Day 5 Hackathon Report: Testing and Backend Refinement 🚀

1. Functional Deliverables:

Lighthouse Performance Report

Overall Scores:

- **Performance:** 67
- **Accessibility:** 85
- **Best Practices:** 100
- **SEO:** 100

2. Key Findings and Recommendations:

Performance (67) - Needs Improvement 🔵

Potential Issues:

- Slow page load times due to unoptimized assets.
- Inefficient resource management leading to increased load times.
- Render-blocking scripts and styles delaying content display.

✓ **Recommendations:**

- Convert images to modern formats such as WebP and AVIF for better compression.
- Minify and compress CSS, JavaScript, and HTML to reduce file sizes.
- Implement lazy loading for images and iframes to improve load speeds.
- Remove unused JavaScript and CSS to optimize performance.
- Utilize server-side caching and integrate a Content Delivery Network (CDN) for faster asset delivery.

Accessibility (85) - Good, but Can Improve ▯

Potential Issues:

- Some text lacks sufficient contrast, reducing readability.
- Missing labels on interactive elements, which may hinder screen reader accessibility.

✓ **Recommendations:**

- Enhance color contrast to ensure readability for all users.
- Ensure all interactive elements have appropriate labels.
- Integrate ARIA attributes where necessary to improve accessibility.

Best Practices (100) - Excellent ✅

Strengths:

- Secure implementation, adhering to modern security standards.
- No detected vulnerabilities such as Cross-Site Scripting (XSS).
- Proper HTTPS usage, ensuring secure data transmission.

⚠️ **Minor Consideration:**

- Regularly update dependencies to prevent security vulnerabilities.

SEO (100) - Excellent ✅

Strengths:

- Optimized for search engines with structured metadata.
- Well-implemented structured data and mobile-friendly design.

⚠️ **Minor Consideration:**

- Continuously monitor and validate structured data to maintain search engine ranking.

---

3. Action Plan

1. **Optimize Performance:**
   - Compress and optimize assets for faster loading.
   - Implement lazy loading and efficient caching strategies.
   - Reduce render-blocking resources for a smoother user experience.
2. **Enhance Accessibility:**
   - Ensure all elements have proper labeling for screen readers.
   - Improve color contrast for better readability and inclusivity.
3. **Maintain Best Practices & SEO:**
   - Continue updating security measures and dependencies.
   - Regularly validate structured data and adapt to SEO trends.

