

Days 5

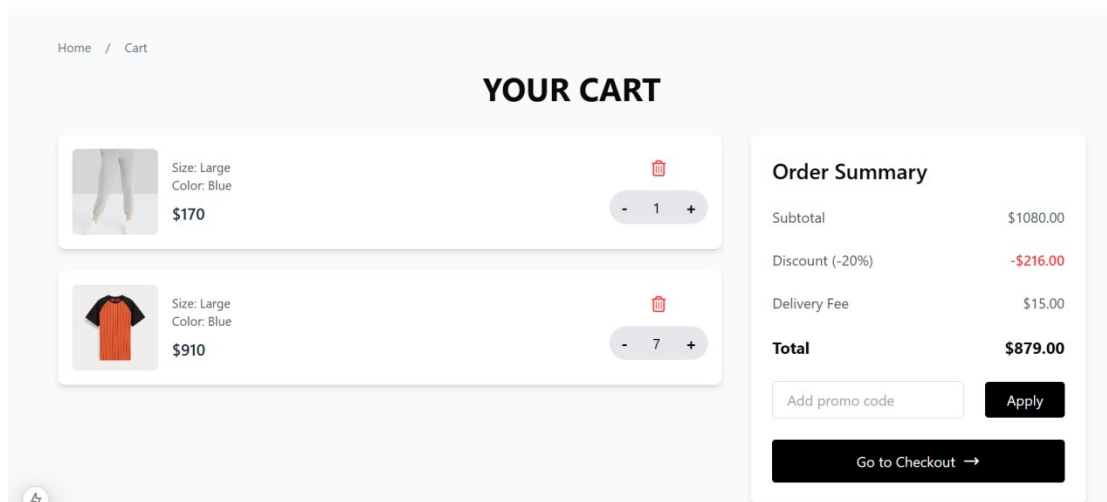
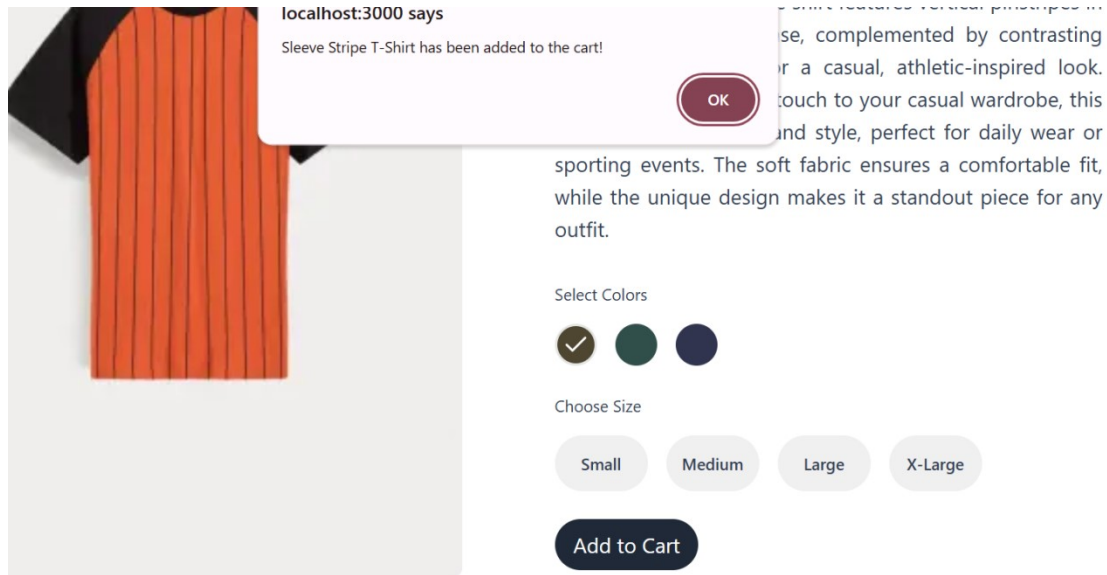
Fronted Testing

