LIGHTHOUSE TEST LINK

https://lighthouse-metrics.com/lighthouse/checks/d8254cfb-6625-4f9d-8d1c-937a524f6645

Documentation Report

1. Introduction

This report provides a detailed overview of the testing, performance optimization, security measures, and challenges faced during the development and deployment of the **Product Listing Application**. The application is built using **Next.js**, **Sanity CMS**, and **Vercel** for deployment. The goal was to ensure seamless integration with Sanity, optimize performance, and implement robust security measures.

---

2. Test Cases Executed and Results

2.1 Test Case Overview

The following test cases were executed to ensure the application functions as expected:

Test Case ID	Description	Status	Remarks
TC001	Verify `NEXT_PUBLIC_BASE_URL` is correctly set	Passed	Environment variable updated successfully.
TC002	Update Sanity CORS Origins	Passed	Vercel URL added to Sanity CORS origins.
TC003	Verify Sanity Client Configuration	Passed	Sanity client configured correctly.
TC004	Test Fetching Products from Sanity	Passed	Products fetched successfully.
TC005	Check for CORS Errors	Passed	No CORS errors observed.
TC006	Verify Environment Variables in Vercel	Passed	Environment variables set correctly.

TC007	Test Application Deployment	Passed	Application deployed successfully.

2.2 Detailed Test Case Results

TC001: Verify `NEXT_PUBLIC_BASE_URL` is Correctly Set

- **Steps**:
    1. Check `.env` file for `NEXT_PUBLIC_BASE_URL`.
    2. Verify the URL is accessible.
- **Expected Result**: `NEXT_PUBLIC_BASE_URL` is set to a valid web link (e.g., Vercel URL).
- **Actual Result**: `NEXT_PUBLIC_BASE_URL` is correctly set to the Vercel URL.
- **Remarks**: Environment variable updated successfully.

TC002: Update Sanity CORS Origins

- **Steps**:
    1. Go to Sanity dashboard.
    2. Add Vercel URL to CORS origins.
    3. Enable "Allow credentials."
- **Expected Result**: Vercel URL is added to Sanity CORS origins, and credentials are allowed.
- **Actual Result**: Vercel URL is added to Sanity CORS origins, and credentials are allowed.
- **Remarks**: CORS settings updated successfully.

TC003: Verify Sanity Client Configuration

- **Steps**:
    1. Check Sanity client configuration in code.
    2. Ensure environment variables are used correctly.
- **Expected Result**: Sanity client is configured with correct project ID, dataset, and API version.
- **Actual Result**: Sanity client is configured correctly.
- **Remarks**: Sanity client is working as expected.

TC004: Test Fetching Products from Sanity

- **Steps**:
    1. Run the `fetchProducts` function.
    2. Check network requests in the browser console.
- **Expected Result**: Products are fetched successfully from Sanity.

- **Actual Result**: Products are fetched successfully.
- **Remarks**: No errors in fetching data.

TC005: Check for CORS Errors

- **Steps**:
    1. Inspect browser console for CORS errors.
    2. Verify CORS settings in Sanity.
- **Expected Result**: No CORS errors in the browser console.
- **Actual Result**: No CORS errors observed.
- **Remarks**: CORS configuration is correct.

TC006: Verify Environment Variables in Vercel

- **Steps**:
    1. Check Vercel environment settings.
    2. Ensure all required variables are set.
- **Expected Result**: All environment variables are correctly set in Vercel.
- **Actual Result**: All environment variables are correctly set.
- **Remarks**: Environment variables are properly configured.

TC007: Test Application Deployment

- **Steps**:
    1. Deploy the application to Vercel.
    2. Verify the app is accessible.
- **Expected Result**: Application is deployed and accessible at the Vercel URL.
- **Actual Result**: Application is deployed and accessible.
- **Remarks**: Deployment succeeded without errors.

3. Performance Optimization Steps Taken

3.1 Use of Sanity CDN

- Sanity's built-in CDN was utilized to ensure fast and efficient data fetching. This significantly reduced the latency in retrieving product data.

3.2 Pagination

- Pagination was implemented to limit the number of products fetched at once. This improved the application's performance by reducing the load on the frontend.

3.3 Image Optimization

- The Next.js `Image` component was used to optimize image loading. This included lazy loading and serving images in modern formats (e.g., WebP).

3.4 Code Splitting

- Dynamic imports and code splitting were used to load components only when needed, reducing the initial load time of the application.

4. Security Measures Implemented

4.1 CORS Configuration

- The Vercel URL was added to Sanity's CORS origins to prevent unauthorized access. The "Allow credentials" option was enabled to ensure secure requests.

4.2 Environment Variables

- Sensitive data such as the Sanity project ID and API keys were stored in environment variables. This prevented exposure of sensitive information in the codebase.

4.3 HTTPS

- The application was deployed on Vercel, which automatically enforces HTTPS for all requests. This ensured secure communication between the client and server.

4.4 Input Validation

- All user inputs (e.g., search terms) were validated on the client and server sides to prevent injection attacks.

5. Challenges Faced and Resolutions Applied

5.1 Challenge: CORS Errors

- **Description**: Initially, the application faced CORS errors when fetching data from Sanity.
- **Resolution**: The Vercel URL was added to Sanity's CORS origins, and the "Allow credentials" option was enabled.

5.2 Challenge: Environment Variables Not Recognized in Vercel

- **Description**: Environment variables were not being recognized after deployment to Vercel.

- **Resolution**: The environment variables were verified and updated in Vercel's project settings.

5.3 Challenge: `useSearchParams` Hook Causing CSR Bailout Errors

- **Description**: The `useSearchParams` hook caused client-side rendering (CSR) bailout errors in Next.js.
- **Resolution**: The component was wrapped in a `Suspense` boundary to defer rendering until the necessary data was available.

5.4 Challenge: Fetching Products from Sanity

- **Description**: Products were not being fetched due to incorrect query filters.
- **Resolution**: The query filters were updated to ensure all products were fetched correctly.

6. Conclusion

The **Product Listing Application** was successfully developed, tested, and deployed. All test cases passed, and the application is fully functional. Performance optimizations and security measures were implemented to ensure a smooth and secure user experience. The challenges faced during development were resolved effectively, and the application is now ready for production use.