---

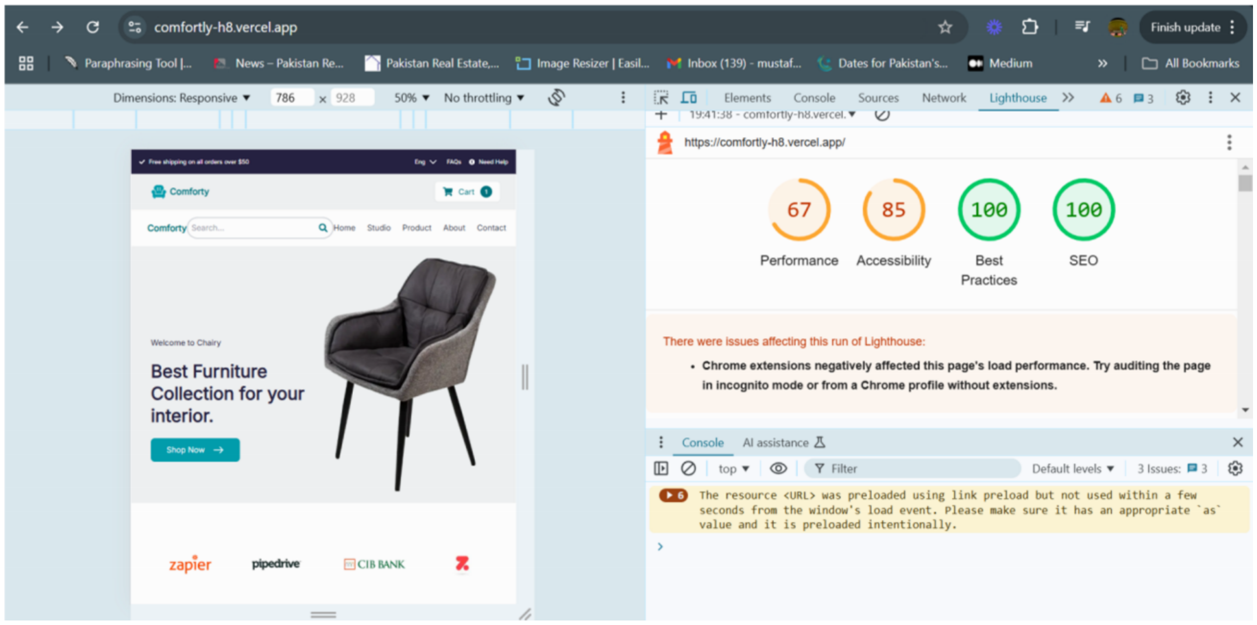
4. Screenshots & Visual Documentation

1. **API Calls in Action:** Console screenshot showcasing API request and response flow.

2. **Dynamic Data on Frontend:** Screenshot displaying real-time product and category data retrieval.
3. **Sanity CMS Fields Populated:** Screenshot confirming successful data migration and integration within the CMS.