1. Add to Cart Functionaty

Code Screenshot

```
src > app > Cart > ⚙️ page.tsx > ...
1  |use client";
2  |import { FaRegTrashCan } from "react-icons/fa6";
3  |import React from "react";
4  |import { useDispatch, useSelector } from "react-redux";
5  |import { RootState } from "../redux/Store";
6  |import Image from "next/image";
7  |import { remove, incrementQuantity, decrementQuantity } from "../redux/Cartslice";
8  |import Link from "next/link";
9
10 |interface CartItem {
11 |  id: number;
12 |  title: string;
13 |  price: number;
14 |  imageUrl: string;
15 |  quantity: number;
16 |}
17
18 |const Cartpage: React.FC = () => {
19 |  const dispatch = useDispatch();
20 |  const cartItems = useSelector((state: RootState) => state.cart);
21
22 |  const handleRemove = (id: number) => {
23 |    dispatch(remove(id));
24 |  };
25
26 |  const handleIncrement = (id: number) => {
27 |    dispatch(incrementQuantity(id));
28 |  };
29
30 |  const handleDecrement = (id: number) => {
31 |    dispatch(decrementQuantity(id));
32 |  };
33
34 |  const subtotal = cartItems.reduce(
35 |    (total, item) => total + item.price * item.quantity,
36 |    0
37 |  );
38 |  const discount = subtotal * 0.2; // 20% discount
39 |  const deliveryFee = 15; // Fixed delivery fee
40 |  const total = subtotal - discount + deliveryFee;
41
42 |  return (
43 |    <div className="min-h-screen bg-gray-50 py-8 px-4 flex flex-col items-center">
44 |      { /* Breadcrumb */ }
45 |      <div className="flex gap-4 text-sm text-gray-500 mb-4 w-full max-w-6xl">
46 |        <Link href="/"> <span> Home</span> </Link>
47 |        <span></span>
48 |        <span> Cart</span>
49 |      </div>
50
51 |      { /* Page Title */ }
52 |      <h3 className="text-4xl font-bold text-center mb-8">YOUR CART</h3>
```

Display Functionaty



1. Product Listing Component:

- Render product data dynamically in a grid layout.
- Include fields like:

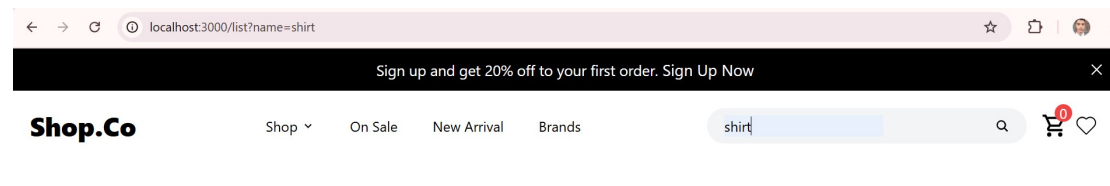
- o Product Name
- o Price
- o Image
- o Stock Status

2. Product Detail Component: .

Create individual product detail pages using dynamic routing in Next.js. Include detailed fields such as:

- o Product Description
- o Price o Available Sizes or Colors

3. Search Bar:



Expected Output:

1. Fully tested and functional marketplace components, validated against professional testing standards.
2. Clear and user-friendly error handling mechanisms implemented across all functionalities
3. Optimized performance with faster load times and smoother interactions.
4. A responsive design tested thoroughly on multiple browsers and devices.
5. A comprehensive CSV-based testing report documenting test cases, results, and resolutions.
6. Well-structured documentation summarizing all testing and optimization efforts.
7. Error handling mechanisms with clear messages and fallback UI.
8. Optimized performance for faster page load times.
9. A responsive design verified on multiple browsers and devices.
10. Comprehensive documentation of testing and fixes.

