Day 5 - Testing and Backend Refinement - [SHOP.CO]

1. Introduction

This report outlines the testing activities conducted for the E-Commerce Marketplace project. The primary objectives of the testing were to ensure functionality, performance, error handling, and cross-browser compatibility, preparing the application for deployment.

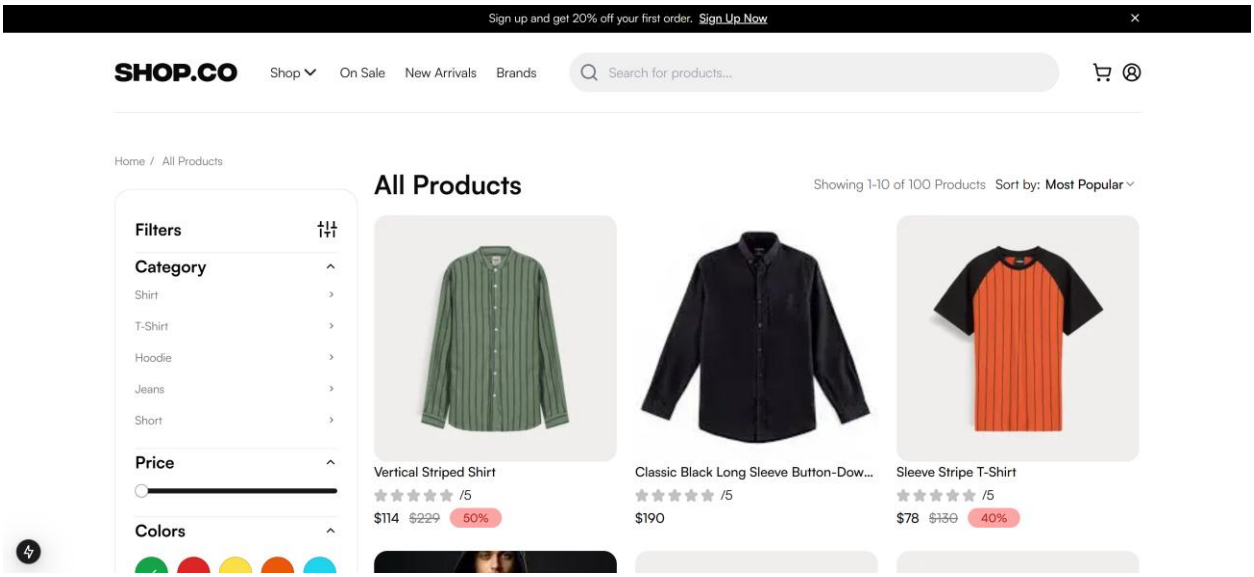
The testing was conducted using a combination of manual and automated tools such as **Postman**, **Lighthouse**, and **BrowserStack** etc. The findings and recommendations are detailed below.

2. Testing Overview

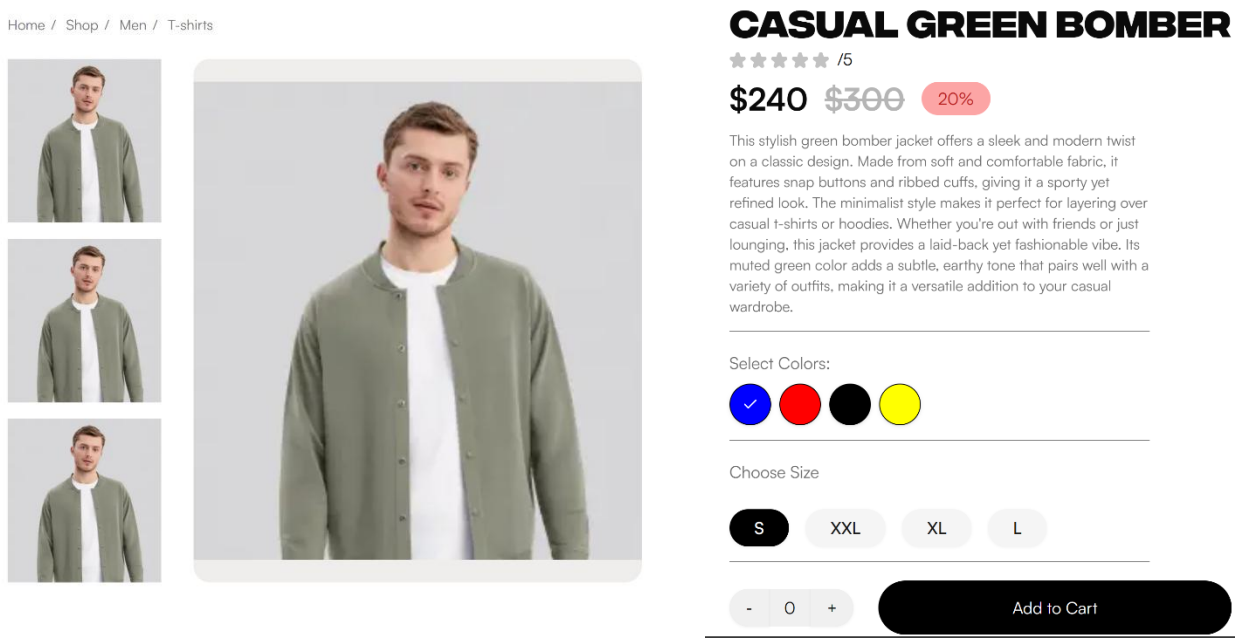
Types of Testing Performed

- **Functional Testing**: Validating core features such as product listing, cart functionality, and product detailed pages.