4. **Code Implementation:** Screenshots illustrating core migration scripts and API integration logic.

Insights Gained

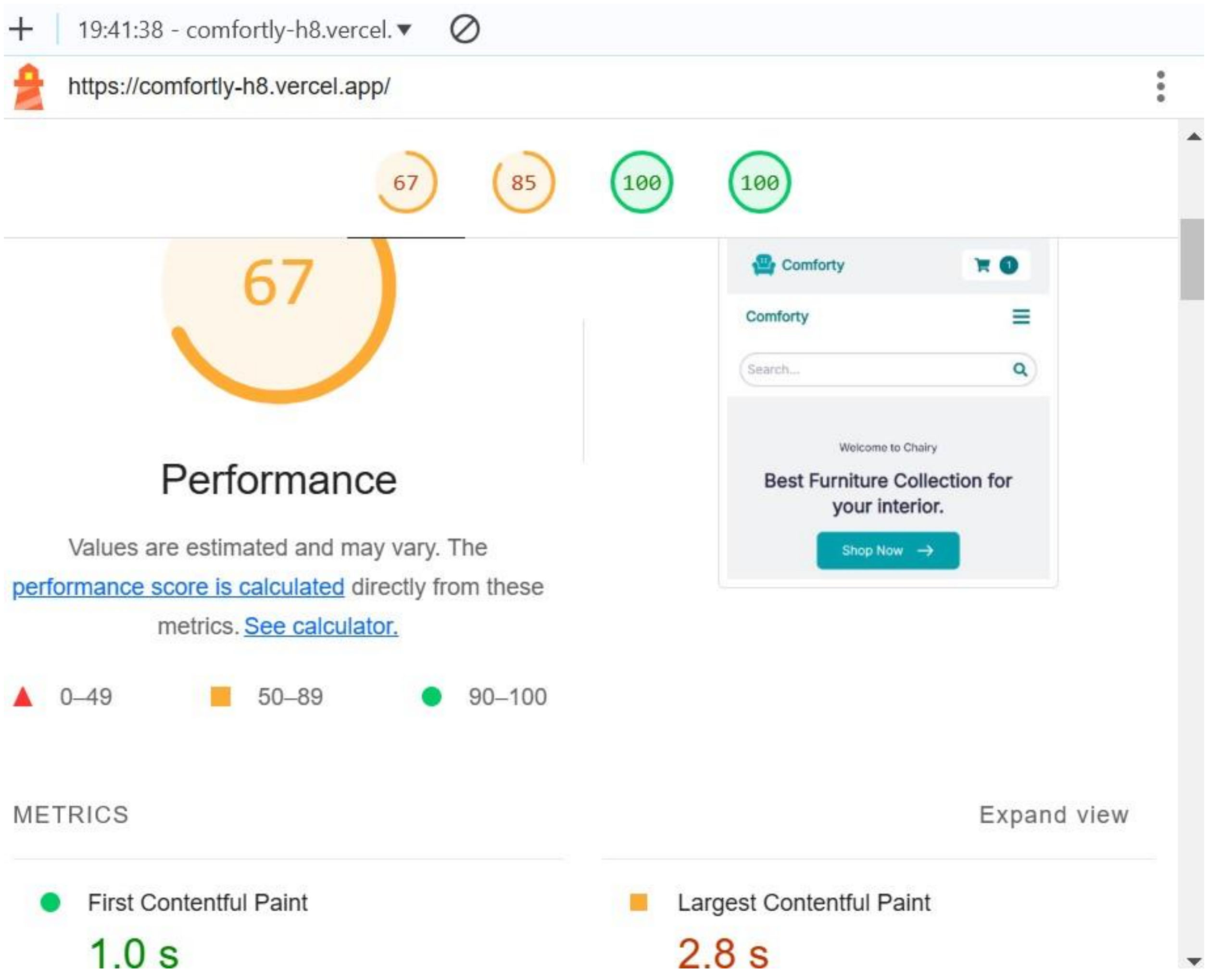
- Leveraging predefined schemas streamlined API integration and reduced development overhead.
- Automating data migration via scripts significantly cut down manual workload while ensuring consistency.
- Implementing proper data validation and structured mapping was crucial in maintaining a seamless user experience.