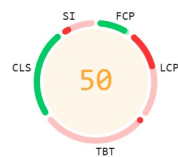
PERFORMANCE WEBSITE

http://localhost:3000/

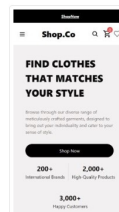


There were issues affecting this run of Lighthouse:

- The tested device appears to have a slower CPU than Lighthouse expects. This can negatively affect your performance score. Learn more about [calibrating an appropriate CPU slowdown multiplier](#).



Values are estimated and may vary. The [performance score is](#)

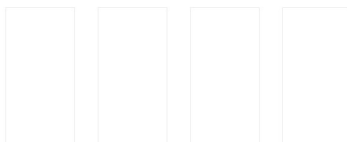


● First Contentful Paint
1.7 s

▲ Total Blocking Time
2,810 ms

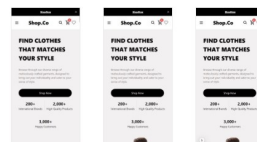
▲ Speed Index
6.8 s

View Treemap



▲ Largest Contentful Paint
4.1 s

● Cumulative Layout Shift
0



Show audits relevant to: [All](#) [FCP](#) [LCP](#) [TBT](#)



Best Practices

TRUST AND SAFETY

- ☐ Ensure CSP is effective against XSS attacks ▼
- ☐ Use a strong HSTS policy ▼
- ☐ Ensure proper origin isolation with COOP ▼



PASSED AUDITS (25)

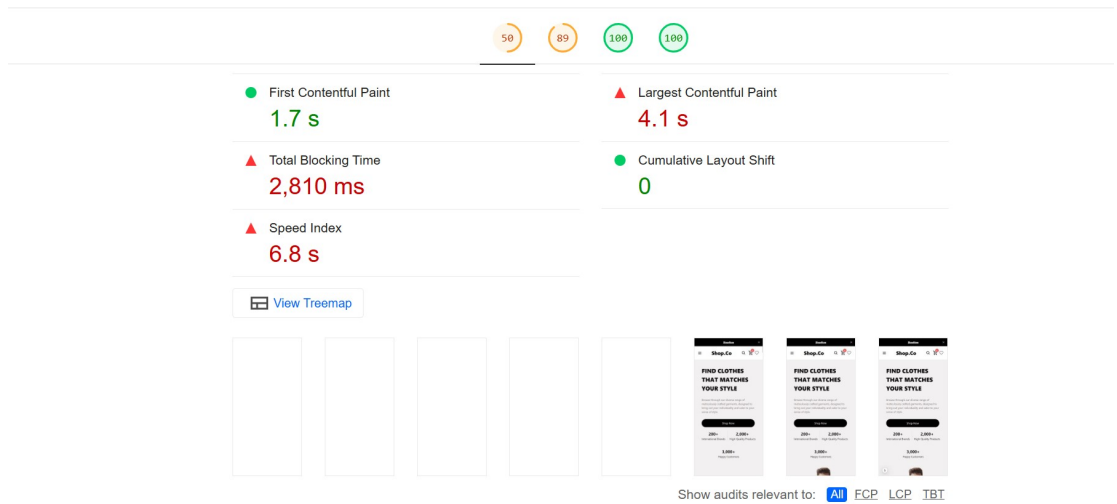
Show



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



1	Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Assigned To	Remarks
2	TC001	Validate product listing page	Open product page > Verify products	Products displayed correctly	Products displayed correctly	Passed	Low	-	No issues found
3	TC002	Test API error handling	Disconnect API > Refresh page	Show fallback UI with error message	Error message shown	Passed	Medium	-	Handled gracefully
4	TC003	Check cart functionality	Add product to cart > Verify cart contents	Cart updates with added product	Cart updates as expected	Passed	High	-	Works as expected
5	TC004	Ensure responsiveness on mobile	Resize browser window > Check layout	Layout adjusts properly to screen size	Responsive layout working as intended	Passed	Medium	-	Test successful

ALI ASGHAR M. RIZWAN