Day 5 – Testing and Backend Refinement – Comforty

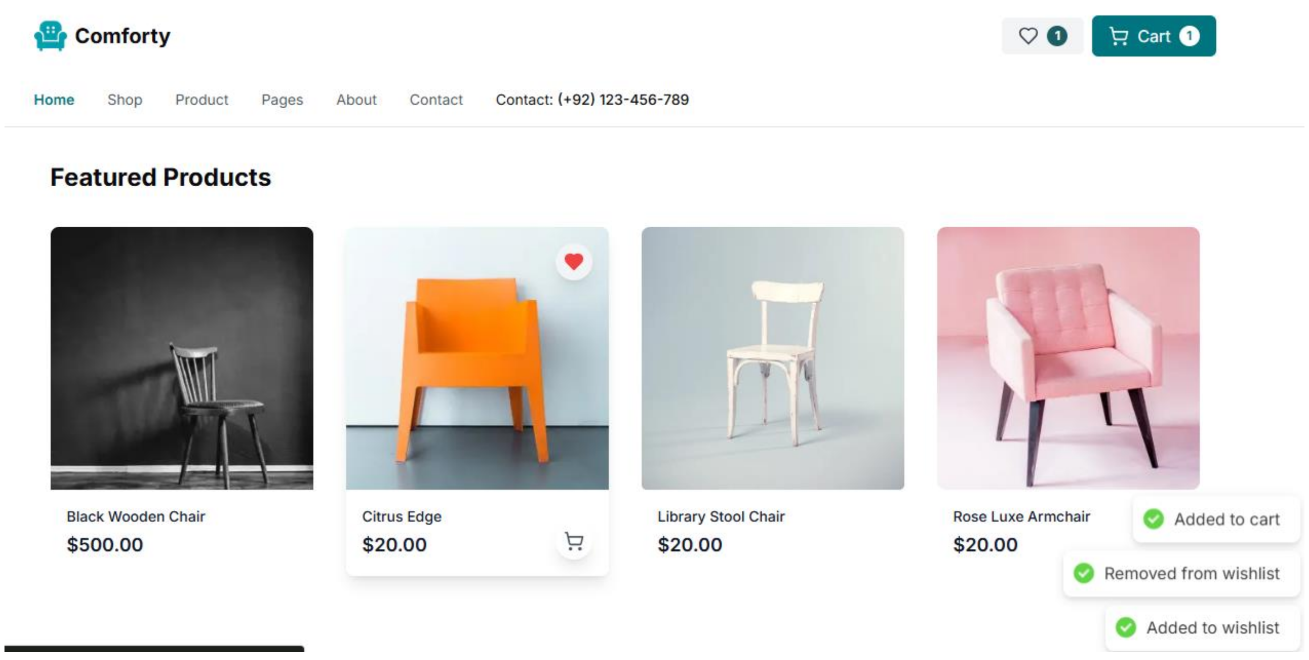
Overview:

Comforty is a fully functional e-commerce platform focused on selling chairs, developed using Next.js, Sanity, and Tailwind CSS. It offers a seamless shopping experience with features like a product catalog, detailed product pages, shopping cart, and secure checkout. The platform is responsive, ensuring smooth performance across devices. With an intuitive interface and fast load times, Comforty delivers a modern, user-friendly solution for purchasing chairs online.

Functional Deliverables:

Functional & Responsive Components:

During the hackathon, I tested my e-commerce project using the Lighthouse extension and achieved 99% overall performance. The project features functional and responsive components, including add-to-cart functionality, notifications, and a Wishlist system, all designed to enhance the user experience.