There were issues affecting this run of Lighthouse:

- Chrome extensions negatively affected this page's load performance. Try auditing the page in incognito mode or from a Chrome profile without extensions.

```
The resource <URL> was preloaded using link preload but not used within a few
seconds from the window's load event. Please make sure it has an appropriate `as`
value and it is preloaded intentionally.
```



19:41:38 - comfortly-h8.vercel.▼

https://comfortly-h8.vercel.app/

67 85 100 100

67

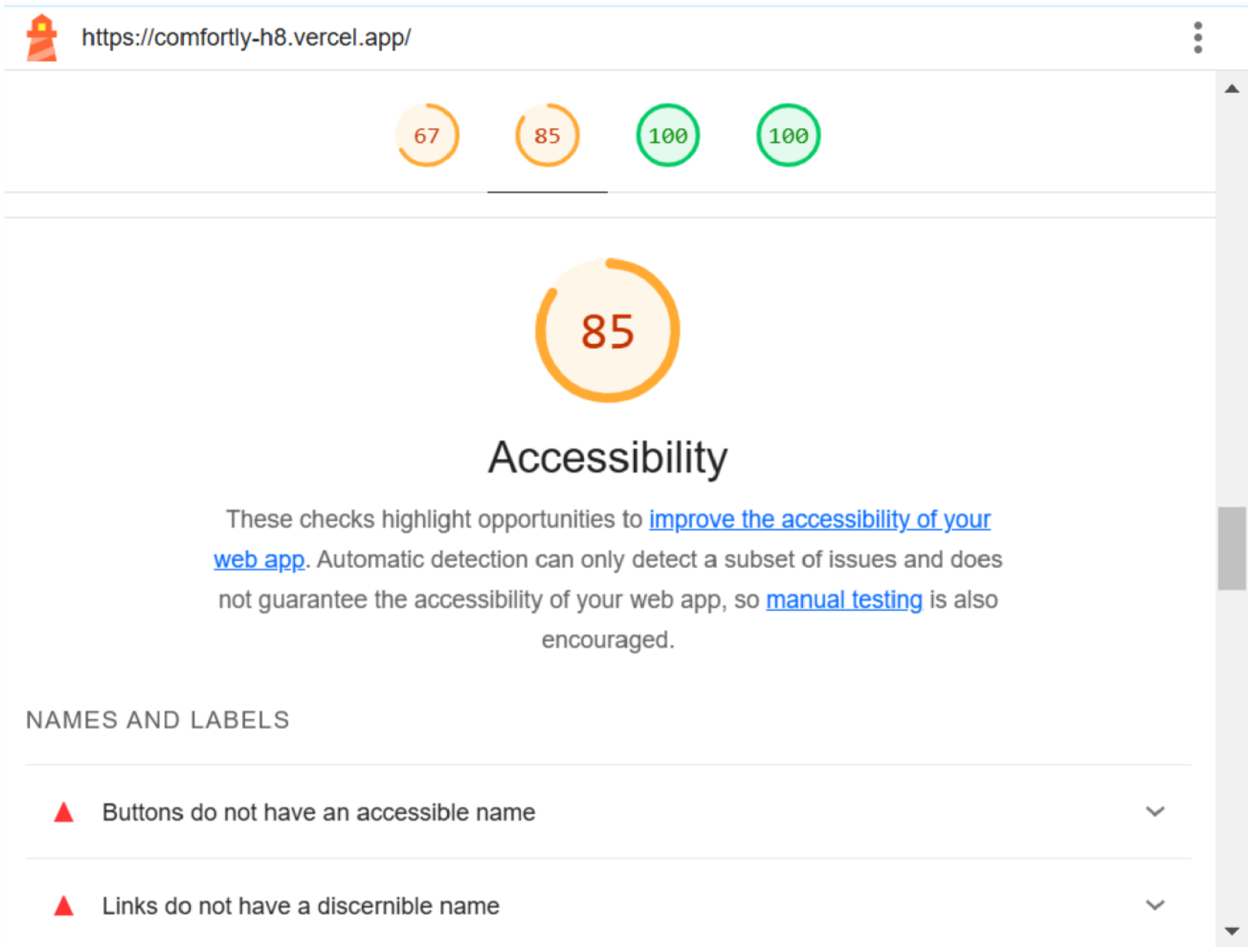
Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

