Day 5 - Testing and Backend Refinement

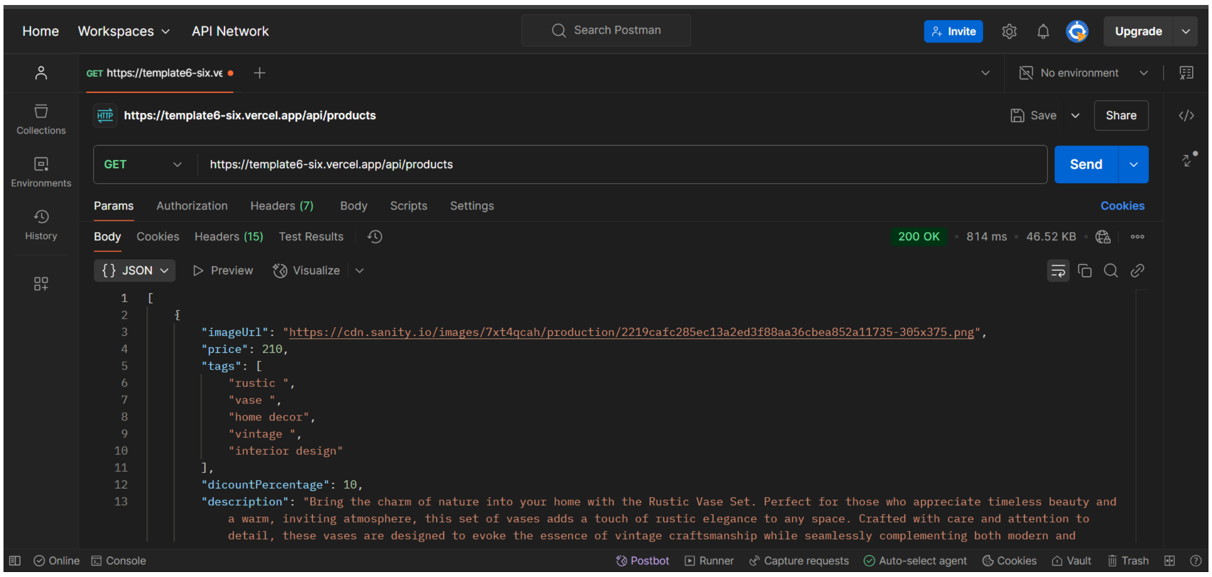
Introduction:

In the evolving landscape of eCommerce, ensuring a seamless user experience, high performance, and secure interactions is crucial. On Day 5 of the Marketplace Builder Hackathon, I focused on refining the backend, implementing rigorous testing protocols, and optimizing performance for my online marketplace. This document highlights the key steps I undertook, including functional testing, error handling strategies, performance enhancements, and cross-browser/device testing.

Step 1: Functional Testing

1. **Core Features Testing:**
- **Product Listing:** Verified correct display of product names, images, prices, and availability.
- **Filters and Search:** Tested accurate filtering and search results based on user inputs.
- **Cart Operations:** Validated the cart functionality by ensuring users could add, remove, and update items.

2. **Testing Tools Used:**
- **Postman:** For API response validation.



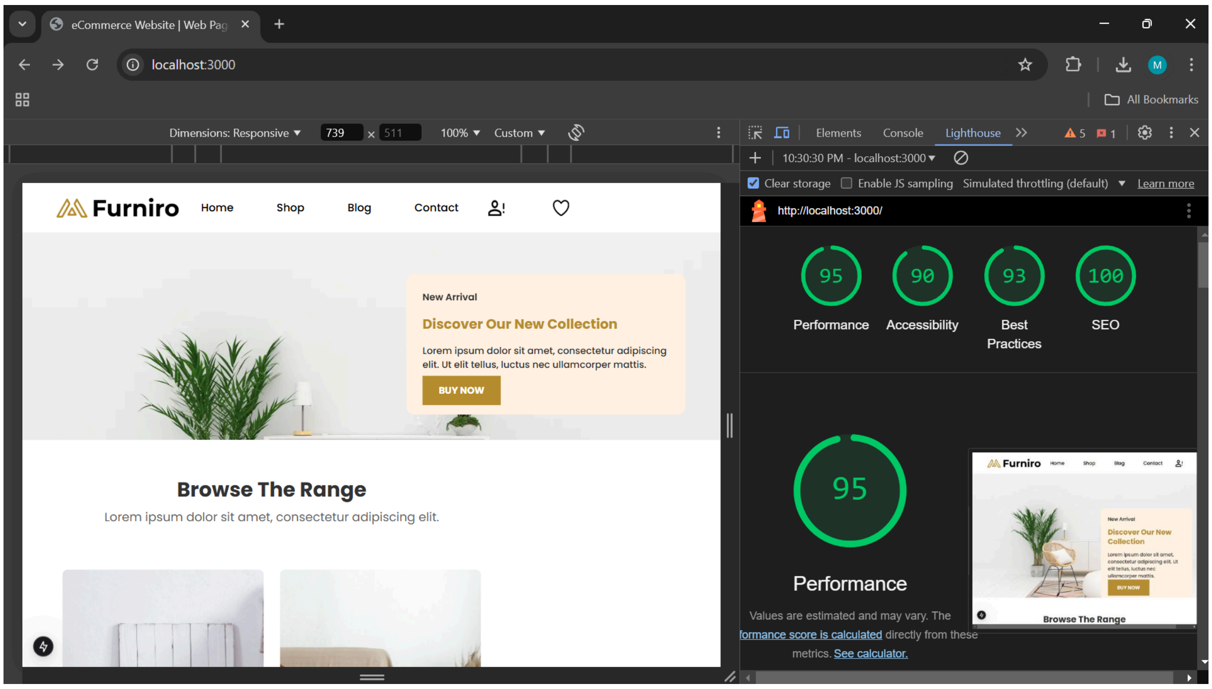
GET https://template6-six.vercel.app/api/products