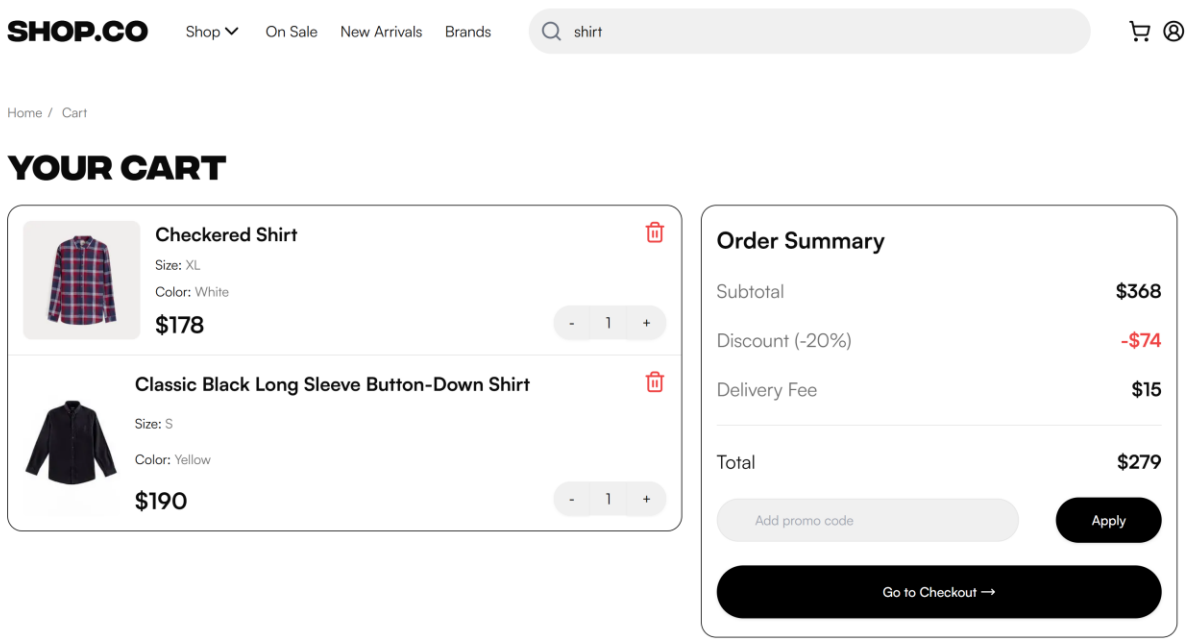
Products Listing Page**:**



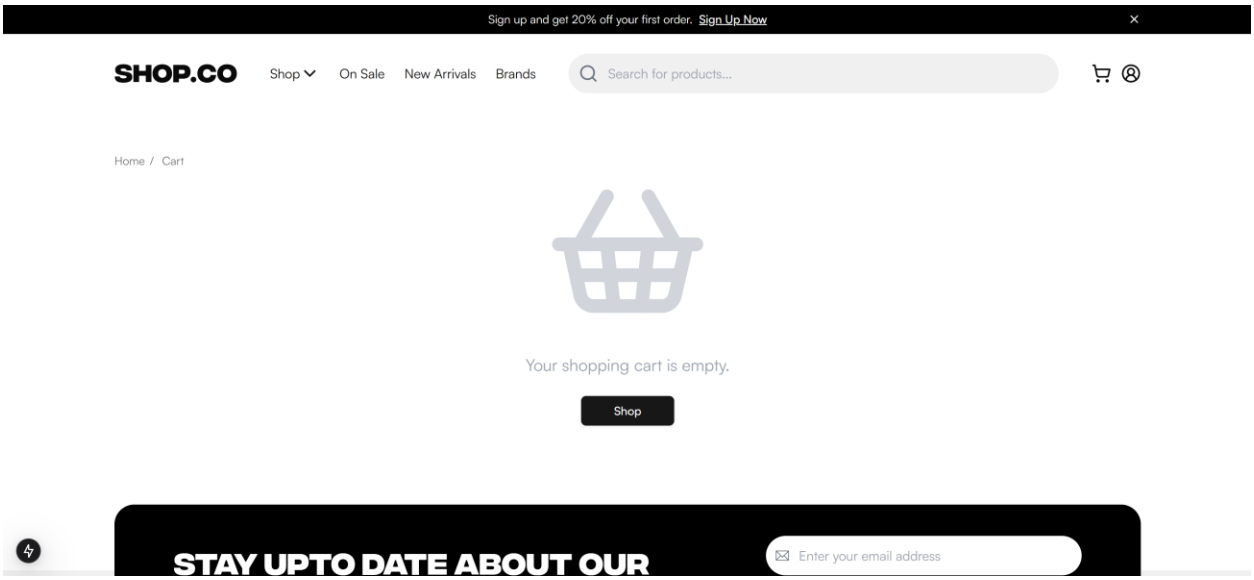
Product Detailed Page:



Cart Functionality:

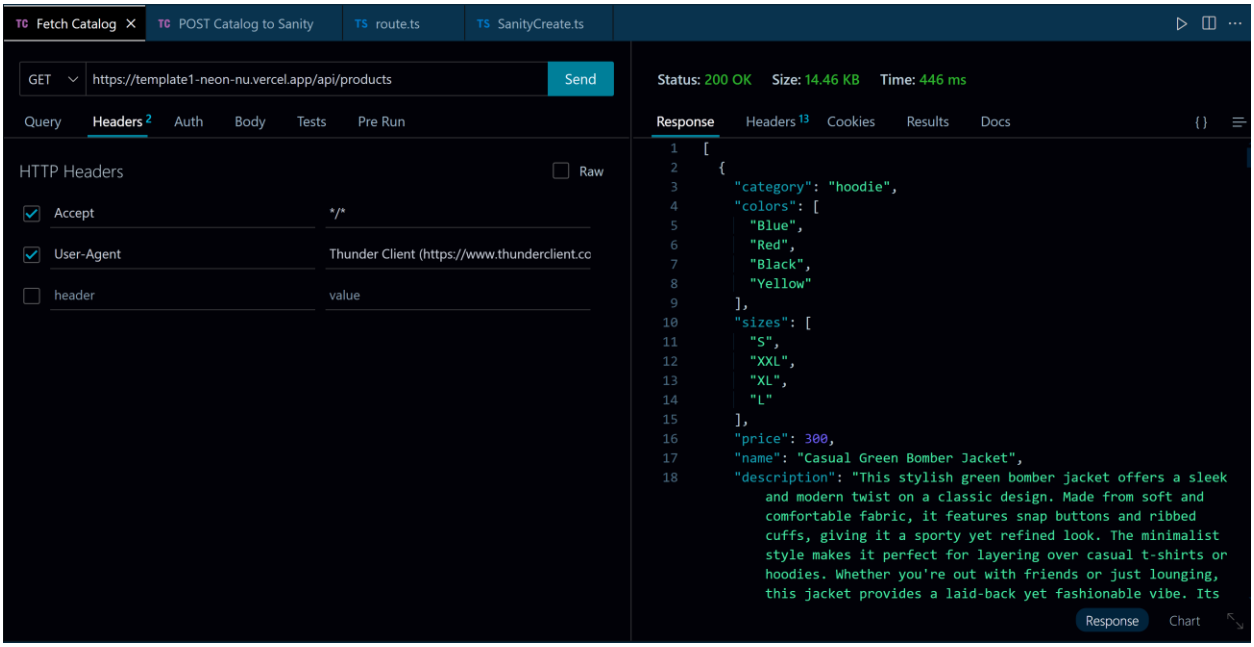


FallBack UI when Cart is Empty:



- **API Testing**: Ensuring backend endpoints respond correctly and error handling works as expected**.**

Data Fetching using GET:



Fetched Data to /api/catalog to to upload on sanity.

Fetch Catalog | POST Catalog to Sanity | route.ts | SanityCreate.ts

POST http://localhost:3000/api/catalog Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format


```
[
  {
    "category": "hoodie",
    "colors": [
      "Blue",
      "Red",
      "Black",
      "Yellow"
    ],
    "sizes": [
      "S",
      "XXL",
      "XL",
      "L"
    ],
    "price": 300,
    "name": "Casual Green Bomber Jacket",
    "description": "This stylish green bomber jacket offers a
      sleek and modern twist on a classic design. Made from soft
      and comfortable fabric, it features snap buttons and
```


Status: 200 OK Size: 943 Bytes Time: 327 ms

Response Headers 5 Cookies Results Docs { }


```
{
  "message": "Catalog created successfully",
  "data": [
    {
      "category": "hoodie",
      "colors": [
        "Blue",
        "Red",
        "Black",
        "Yellow"
      ],
      "sizes": [
        "S",
        "XXL",
        "XL",
        "L"
      ],
      "price": 300,
      "name": "Casual Green Bomber Jacket",
      "description": "This stylish green bomber jacket offers a sleek
        and modern twist on a classic design. Made from soft and
        comfortable fabric, it features snap buttons and ribbed
        cuffs, giving it a sporty yet refined look. The minimalist
        style makes it perfect for layering over casual t-shirts or
```


Response Chart

API Error Handling:

POST http://localhost:3000/api/catalog Send

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

Status: 500 Internal Server Error Size: 78 Bytes Time: 226 ms

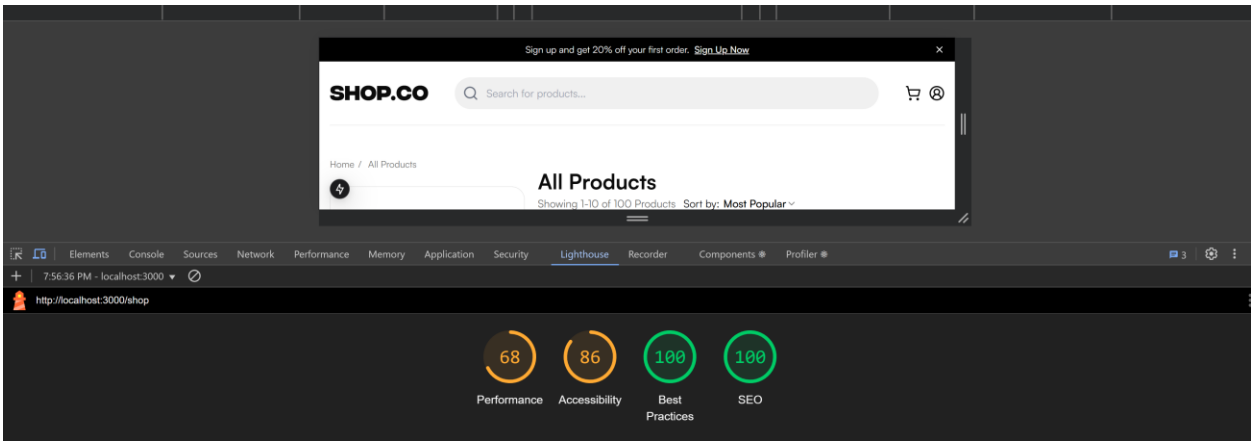
Response Headers 5 Cookies Results Docs { }


```
{
  "error": "Failed to create catalog SyntaxError:
      Unexpected end of JSON input"
}
```