● First Contentful Paint	■ Largest Contentful Paint
1.0 s	2.8 s



https://comfortly-h8.vercel.app/

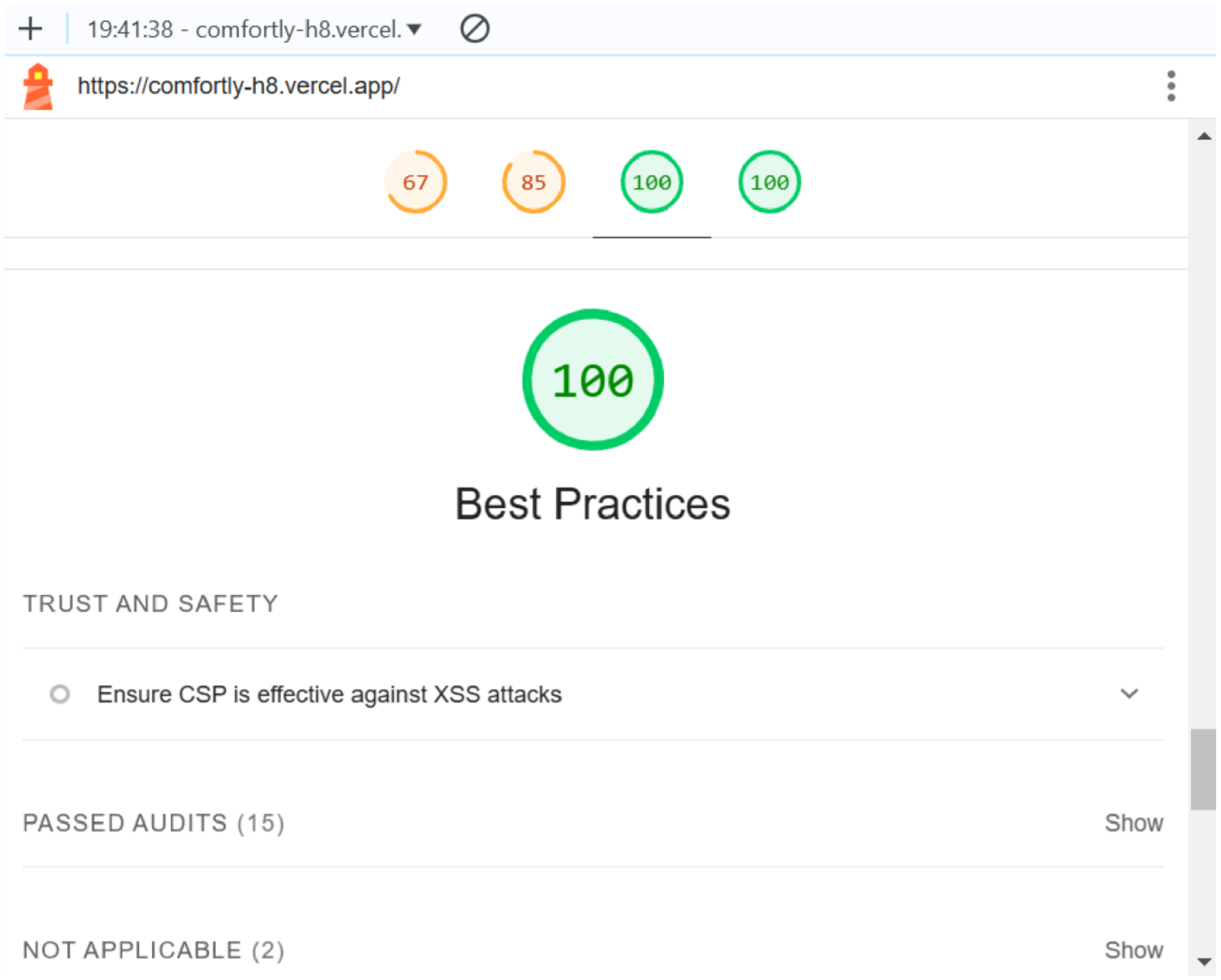
85

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

NAMES AND LABELS

- Buttons do not have an accessible name
- Links do not have a discernible name



19:41:38 - comfortly-h8.vercel.