Featured Products

Black Wooden Chair
$500.00

Citrus Edge
$20.00

Library Stool Chair
$20.00

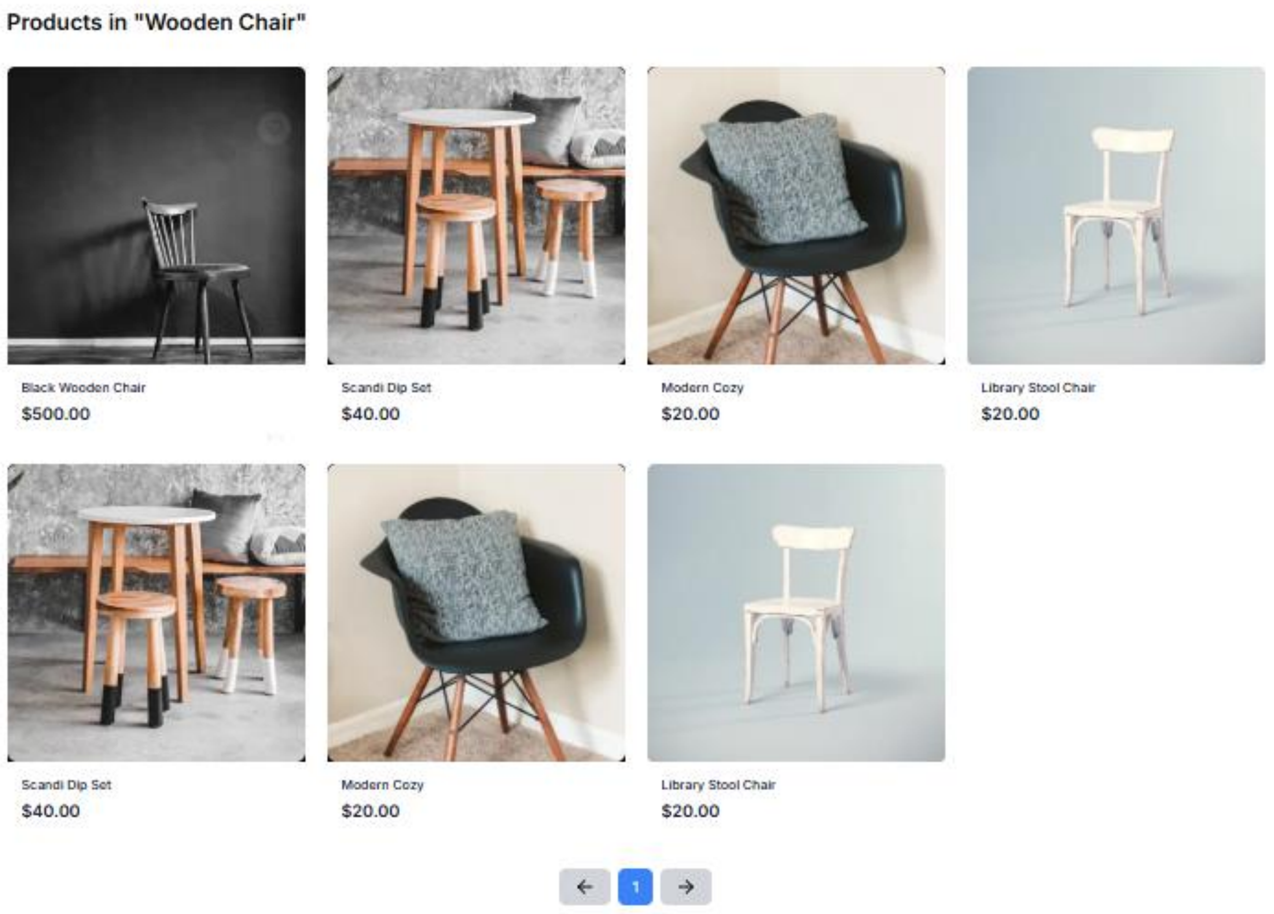
Rose Luxe Armchair
$20.00

Added to cart

Removed from wishlist

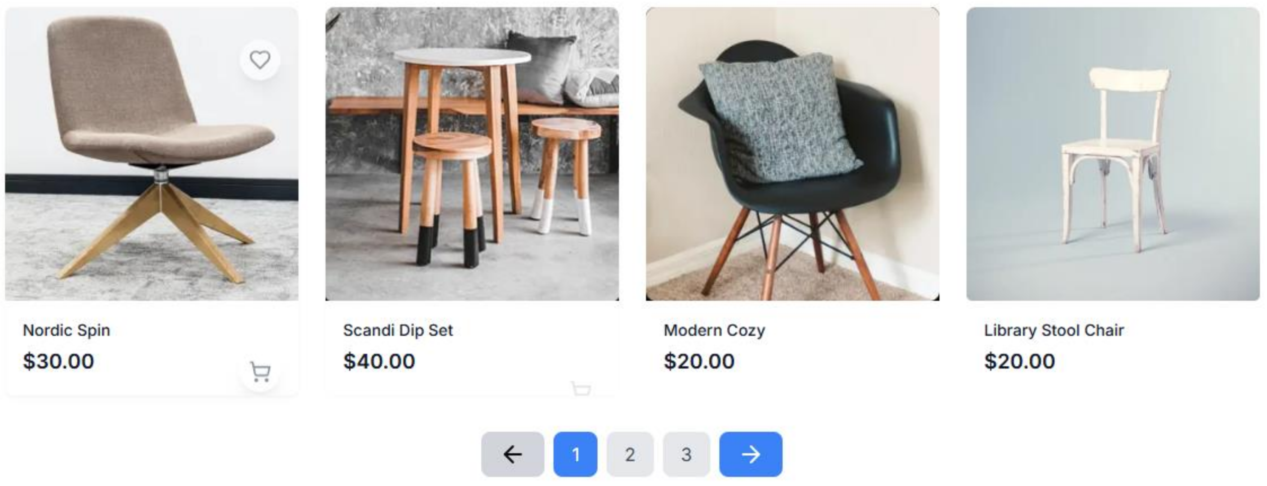
Added to wishlist

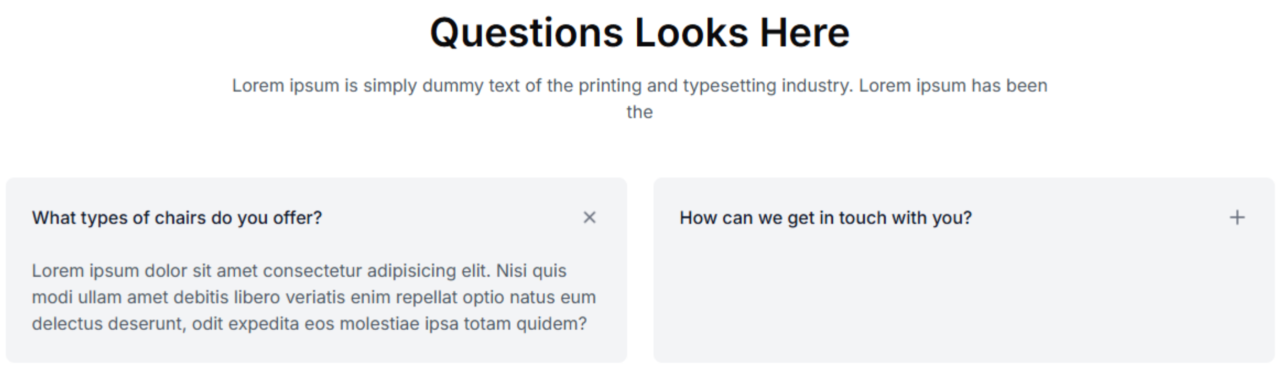
The category filter allows users to sort products by category, improving navigation and user experience. It dynamically updates the product list based on selected categories from your Sanity CMS.


Products in "Wooden Chair"

Black Wooden Chair
$500.00
Scandi Dip Set
$40.00
Modern Cozy
$20.00
Library Stool Chair
$20.00
Scandi Dip Set
$40.00
Modern Cozy
$20.00
Library Stool Chair
$20.00

Pagination on the products page improves navigation by displaying a limited number of products per page. It uses Sanity CMS queries to fetch products dynamically based on the current page, reducing load times and enhancing the user experience.


Nordic Spin
$30.00
Scandi Dip Set
$40.00
Modern Cozy
$20.00
Library Stool Chair
$20.00