200 OK • 814 ms • 46.52 KB

```json
[
    {
        "imageUrl": "https://cdn.sanity.io/images/7xt4qcah/production/2219cafc285ec13a2ed3f88aa36cbea852a11735-305x375.png",
        "price": 210,
        "tags": [
            "rustic ",
            "vase ",
            "home decor",
            "vintage ",
            "interior design"
        ],
        "dicountPercentage": 10,
        "description": "Bring the charm of nature into your home with the Rustic Vase Set. Perfect for those who appreciate timeless beauty and
            a warm, inviting atmosphere, this set of vases adds a touch of rustic elegance to any space. Crafted with care and attention to
            detail, these vases are designed to evoke the essence of vintage craftsmanship while seamlessly complementing both modern and
```

- **Lighthouse:** For performance auditing, identifying areas for improvement.

Prepared By: Mariyam Asif


eCommerce Website | Web Pag
localhost:3000
All Bookmarks
Dimensions: Responsive 739 × 511 100% Custom
Elements Console Lighthouse
10:30:30 PM - localhost:3000
Clear storage Enable JS sampling Simulated throttling (default) Learn more
http://localhost:3000/
Furniro Home Shop Blog Contact
New Arrival
Discover Our New Collection
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut elit tellus, luctus nec ullamcorper mattis.
BUY NOW
Browse The Range
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
95 Performance
90 Accessibility
93 Best Practices
100 SEO
95 Performance
Values are estimated and may vary. The formance score is calculated directly from these metrics. See calculator.