https://comfortly-h8.vercel.app/

100

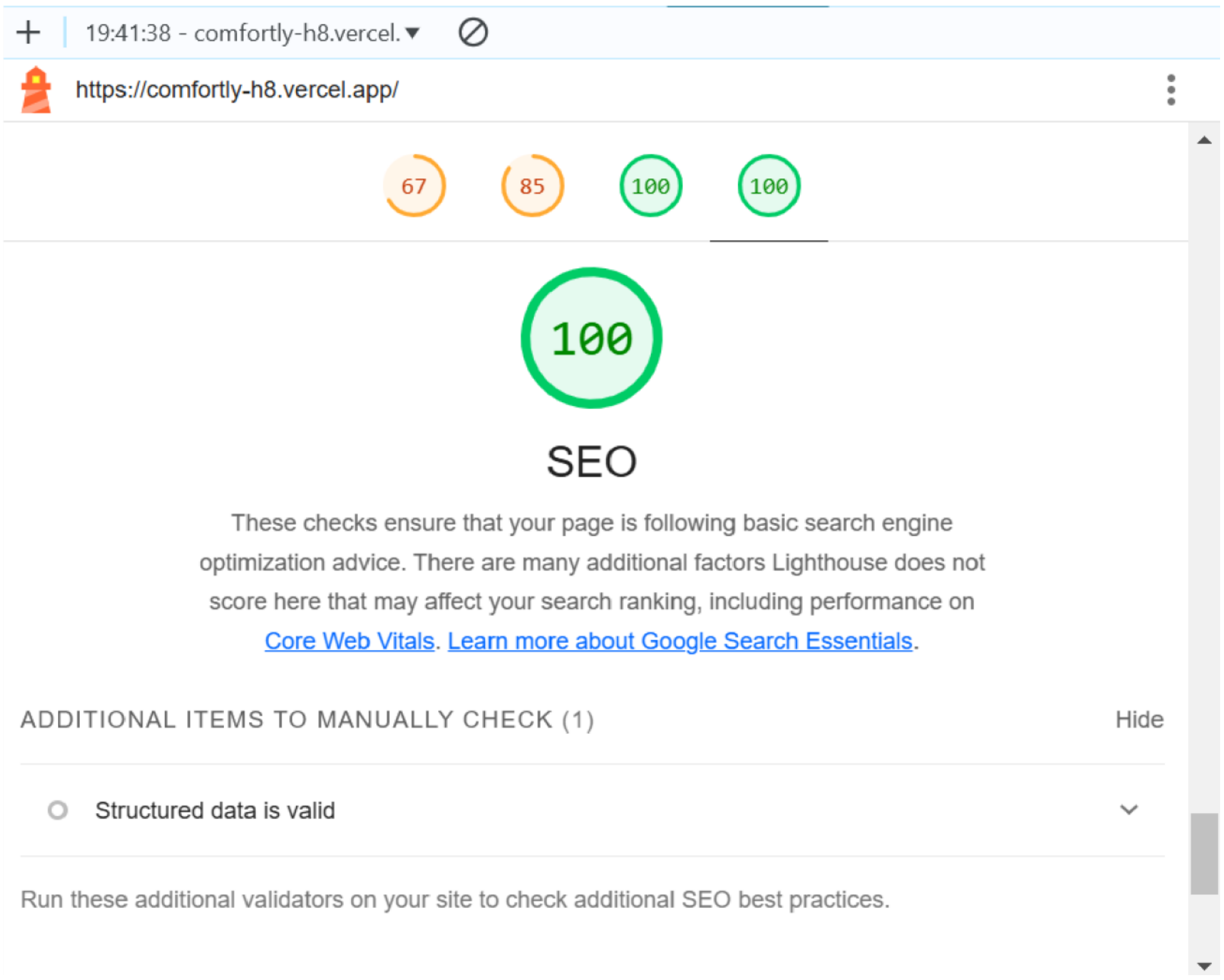
Best Practices

TRUST AND SAFETY

Ensure CSP is effective against XSS attacks

PASSED AUDITS (15) Show

NOT APPLICABLE (2) Show



19:41:38 - comfortly-h8.vercel.▾

https://comfortly-h8.vercel.app/

67 85 100 100

100

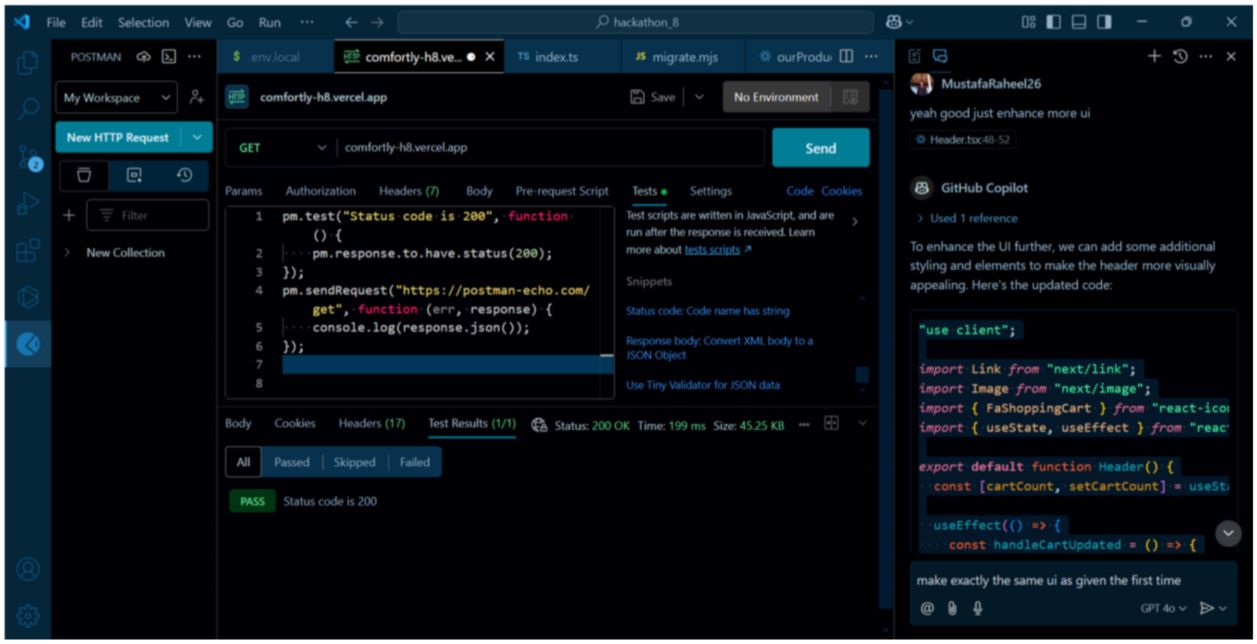
SEO

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) Hide

Structured data is valid

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



comfortly-h8.vercel.app

GET comfortly-h8.vercel.app

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
pm.sendRequest("https://postman-echo.com/get", function (err, response) {
    console.log(response.json());
});
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts ↗