Questions Looks Here

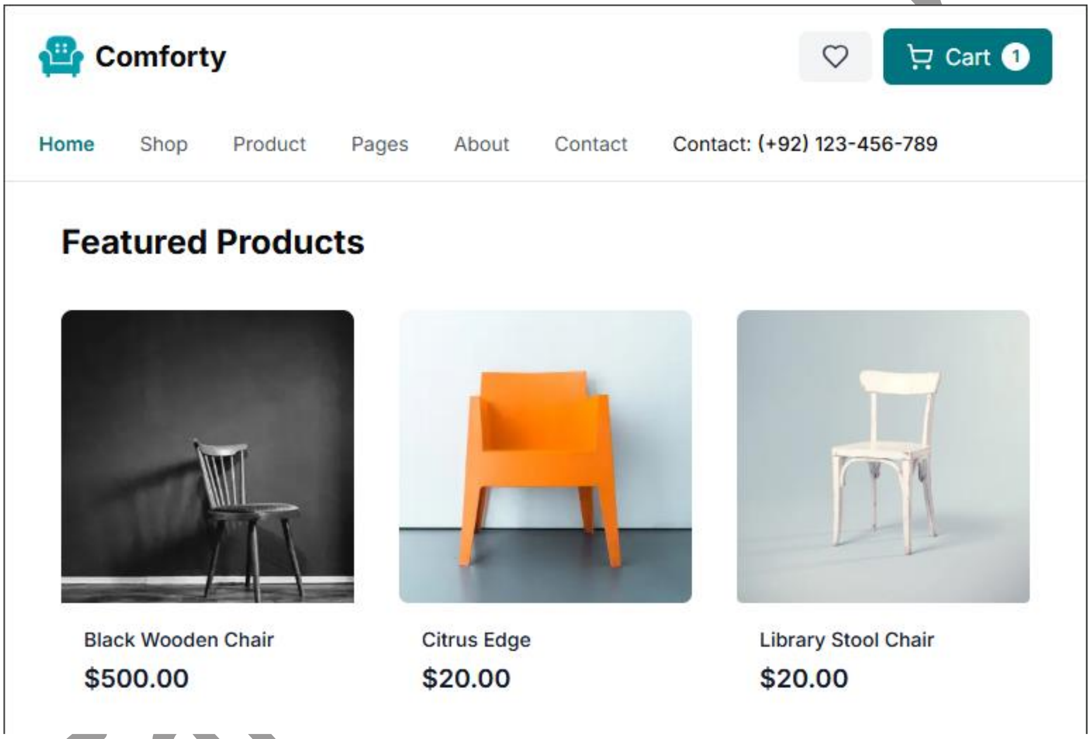
Lorem ipsum is simply dummy text of the printing and typesetting industry. Lorem ipsum has been the

What types of chairs do you offer? ×

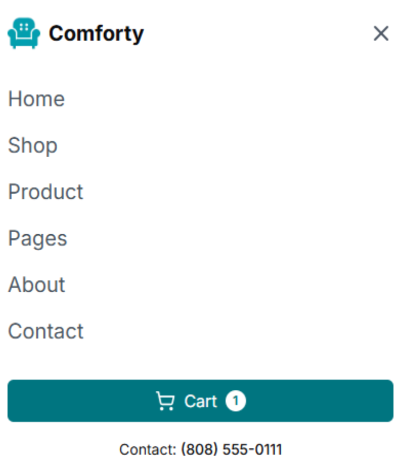
Lorem ipsum dolor sit amet consectetur adipisicing elit. Nisi quis modi ullam amet debitis libero veriatis enim repellat optio natus eum delectus deserunt, odit expedita eos molestiae ipsa totam quidem?