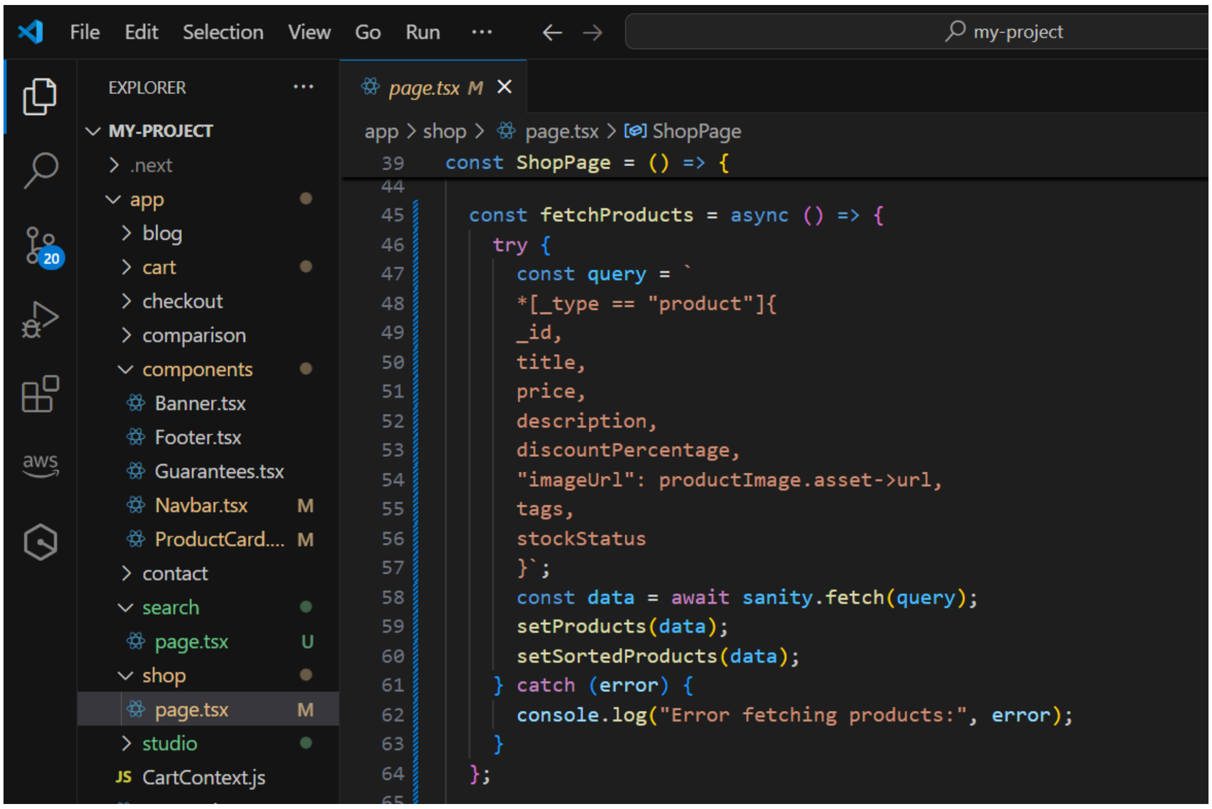
3. **Testing Execution:**

- Developed test cases for each feature.
- Simulated user actions like browsing products, navigation and adding/removing items from cart.
- Ensured the product list displayed correctly after filtering by name.

Step 2: Error Handling

1. **Error Messages:**
   - Integrated **try-catch** blocks to handle API errors effectively:

Prepared By: Mariyam Asif

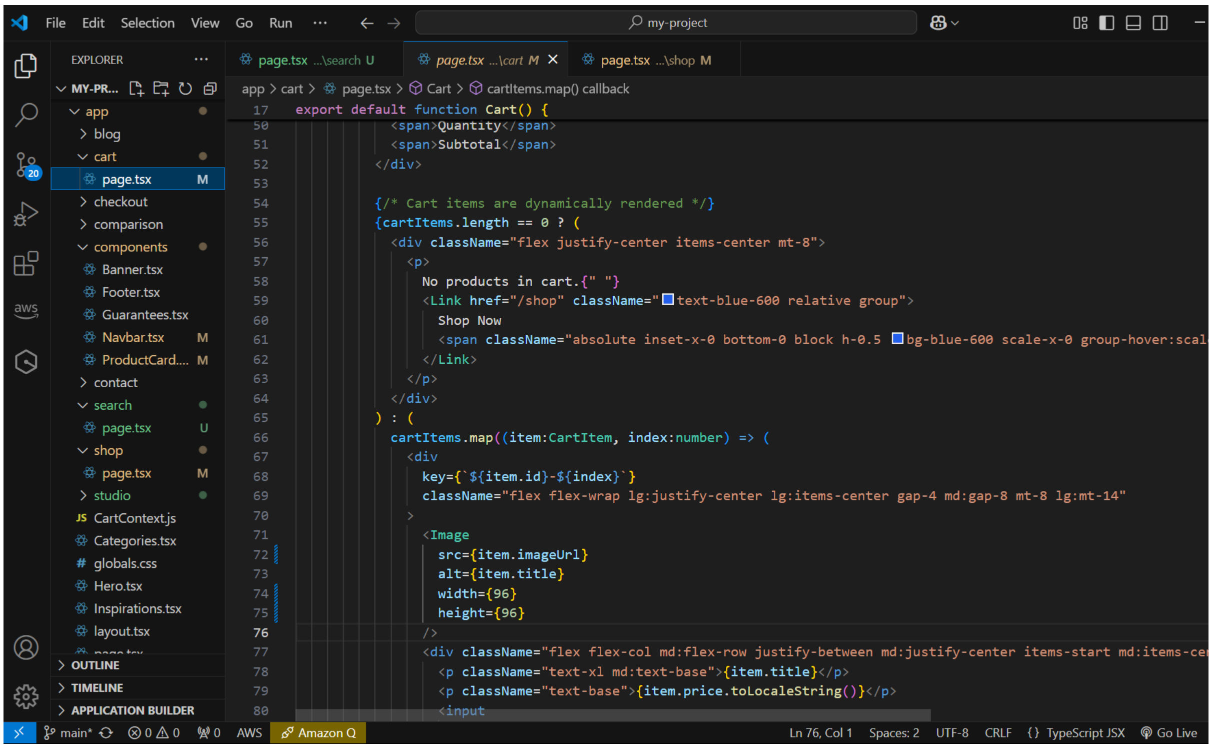


```
const ShopPage = () => {

  const fetchProducts = async () => {
    try {
      const query = `
      *[_type == "product"]{
      _id,
      title,
      price,
      description,
      discountPercentage,
      "imageUrl": productImage.asset->url,
      tags,
      stockStatus
      }`;
      const data = await sanity.fetch(query);
      setProducts(data);
      setSortedProducts(data);
    } catch (error) {
      console.log("Error fetching products:", error);
    }
  };
```