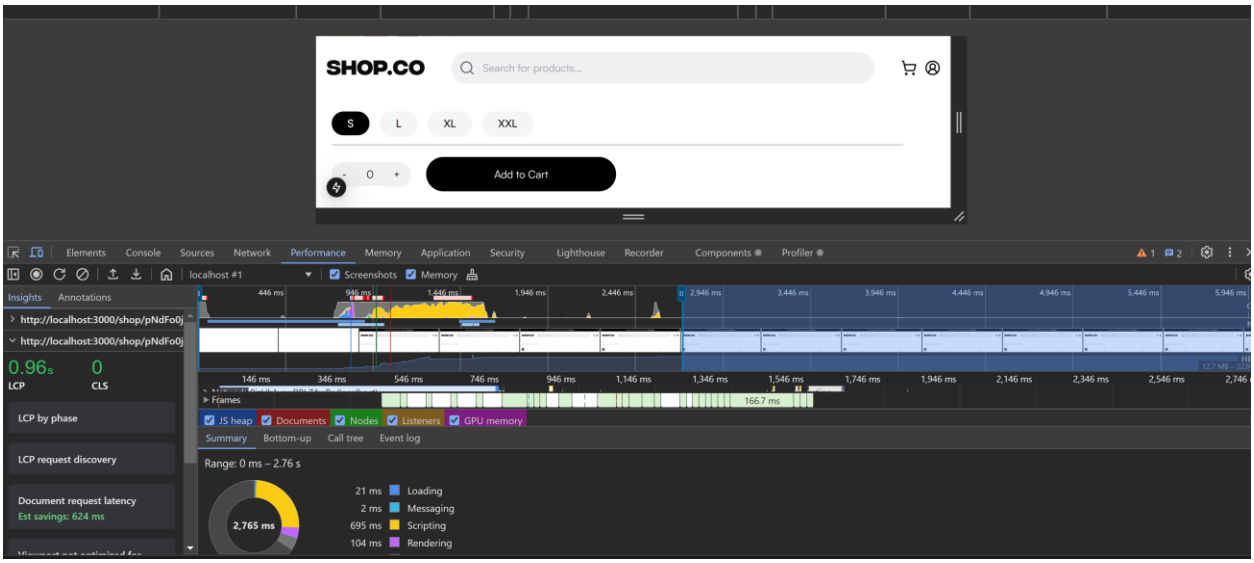

Response Chart

- **Performance Testing**: Measuring page load times and optimizing asset delivery.

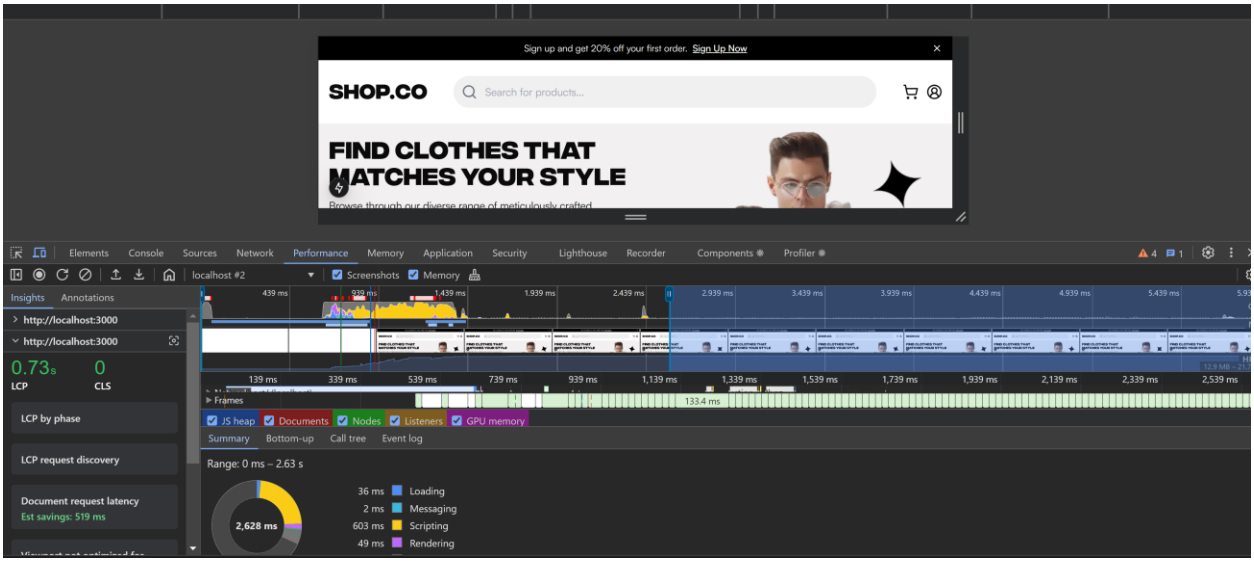
Using Browser Dev Tool:



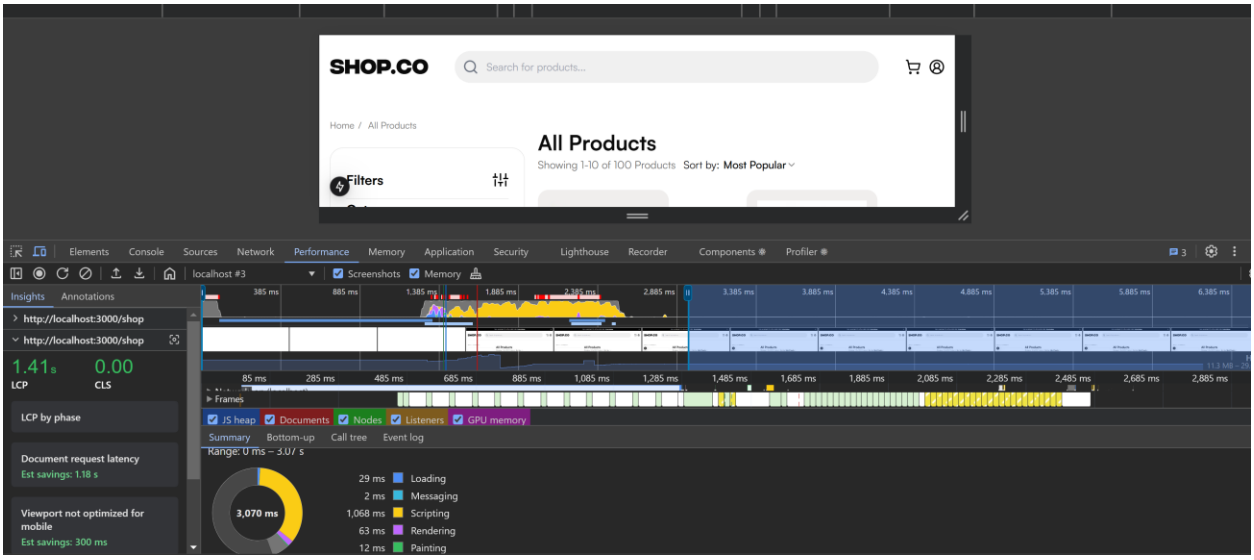
Add To Cart:



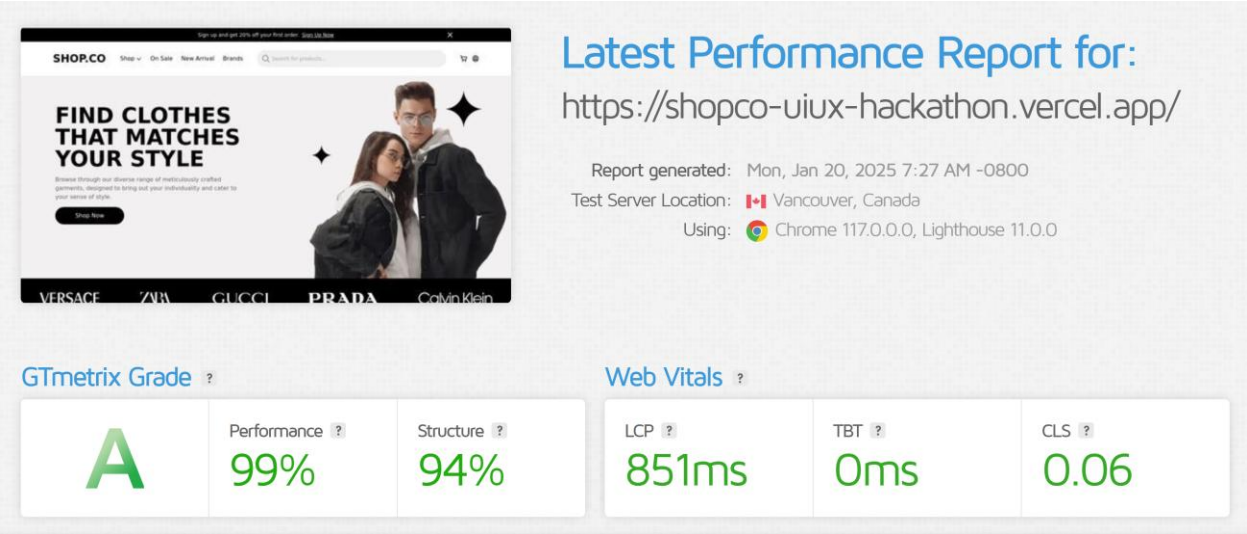
Home Page:



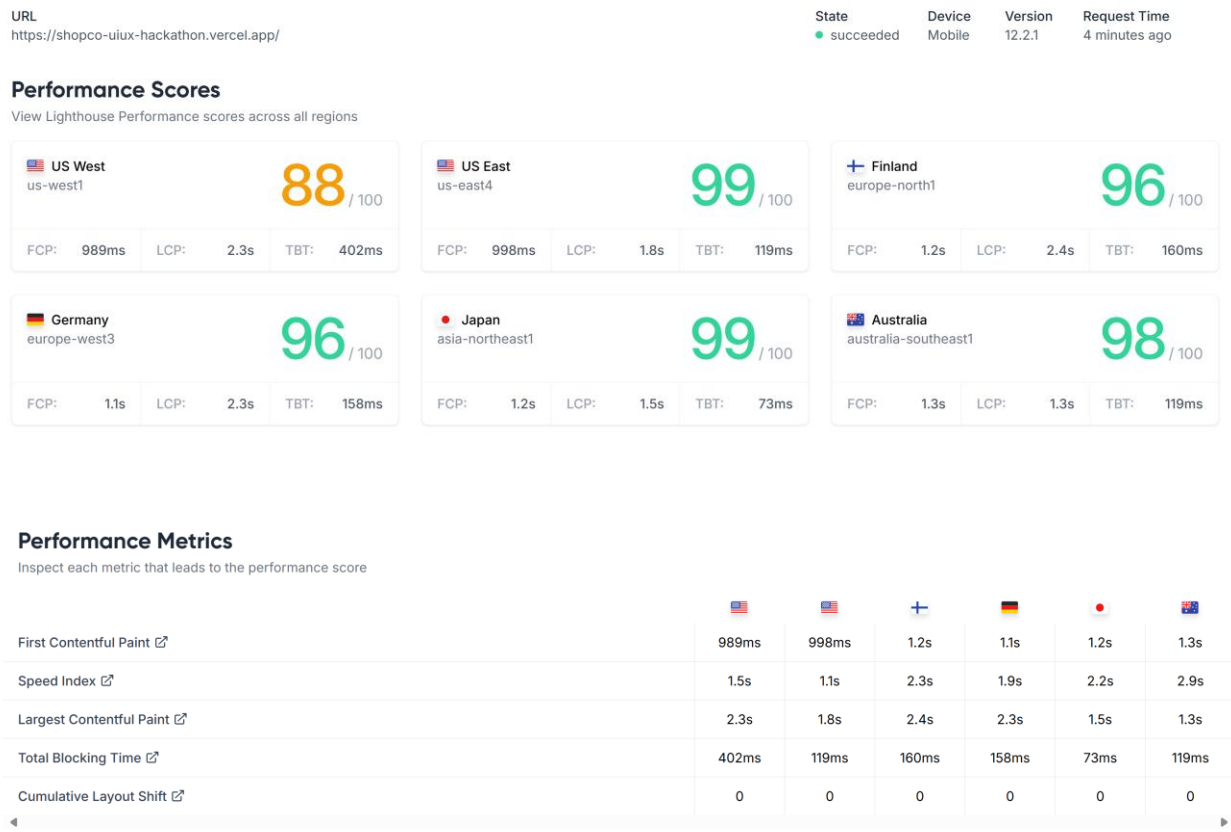
Product Listing Page:



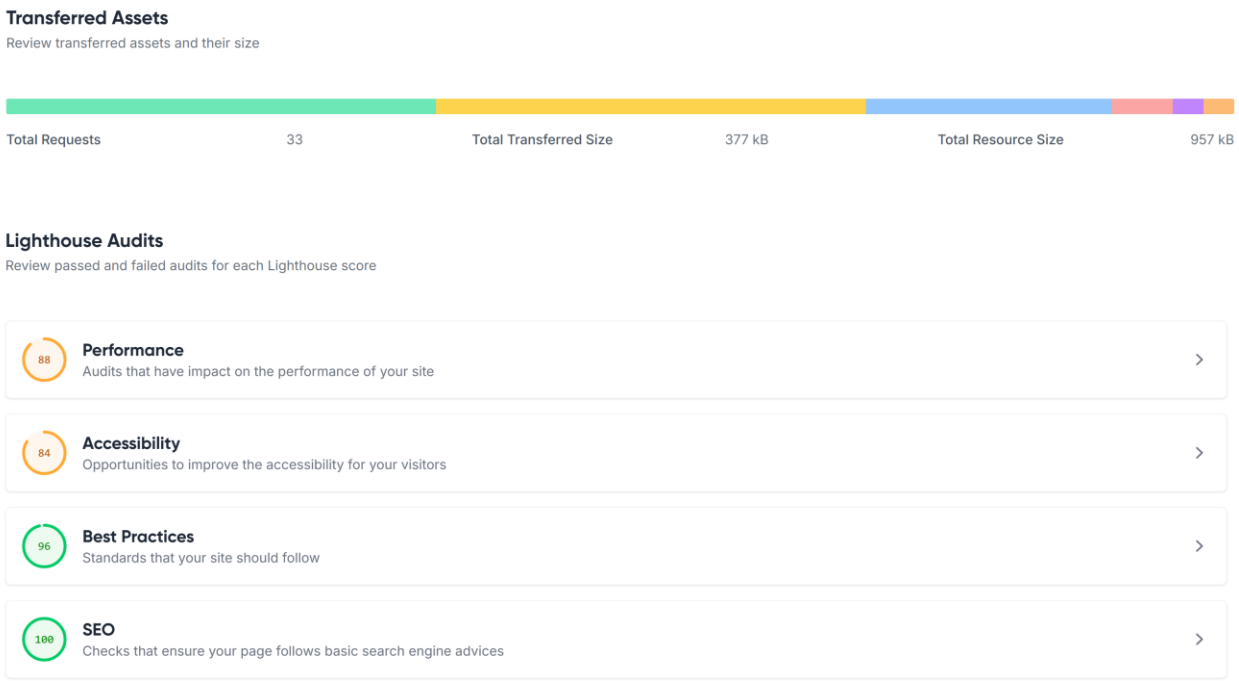
Using GTMetrix:



Using Lighthouse Metrix:

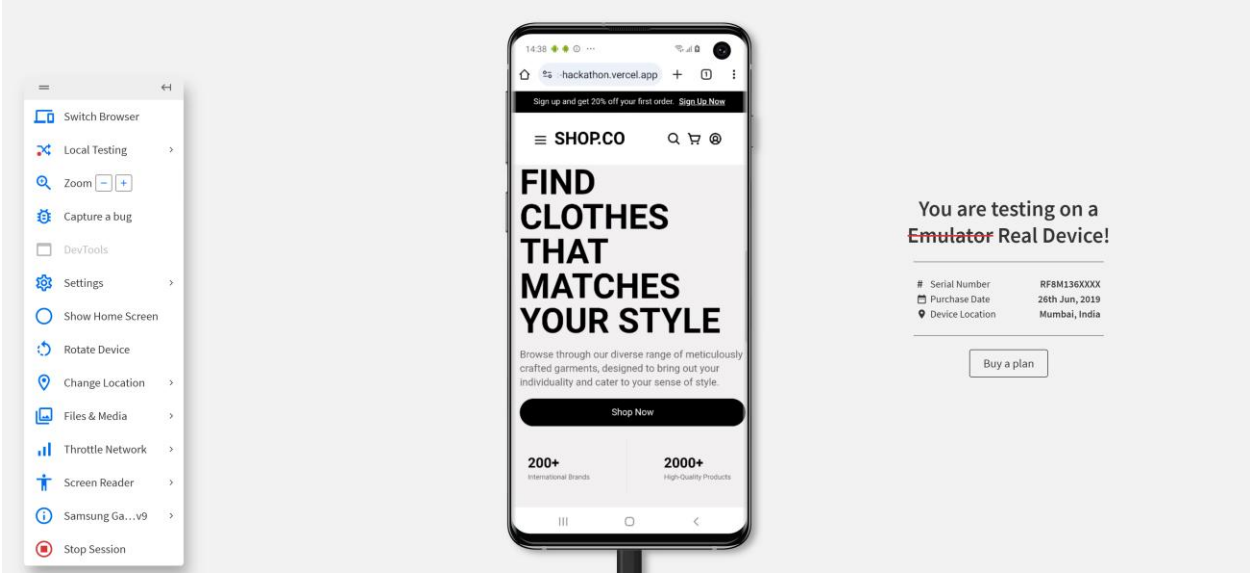


First Contentful Paint	989ms	998ms	1.2s	1.1s	1.2s	1.3s
Speed Index	1.5s	1.1s	2.3s	1.9s	2.2s	2.9s
Largest Contentful Paint	2.3s	1.8s	2.4s	2.3s	1.5s	1.3s
Total Blocking Time	402ms	119ms	160ms	158ms	73ms	119ms
Cumulative Layout Shift	0	0	0	0	0	0

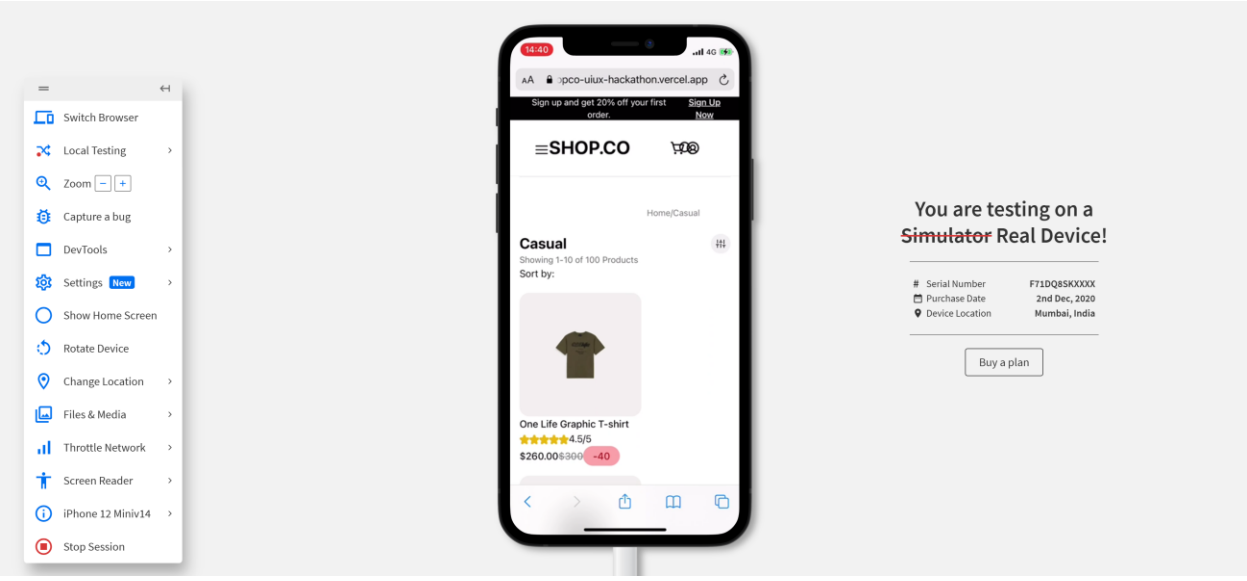


- **Cross-Browser and Device Testing**: Ensuring responsiveness and compatibility on various browsers and devices.