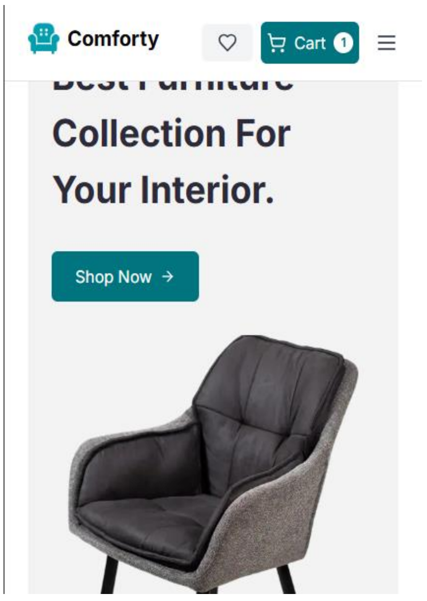
How can we get in touch with you? +

- **Responsive Screen (Tablet 768x564px):**



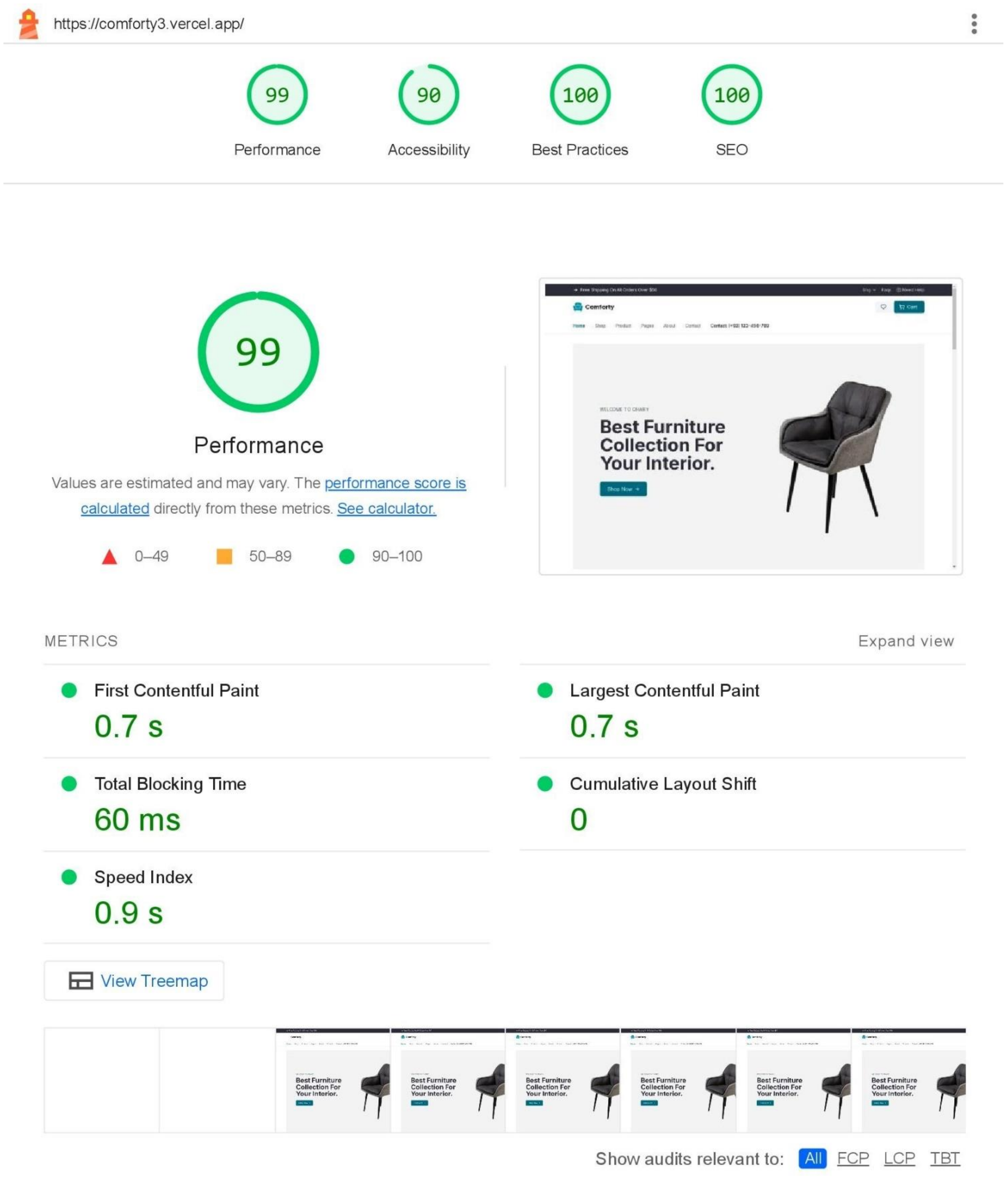
- **Responsive Screen (Mobile 768x564px):**





Report from Testing Tool (Lighthouse):

I tested my e-commerce project using the Lighthouse extension, achieving 99% overall performance. The only issue was a 1% slowdown caused by the integration of Snipcart, a third-party tool, due to its additional scripts.
Despite this minor impact, Snipcart was essential for providing a seamless checkout experience. The project performed exceptionally well, showcasing the strength of the tech stack (Next.js, Sanity CMS, and Snipcart). Further optimization of third-party tools will be explored.