2. **Fallback UI:**

- Implemented fallback UI for scenarios where data is unavailable, such as a "No products found" message in the search page or an empty cart fallback text.



```
export default function Cart() {
          <span>Quantity</span>
          <span>Subtotal</span>
        </div>

        {/* Cart items are dynamically rendered */}
        {cartItems.length == 0 ? (
          <div className="flex justify-center items-center mt-8">
            <p>
              No products in cart.{" "}
              <Link href="/shop" className="text-blue-600 relative group">
                Shop Now
                <span className="absolute inset-x-0 bottom-0 block h-0.5 bg-blue-600 scale-x-0 group-hover:scal
              </Link>
            </p>
          </div>
        ) : (
          cartItems.map((item:CartItem, index:number) => (
            <div
              key={`${item.id}-${index}`}
              className="flex flex-wrap lg:justify-center lg:items-center gap-4 md:gap-8 mt-8 lg:mt-14"
            >
              <Image
                src={item.imageUrl}
                alt={item.title}
                width={96}
                height={96}
              />
              <div className="flex flex-col md:flex-row justify-between md:justify-center items-start md:items-cer
                <p className="text-xl md:text-base">{item.title}</p>
                <p className="text-base">{item.price.toLocaleString()}</p>
                <input
```

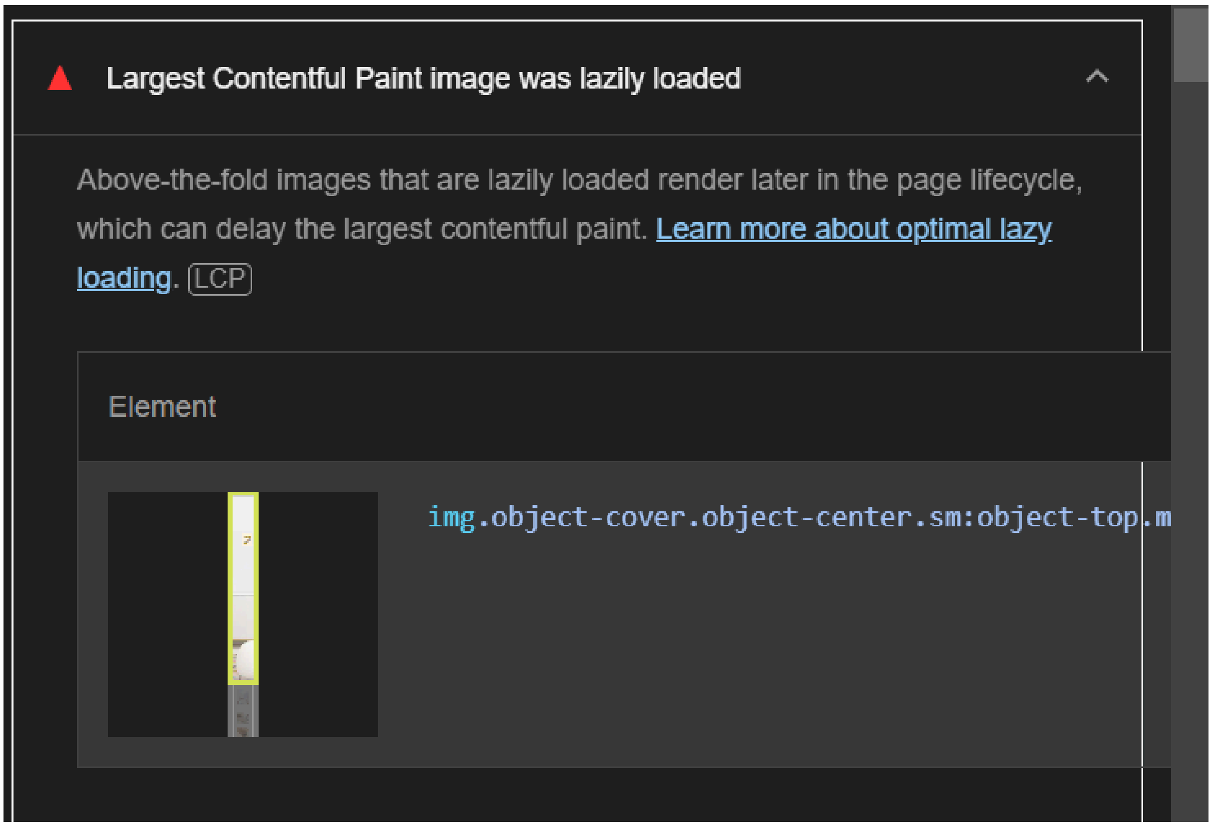
Prepared By: Mariyam Asif

Step 3: Performance Optimization

1. **Optimizing Assets:**
   - **Image Optimization:** Compressed images and implemented lazy loading for all images to improve page load times.
   - **Performance Auditing:** Ran performance audits using Lighthouse to address issues like unused CSS and JavaScript bundles.

2. Page Speed Improvements:

- **Problem # 1:** Lazy loading of above-the-fold images delayed the Largest Contentful Paint (LCP), which negatively impacted page load speed.



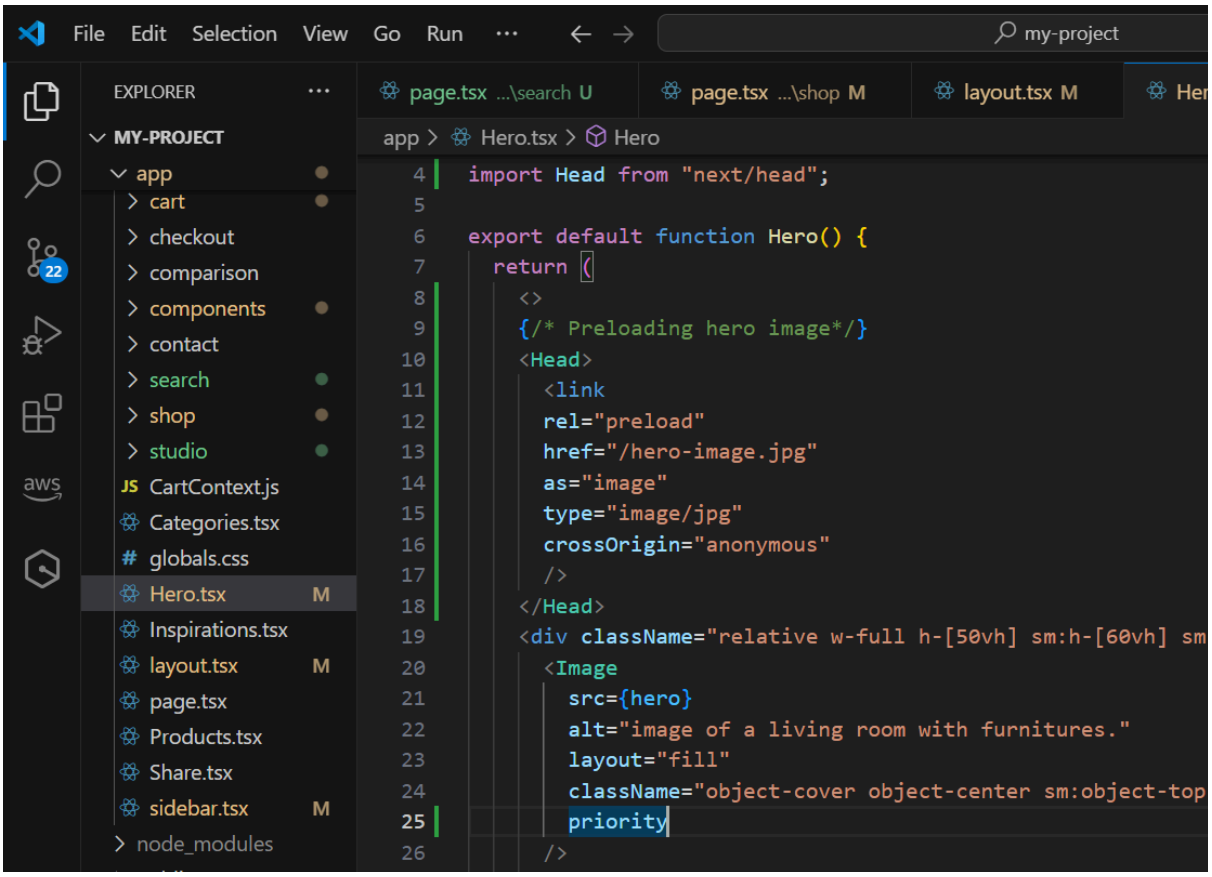
Largest Contentful Paint image was lazily loaded