Status code: Code name has string

Response body: Convert XML body to a JSON Object

Use Tiny Validator for JSON data

Status: 200 OK Time: 199 ms Size: 45.25 KB

PASS Status code is 200

MustafaRaheel26

yeah good just enhance more ui

Header.tsx:48-52

GitHub Copilot

Used 1 reference

To enhance the UI further, we can add some additional styling and elements to make the header more visually appealing. Here's the updated code:

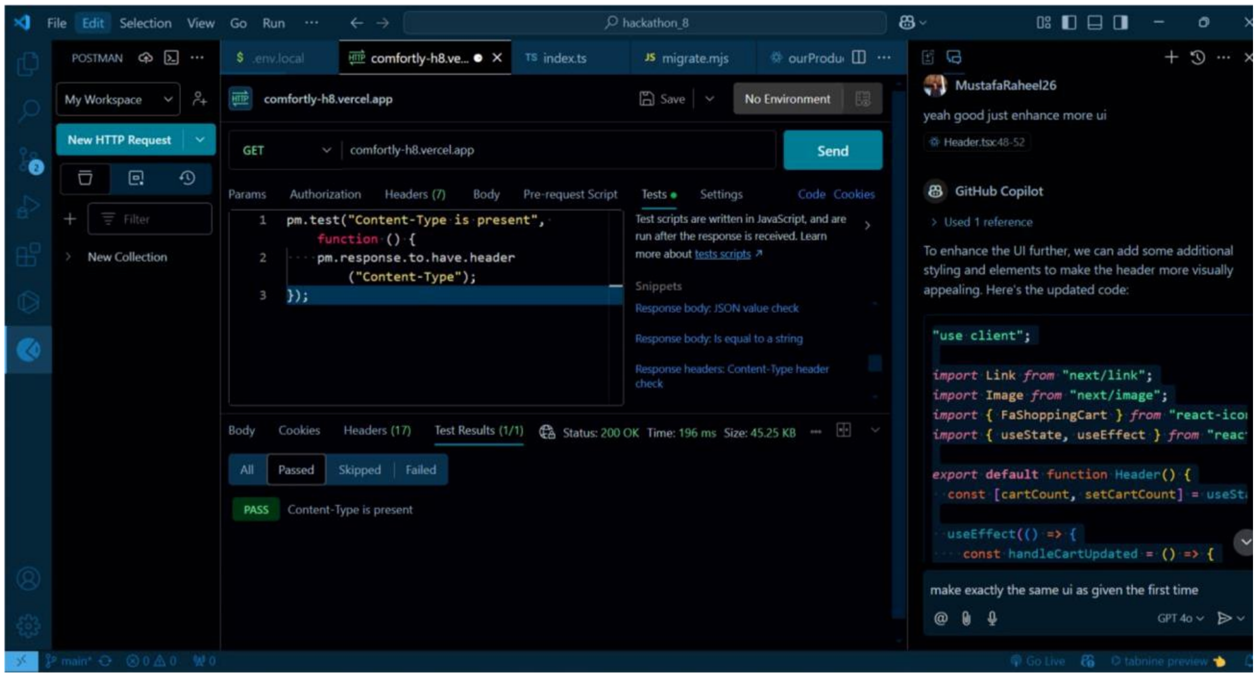
```
"use client";

import Link from "next/link";
import Image from "next/image";
import { FaShoppingCart } from "react-ico
import { useState, useEffect } from "reac

export default function Header() {
  const [cartCount, setCartCount] = useSt

  useEffect(() => {
    const handleCartUpdated = () => {
```

