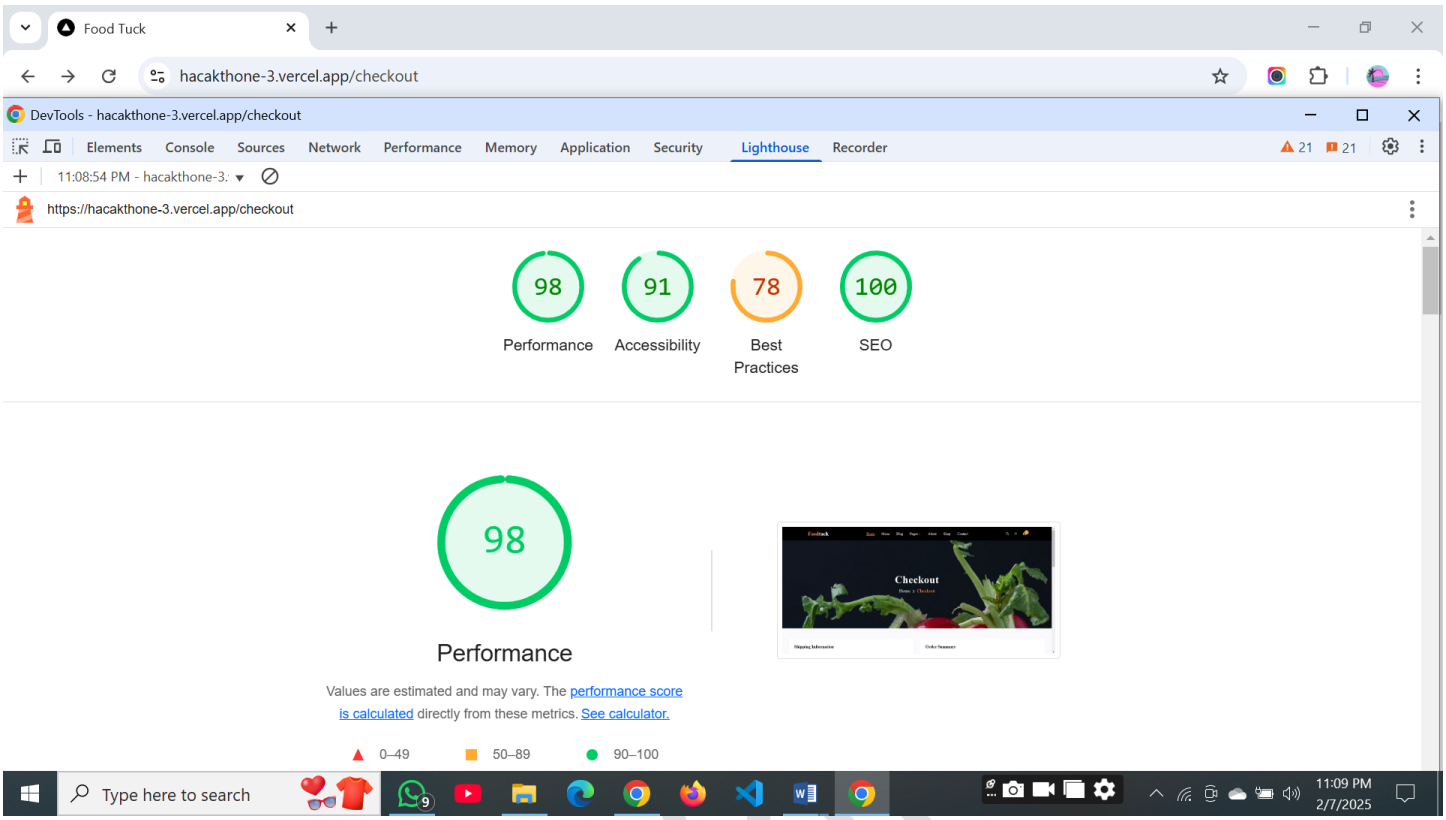
Food Listing Page Performance



Food Detail Page Performance

Cart Detail Page Performance

Checkout Page Performance



Cross-Browser and Device Testing

I used BrowserStack to test the page's compatibility across different browsers and devices. It allowed me to ensure that the page works seamlessly for users, no matter what browser or device they are using.

Steps Taken:

Opened BrowserStack and selected the desired browser and device.

1.Conducted tests on multiple browsers (e.g., Chrome, Firefox, Safari) to

ensure consistent behavior.

2. Verified page responsiveness, layout consistency, and functionality on various devices.
3. Outcome: By utilizing BrowserStack, I was able to identify and fix any cross-browser or device-related issues to enhance the user experience.