Above-the-fold images that are lazily loaded render later in the page lifecycle, which can delay the largest contentful paint. Learn more about optimal lazy loading. LCP

Element

`img.object-cover.object-center.sm:object-top.m`

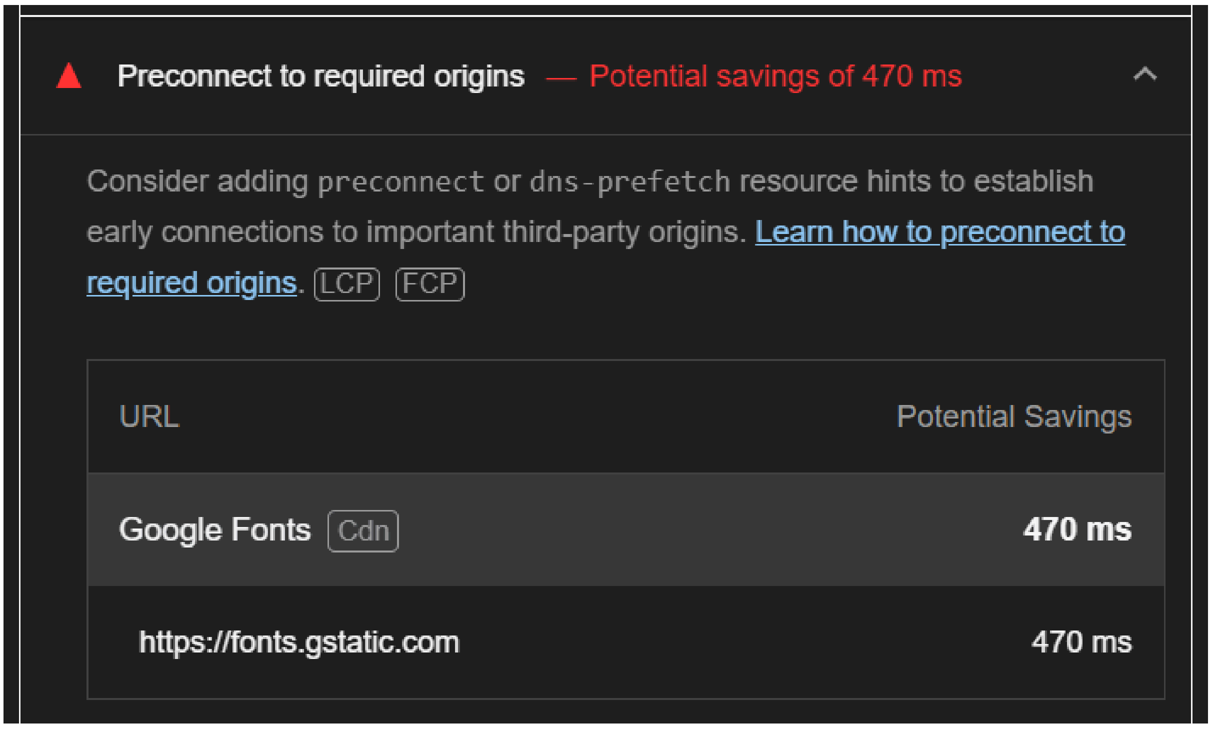
   - **Solution:**
     - Preloaded critical above-the-fold images using the `<link rel="preload">` tag.
     - This allowed the images to load earlier, reducing the LCP time and improving page speed.



```
import Head from "next/head";

export default function Hero() {
  return (
    <>
    {/* Preloading hero image*/}
    <Head>
      <link
      rel="preload"
      href="/hero-image.jpg"
      as="image"
      type="image/jpg"
      crossOrigin="anonymous"
      />
    </Head>
    <div className="relative w-full h-[50vh] sm:h-[60vh] sm
      <Image
        src={hero}
        alt="image of a living room with furnitures."
        layout="fill"
        className="object-cover object-center sm:object-top
        priority
      />
```

- **Problem # 2:** Pre Connecting to external origins was not optimized, resulting in delayed resource loading for external resources like Google Fonts.