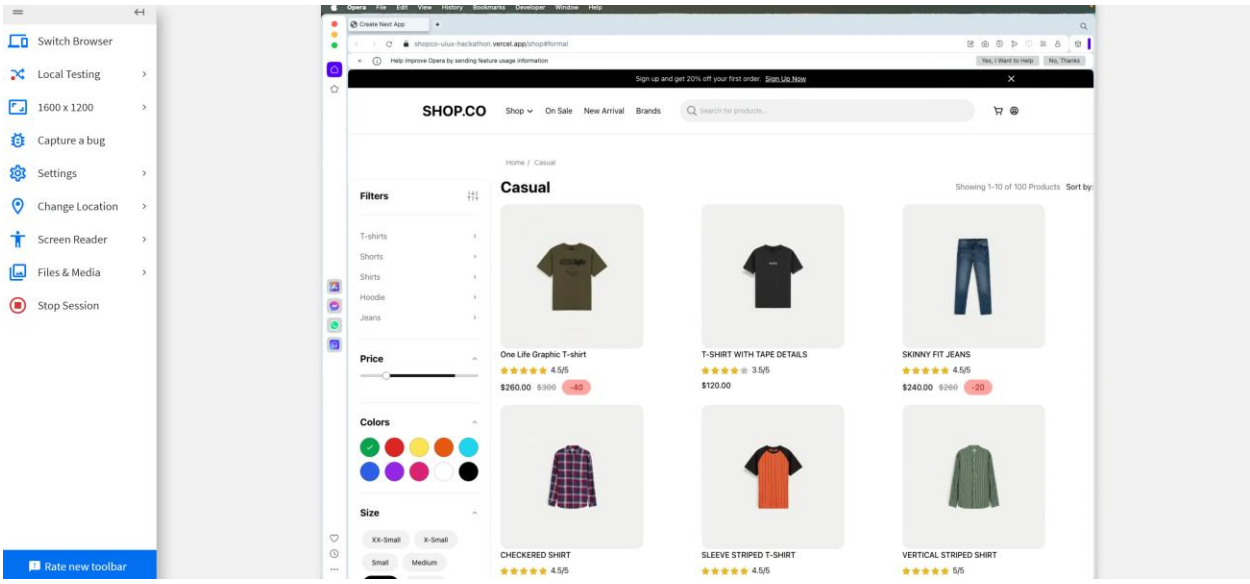
Chrome in Samsung Galaxy-S10:



Firefox in Iphone-12:



Opera in 1600 x 1200 Tablet:



- **Error Handling Testing**: Verifying those fallback mechanisms provide a seamless user experience.

Error Handling using Try-catch:

```
export async function POST(request: NextRequest) {
    try {
        const data = await request.json()
        await data.map((product: ProductType) => {
            return client.create({
                _type: 'catalog',
                name: product.name,
                price: product.price,
                category: product.category,
                discountPercent: product.discountPercent,
                isNew: product.isNew,
                colors: product.colors,
                sizes: product.sizes,
                description: product.description,
                imageUrl: product.imageUrl ? {
                    _type: 'image',
                    asset: {
                        _type: 'reference',
                        _ref: product.imageUrl
                    },
                } : null,
            })
        })
        return NextResponse.json({ message: 'Catalog created successfully', data });

    } catch (error) {
        return NextResponse.json({ error: `Failed to create catalog ${error}` }, { status: 500 });
    }
}
```

```
try {
    const filePath = path.join(process.cwd(), 'public', 'item', `${imageName}`);
    const imageFile = createReadStream(filePath);
    const imageAsset = await client.assets.upload("image", imageFile, { filename: basename(filePath) })
    console.log(`Image Uploaded Successfully: ${imageAsset._id}`);
    return imageAsset._id
} catch (error) {
    return `Failed to upload image, ${error}`

}
```

```
const response = await fetch("https://template1-neon-nu.vercel.app/api/products")
data = await response.json()
data = await Promise.all(data.map(async (productData: ProductType) => {
    try {
        const imageResponse = await fetch(productData.imageUrl);

        if (!imageResponse.ok) {
            throw new Error(`Failed to fetch image: ${productData.imageUrl}`);
        }

        const arrayBuffer = await imageResponse.arrayBuffer();
        const imageBuffer = Buffer.from(arrayBuffer);
        const imageAsset = await client.assets.upload("image", imageBuffer);

        return {
            ...productData,
            imageUrl: imageAsset._id
        };
    } catch (error) {
        console.error(`Error processing product ${productData.name}:`, error);
        return productData;
    }
```

Fallbacks In case no product is available to Fetch:

```
<Container className="relative overflow-x-hidden">
  {catalog.length > 0 ? (
    catalog.map(
      (product) =>
        product._id === id && (
          <div key={product._id} className="flex flex-col">
            <BreadCrumb location={["Home", "Shop", "Men", "T-shirts"]} />
            <Product product={product} />
          </div>
        )
    )
  ) : (
    <p>No products available.</p>
  )}
  <FeedbackTabs />
  <DisplayCard heading="You might also like" button={false} id={id} />
</Container>
```

2. Observations:

Performance on lower side mainly because of images not optimized can be further improved by optimizing image delivery and enabling caching.

Compress large images and enable lazy loading for non-critical assets.