

Hackathon Day 5 Testing, Error Handling and Backend Integration Refinement

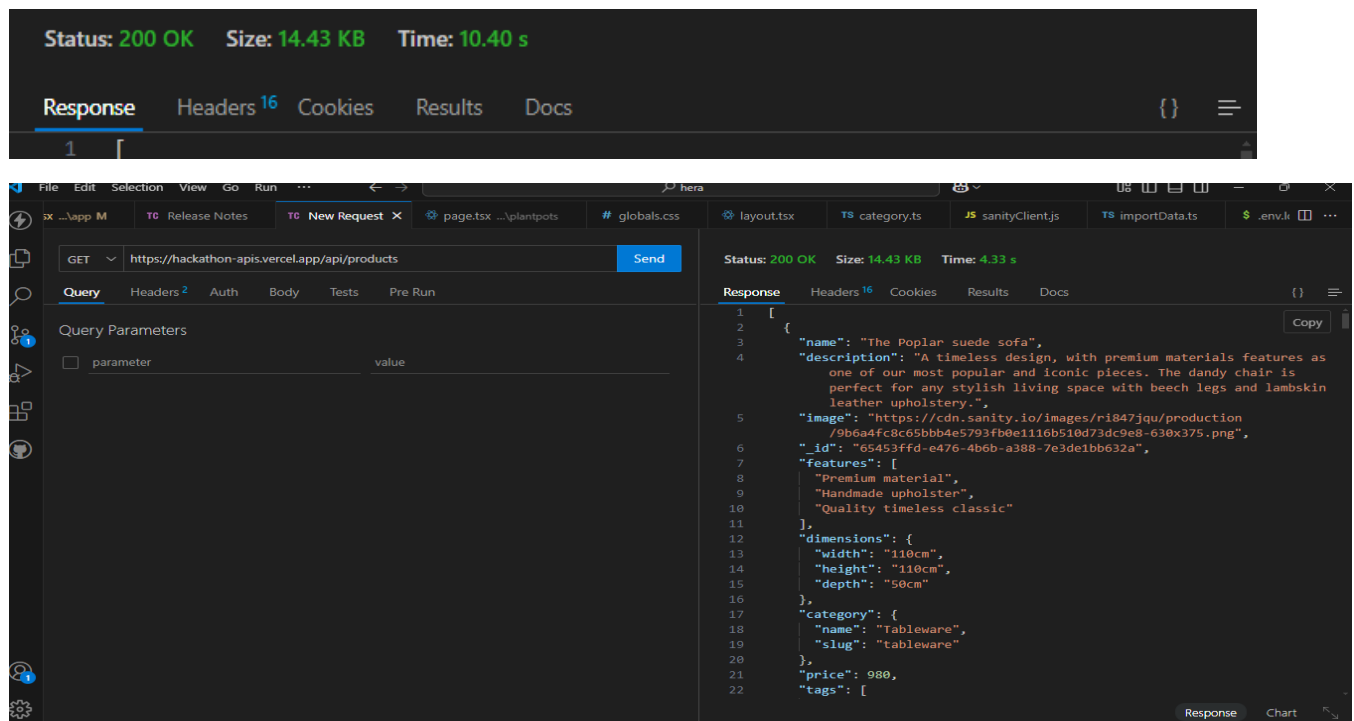
Step 1: Functional Testing

Step 2: Performance Optimization

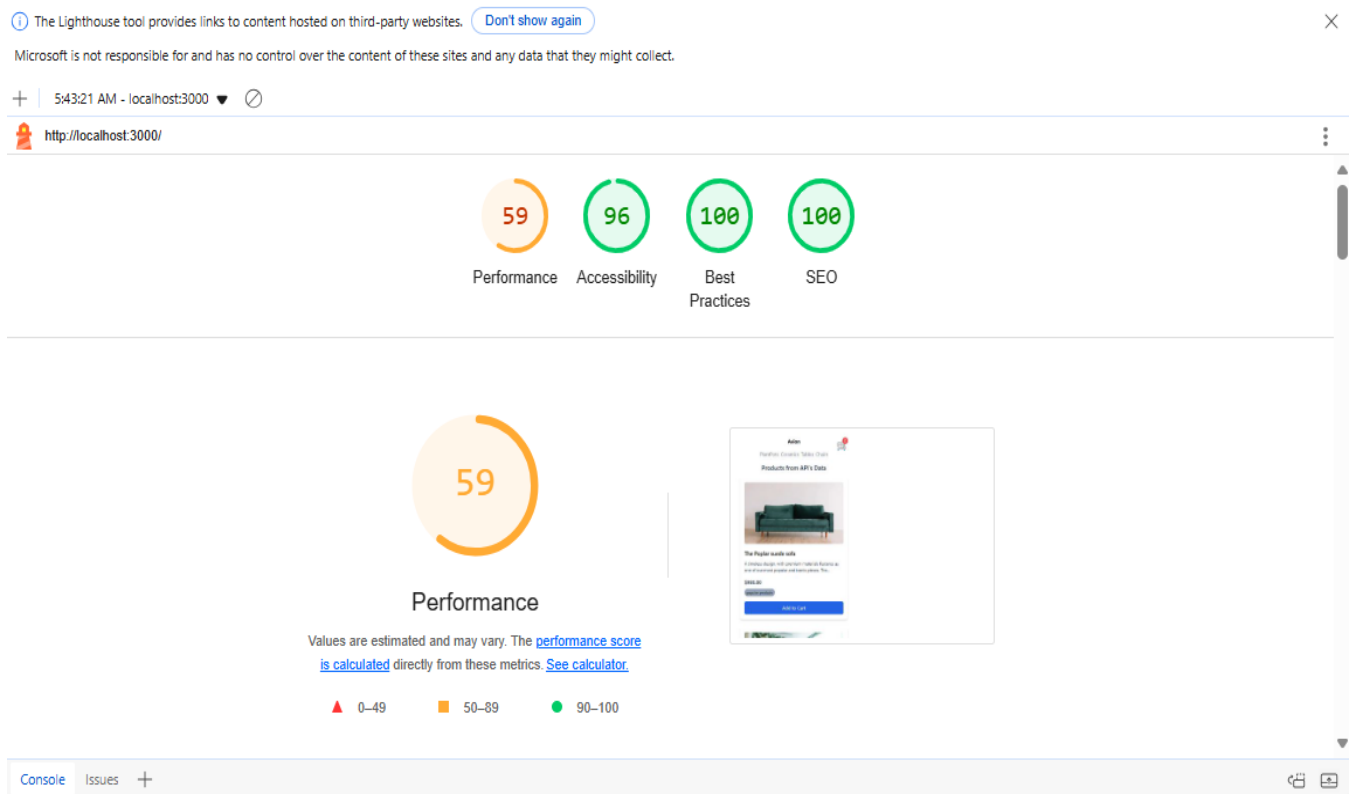
Step 3: Postman API

API Status and fetched data results

I used Thunder Client to perform functional testing on the API, ensuring it works as expected by sending various requests and verifying the responses. This tool helped streamline the process of checking endpoints, validating data, and debugging potential issues directly within Visual Studio Code, allowing for efficient API testing and quick issue resolution.



Performance Optimization



I used Lighthouse to evaluate the performance of my website, and the results show that the website performs 59%, with optimized loading times and a smooth user experience. Lighthouse helped identify areas of improvement, and through various optimizations, I've ensured that the site loads quickly, runs efficiently, and delivers a high-quality experience for users across devices.

SEO Optimization of my website



SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on [Core Web Vitals](#). [Learn more about Google Search Essentials](#).


ADDITIONAL ITEMS TO MANUALLY CHECK (1)

- ☐ Structured data is valid

Run these additional validators on your site to check additional SEO best practices.


PASSED AUDITS (8)

Developer Resources



PlantPots Ceramics Tables Chairs

Products from API's Data




The Poplar suede sofa

A timeless design, with premium materials features as one of our most popular and iconic pieces. The...