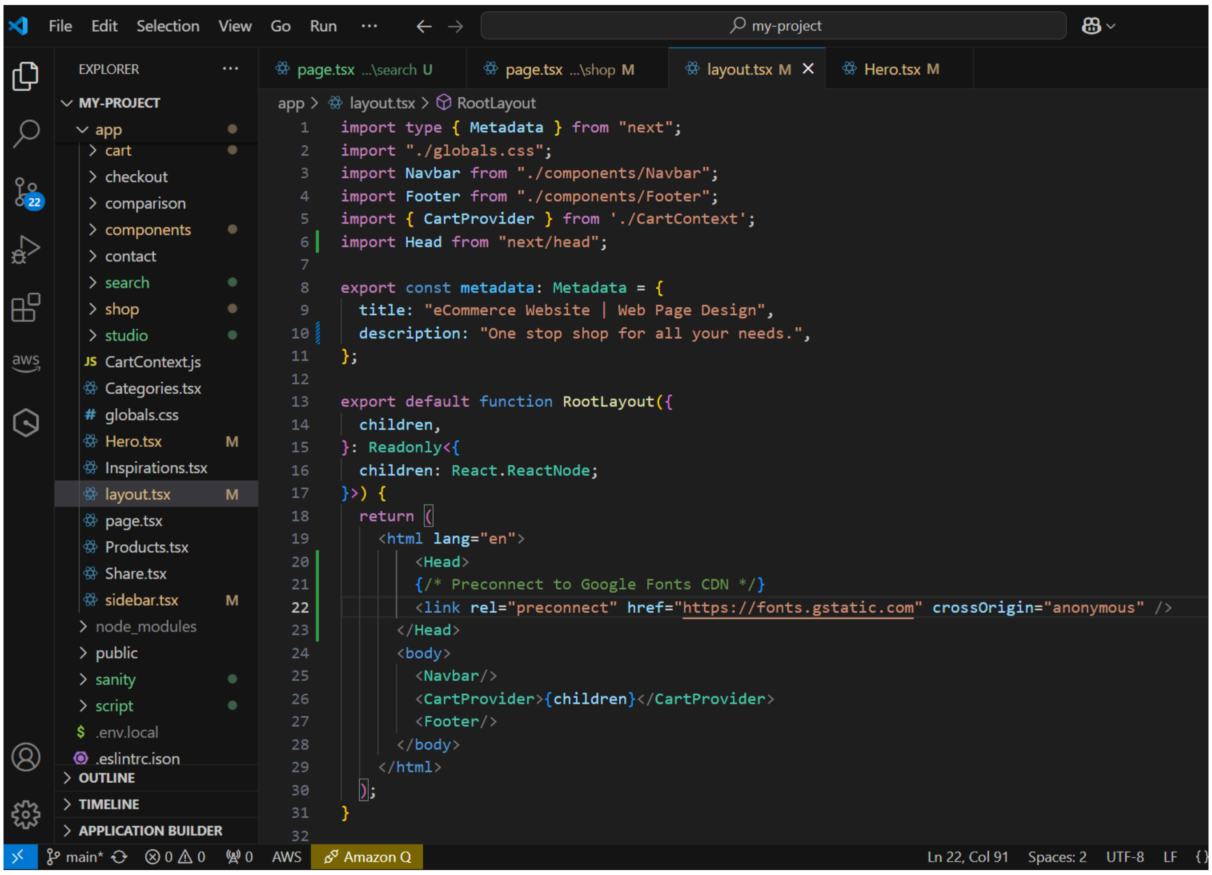
▲ Preconnect to required origins — Potential savings of 470 ms

Consider adding `preconnect` or `dns-prefetch` resource hints to establish early connections to important third-party origins. Learn how to preconnect to required origins. LCP FCP

URL	Potential Savings
Google Fonts Cdn	470 ms
https://fonts.gstatic.com	470 ms

  - **Solution:**

- Implemented the `<link rel="preconnect">` tag in the `<head>` section of the HTML.
- This established early connections to the Google Fonts domain, allowing the browser to handle DNS resolution, SSL negotiation, and TCP handshakes ahead of time, speeding up the loading of external resources.



