# DAY 5 - TESTING, ERROR HANDLING, AND BACKEND

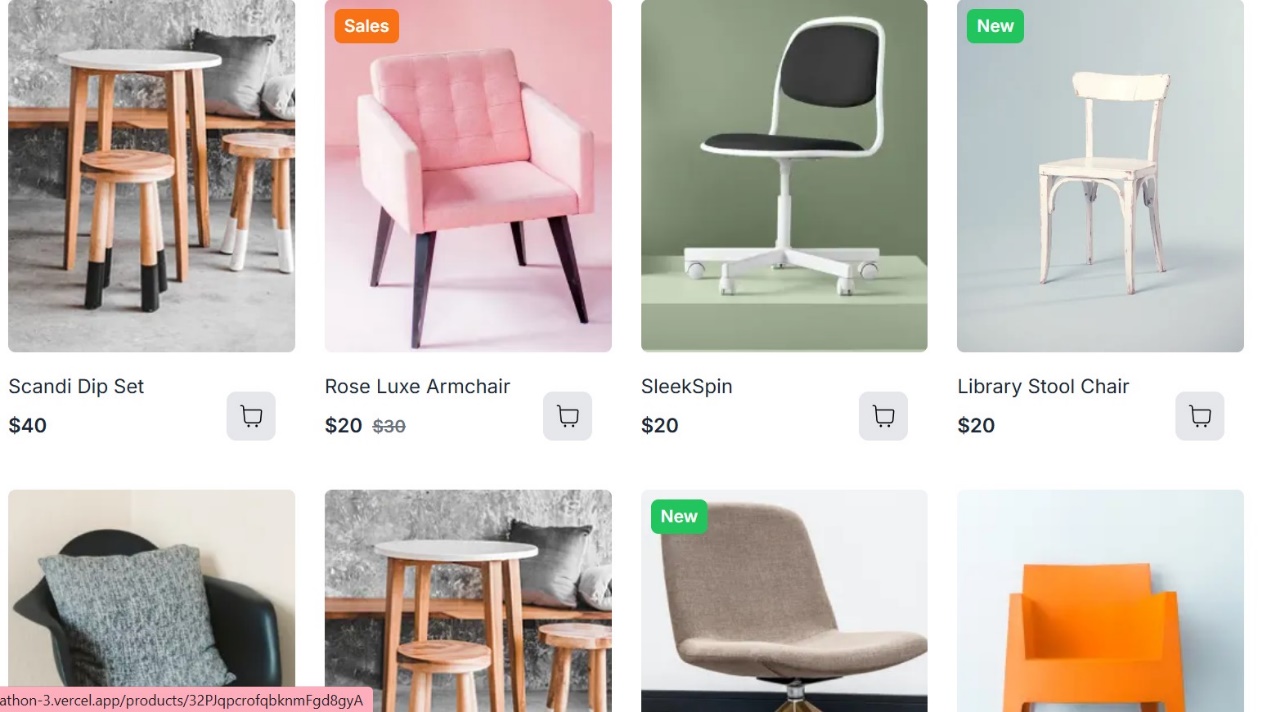
# INTEGRATION REFINEMENT

## **Document Title:** "Day 5-TESTING, ERROR HANDLING, AND BACKEND

## INTEGRATION REFINEMENT

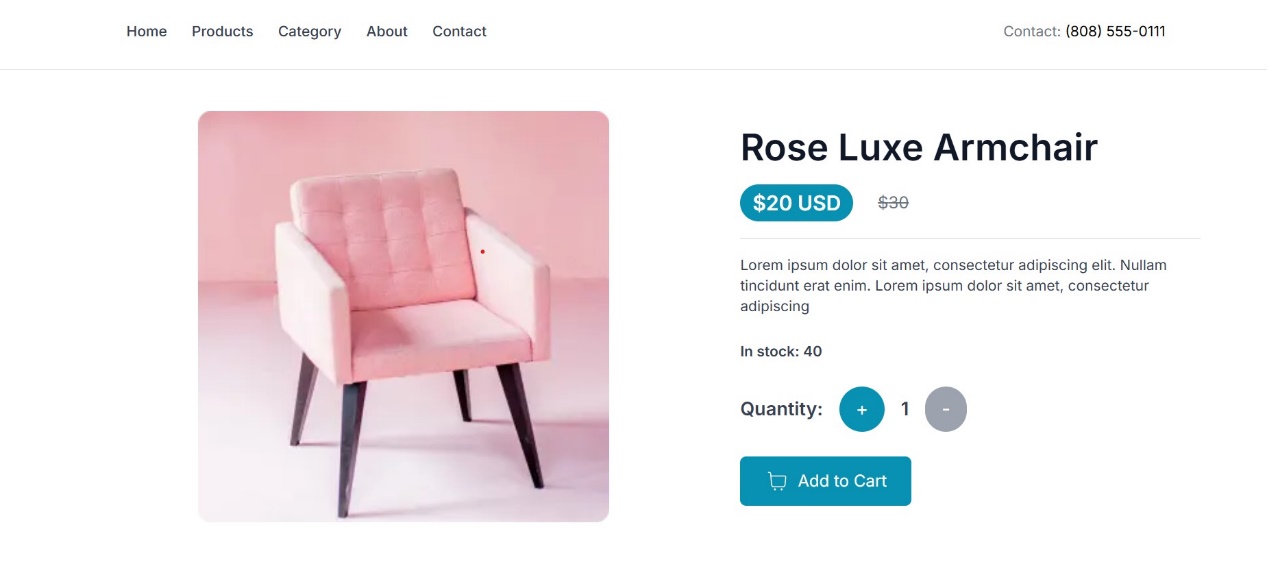