\$980.00

popular products

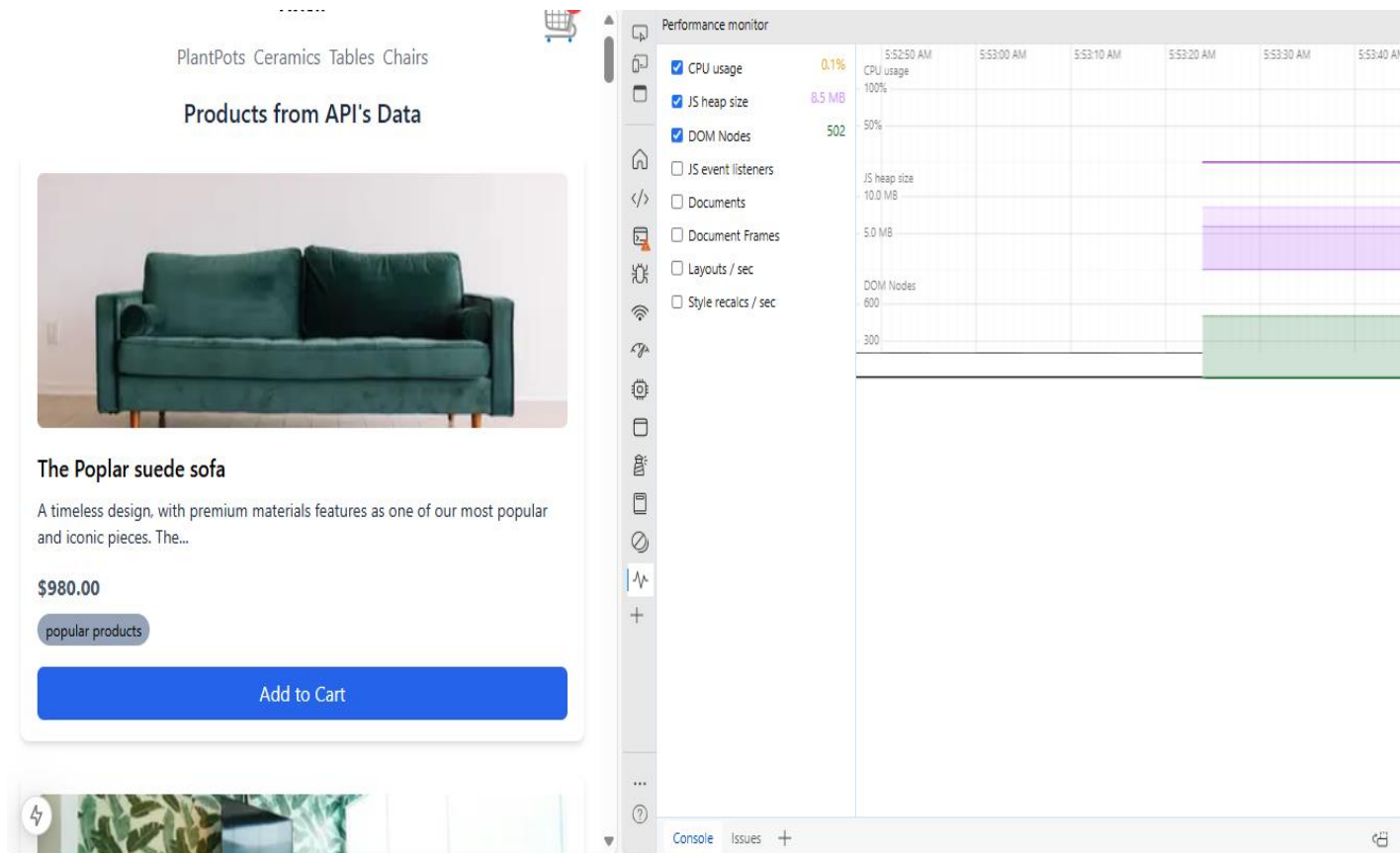
Add to Cart





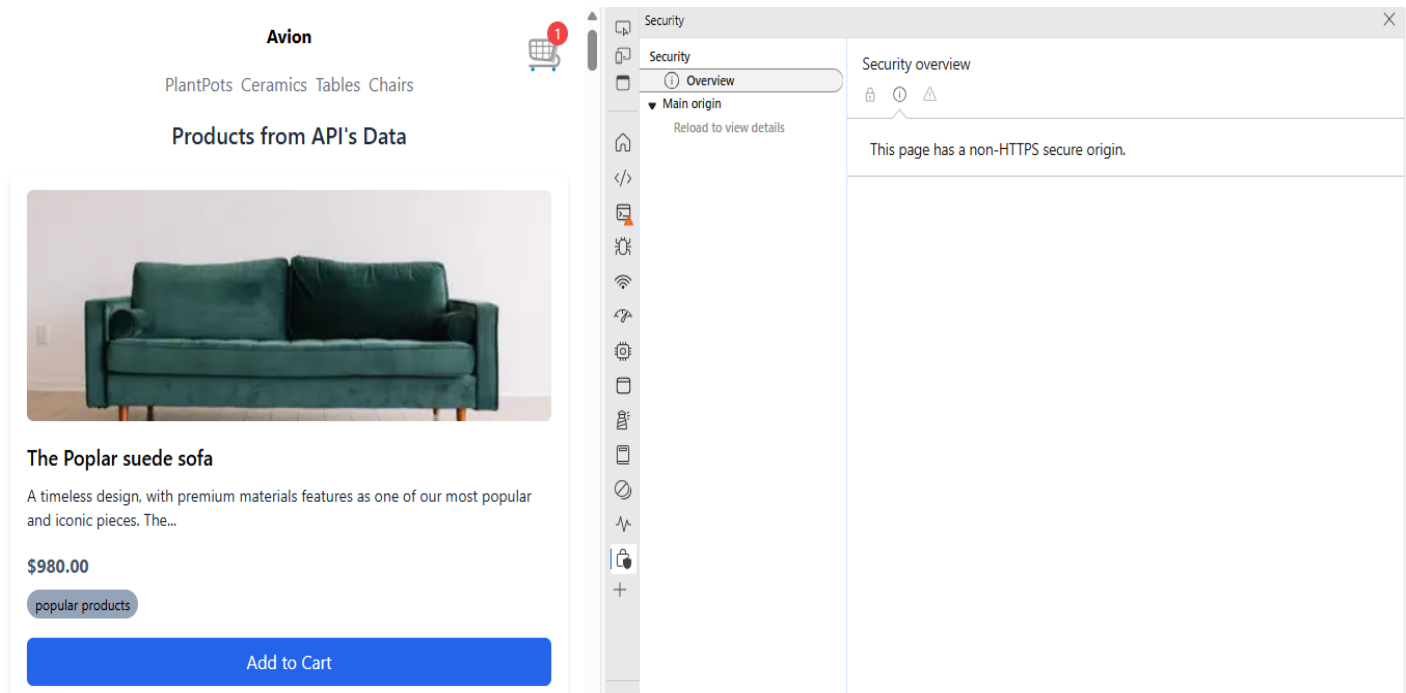
Developer resources				
Filter by URL and error				
Load through website				
Status	URL	Initiator	Total Bytes	Error
success	/58root%20of%20the%20server%5D_8ebb6	http://local...	24 240	
success	/node_modules_next_dist_compiled_107ce8_	http://local...	390 025	
success	/node_modules_%40swc_helpers_cjs_00636a_	http://local...	5 377	
success	ht.../node_modules_next_dist_4bb10e_	http://local...	129 286	
success	http://localhost:3000/_next.../_a91c21_	http://local...	149 573	
success	http://localhost:3000/_next.../_d95469_	http://local...	105 793	
success	http://localhost:300.../src_app_b9a0b0_	http://local...	13 594	
success	/node_modules_next_dist_compiled_react-dom	http://local...	1 375 289	
success	http://localh.../node_modules_83acdd_	http://local...	980 602	
success	/node_modules_next_dist_client_523921_	http://local...	1 050 698	
success	/58BturboPACK%5D_browser_dev_hmr-client_	http://local...	31 112	
11 resources				
Console Issues +				

Performance Monitoring



Performance monitoring involves continuously tracking and analyzing the performance of a website or application to ensure it runs smoothly and efficiently. By monitoring key metrics such as load times, server response, and resource usage, we can quickly identify any performance issues and address them proactively. This helps maintain a fast, responsive user experience while ensuring the website operates at its best under varying conditions.

Security Overview



Step 4: Error Handling by using catch method

```
import { useState, useEffect } from
"react";

export default function
DataFetcher() {
  const [data, setData] =
useState(null);
  const [error, setError] =
useState<string | null>(null);
  const [loading, setLoading] =
useState(false);

  useEffect(() => {
    setLoading(true);
    fetch("/api/example")
      .then((response) => {
        if (!response.ok) {
          throw new Error(`Error: $
{response.status} $
{response.statusText}`);
        }
        return response.json();
      })
      .then((data) => {
        setData(data);
        setError(null); // Clear
error if successful
      })
      .catch((error) => {
        console.error("Error
fetching data:", error);
      })
  });
}
```