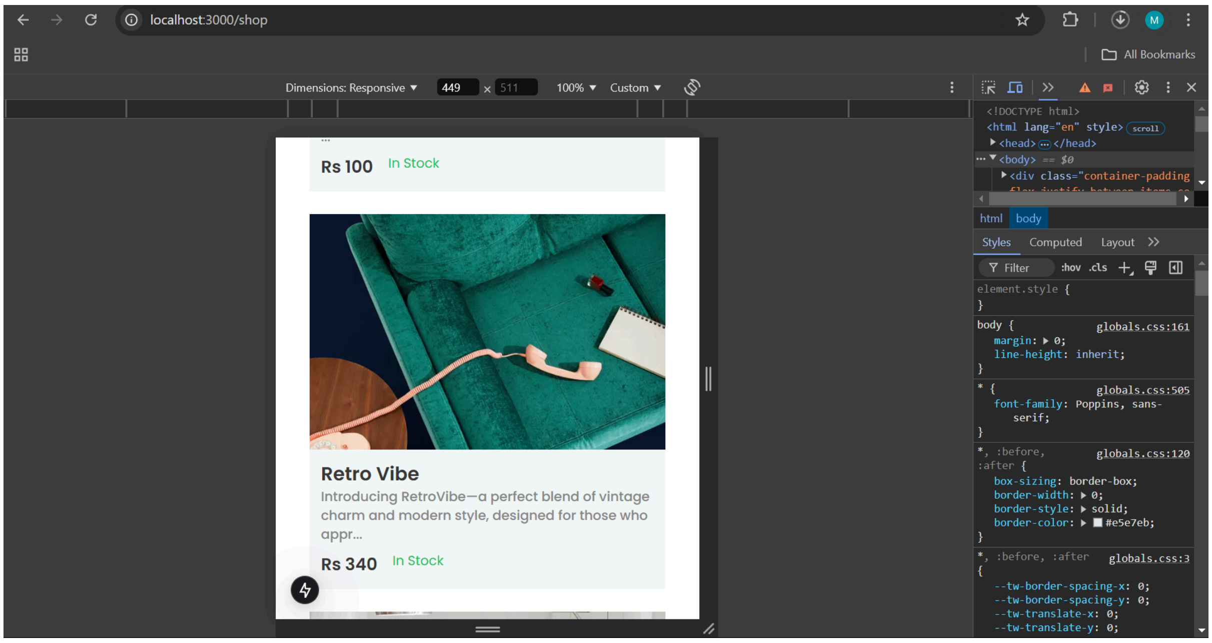
```tsx
import type { Metadata } from "next";
import "./globals.css";
import Navbar from "./components/Navbar";
import Footer from "./components/Footer";
import { CartProvider } from './CartContext';
import Head from "next/head";

export const metadata: Metadata = {
  title: "eCommerce Website | Web Page Design",
  description: "One stop shop for all your needs.",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
        <Head>
        {/* Preconnect to Google Fonts CDN */}
        <link rel="preconnect" href="https://fonts.gstatic.com" crossOrigin="anonymous" />
      </Head>
      <body>
        <Navbar/>
        <CartProvider>{children}</CartProvider>
        <Footer/>
      </body>
    </html>
  );
}
```

Step 4: Cross-Browser and Device Testing

1. **Browser Testing:**
   - Conducted tests on **Chrome, Firefox, Safari, and Edge** to ensure consistent rendering and functionality.
   - Verified all functionalities worked as expected across different browsers.
2. **Device Testing:**
   - Tested on **physical mobile devices** to ensure a responsive, user-friendly experience across platforms.



Rs 100 In Stock

Retro Vibe
Introducing RetroVibe—a perfect blend of vintage charm and modern style, designed for those who appr...
Rs 340 In Stock

```
<!DOCTYPE html>
<html lang="en" style> scroll
▸<head> … </head>
… ▾<body> == $0
  ▸<div class="container-padding
```

```
element.style {
}
body {                    globals.css:161
  margin: ▸ 0;
  line-height: inherit;
}
* {                       globals.css:505
  font-family: Poppins, sans-
    serif;
}
*, :before,               globals.css:120
:after {
  box-sizing: border-box;
  border-width: ▸ 0;
  border-style: ▸ solid;
  border-color: ▸ #e5e7eb;
}
*, :before, :after        globals.css:3
{
  --tw-border-spacing-x: 0;
  --tw-border-spacing-y: 0;
  --tw-translate-x: 0;
  --tw-translate-y: 0;
```

Step 5: Security Testing

- **Secure API Communication:**
  - Secured API requests by enforcing HTTPS and storing sensitive data, like API keys, in environment variables.

Step 6: User Acceptance Testing (UAT)

1. **Simulating Real-World Usage:**
   - Conducted UAT by simulating tasks like browsing products, adding to the cart, and reviewing the cart items.
   - Identified usability issues and improved the interface for smoother interactions.

Conclusion:

Through rigorous testing, backend refinement, and performance optimization, I have ensured that the online marketplace is not only fully functional but also secure, fast, and user-friendly.

Prepared By: Mariyam Asif