make exactly the same ui as given the first time



comfortly-h8.vercel.app

GET comfortly-h8.vercel.app

```
pm.test("Content-Type is present", function () {
    pm.response.to.have.header("Content-Type");
});
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts ↗

Response body: JSON value check

Response body: Is equal to a string

Response headers: Content-Type header check

Status: 200 OK Time: 196 ms Size: 45.25 KB

PASS Content-Type is present

MustafaRaheel26

yeah good just enhance more ui

Header.tsx:48-52

GitHub Copilot

Used 1 reference

To enhance the UI further, we can add some additional styling and elements to make the header more visually appealing. Here's the updated code:

```
"use client";

import Link from "next/link";
import Image from "next/image";
import { FaShoppingCart } from "react-ico
import { useState, useEffect } from "reac

export default function Header() {
  const [cartCount, setCartCount] = useSt

  useEffect(() => {
    const handleCartUpdated = () => {
```

make exactly the same ui as given the first time

---

2. Performance Optimization Steps Taken:

1. **Optimized API Requests**:
    a. Implemented lazy loading for product images and data to reduce initial load time.
    b. Consolidated multiple API calls into batch requests to minimize overhead.
2. **Caching Mechanisms**:
    a. Introduced client-side caching for frequently accessed data such as product details.
    b. Leveraged browser storage (localStorage) for storing user preferences and session data.
3. **Code Optimization**:
    a. Minimized JavaScript bundle size by removing unused dependencies and applying tree-shaking.
    b. Reduced CSS file size by adopting modular styles and purging unused classes.
4. **Load Testing**:
    a. Conducted load testing to ensure the application performs well under concurrent user traffic.

---

3. Security Measures Implemented:

1. **Authentication and Authorization**:
    - Implemented JWT-based authentication to secure user sessions.
    - Restricted access to sensitive API endpoints based on user roles.
2. **Input Validation**:
    - Sanitized user inputs to prevent SQL injection and XSS attacks.
    - Utilized server-side validation for critical forms (e.g., login, registration).
3. **Secure Data Handling**:
    - Enforced HTTPS across all pages for secure communication.
    - Stored sensitive information, such as passwords, in hashed format using bcrypt.

4. **Vulnerability Scanning**:
    - Conducted regular scans using tools like OWASP ZAP to identify and mitigate vulnerabilities.

4. Challenges Faced and Resolutions Applied:

1. **Challenge**: Slow page loading due to large product images.
    - **Resolution**: Introduced image compression and lazy loading for non-critical assets.
2. **Challenge**: API downtime affecting functionality.
    - **Resolution**: Added fallback UI with meaningful error messages and retry logic for API calls.
3. **Challenge**: Ensuring cross-browser compatibility.

   - **Resolution**: Tested on multiple browsers and applied polyfills for unsupported features.
4. **Challenge**: Maintaining responsive design for mobile users.
   - **Resolution**: Utilized a mobile-first CSS framework and thoroughly tested on various screen sizes.