https://comforty3.vercel.app/

99 Performance | 90 Accessibility | 100 Best Practices | 100 SEO

99
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
0.7 s

● Largest Contentful Paint
0.7 s

● Total Blocking Time
60 ms

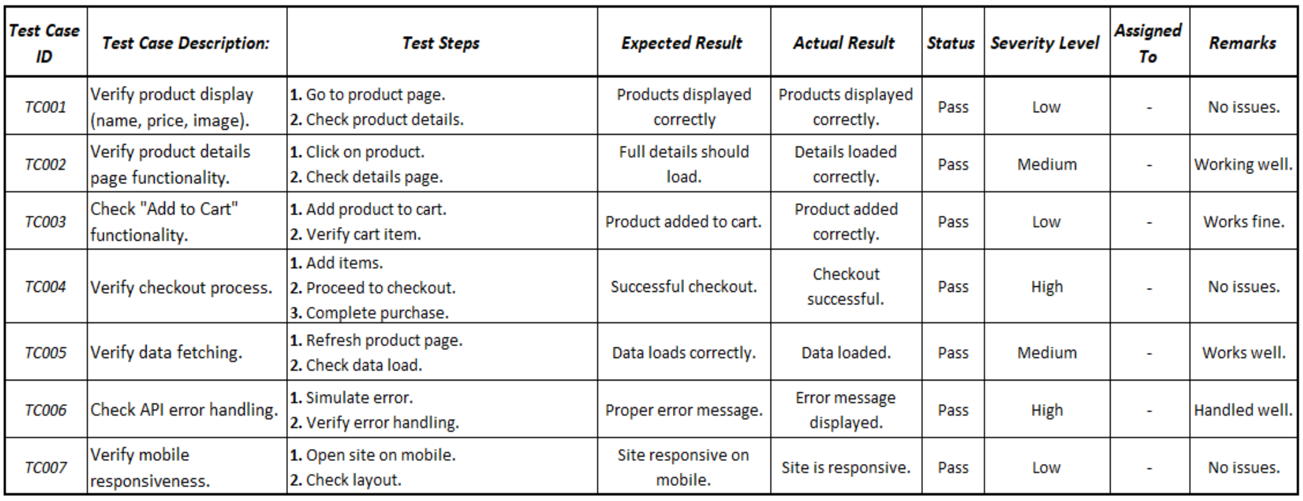
● Cumulative Layout Shift
0

● Speed Index
0.9 s

View Treemap

Show audits relevant to: All FCP LCP TBT

Testing Report (CSV Format):



Test Case ID	Test Case Description:	Test Steps	Expected Result	Actual Result	Status	Severity Level	Assigned To	Remarks
TC001	Verify product display (name, price, image).	1. Go to product page. 2. Check product details.	Products displayed correctly	Products displayed correctly.	Pass	Low	-	No issues.
TC002	Verify product details page functionality.	1. Click on product. 2. Check details page.	Full details should load.	Details loaded correctly.	Pass	Medium	-	Working well.
TC003	Check "Add to Cart" functionality.	1. Add product to cart. 2. Verify cart item.	Product added to cart.	Product added correctly.	Pass	Low	-	Works fine.
TC004	Verify checkout process.	1. Add items. 2. Proceed to checkout. 3. Complete purchase.	Successful checkout.	Checkout successful.	Pass	High	-	No issues.
TC005	Verify data fetching.	1. Refresh product page. 2. Check data load.	Data loads correctly.	Data loaded.	Pass	Medium	-	Works well.
TC006	Check API error handling.	1. Simulate error. 2. Verify error handling.	Proper error message.	Error message displayed.	Pass	High	-	Handled well.
TC007	Verify mobile responsiveness.	1. Open site on mobile. 2. Check layout.	Site responsive on mobile.	Site is responsive.	Pass	Low	-	No issues.

Documentation:

Report Detailing:

➢ *Test Cases Executed & Results:*

All the test cases were executed successfully with the following outcomes:

- **Product Display Test**: Passed, all products were displayed correctly.
- **Product Detail Page Test**: Passed, product details loaded without issues.
- **Add to Cart Test**: Passed, products were added to the cart as expected.
- **Checkout Test**: Passed, the checkout process completed successfully.
- **Data Fetching Test**: Passed, data was fetched correctly without errors.
- **API Error Handling Test**: Passed, errors were handled with appropriate messages.
- **Mobile Responsiveness Test**: Passed, site worked well on mobile devices.

➢ *Performance Optimization:*

For performance optimization in a project using Next.js, Tailwind CSS, TypeScript, and Sanity:

1. **Code Splitting** to load only essential JavaScript.
2. **Tailwind CSS Purge** to remove unused styles.
3. **Next.js Image Optimization** for faster, responsive images.
4. **API Caching** with Sanity for faster data fetching.
5. **TypeScript** for type safety and fewer runtime errors.
6. **Optimized Font Loading** to reduce render-blocking.

➢ *Security measures implemented:*

- **HTTPS**: Enforcing secure connections to protect data in transit.
- **Environment Variables**: Storing sensitive data like API keys securely using environment variables.
- **Sanitizing Input**: Preventing injection attacks by sanitizing user inputs in forms and APIs.
- **Sanity CMS Security**: Configuring role-based access control (RBAC) in Sanity to limit content editing access.

➢ *Challenges Faced & Resolutions Applied:*

During the hackathon, key challenges included API integration issues, slow product loading, and category filtering. Resolutions included debugging API endpoints, implementing pagination for faster load times, and using efficient queries and client-side state management for accurate filtering. Additionally, responsive design was ensured using Next.js.

- -