1. Functional Deliverable:

* **Product listing page with dynamic data:**

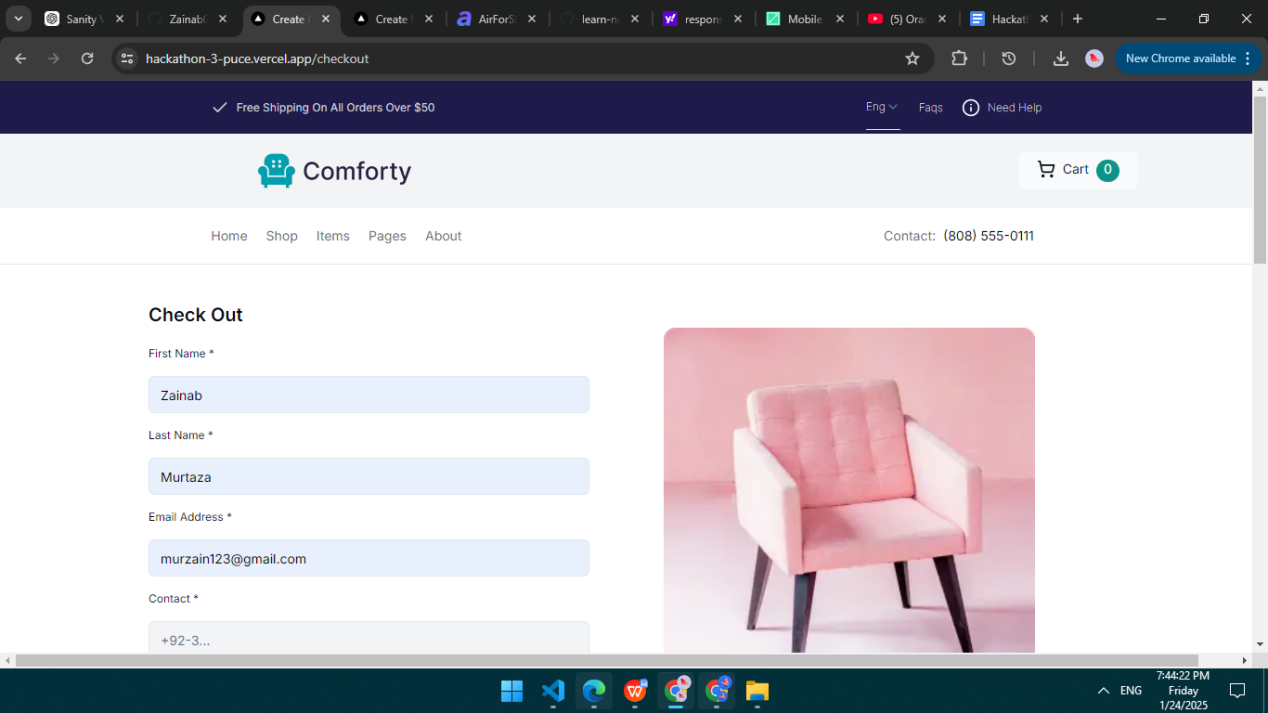


* **Screenshots of Individual product detail pages with accurate routing and**

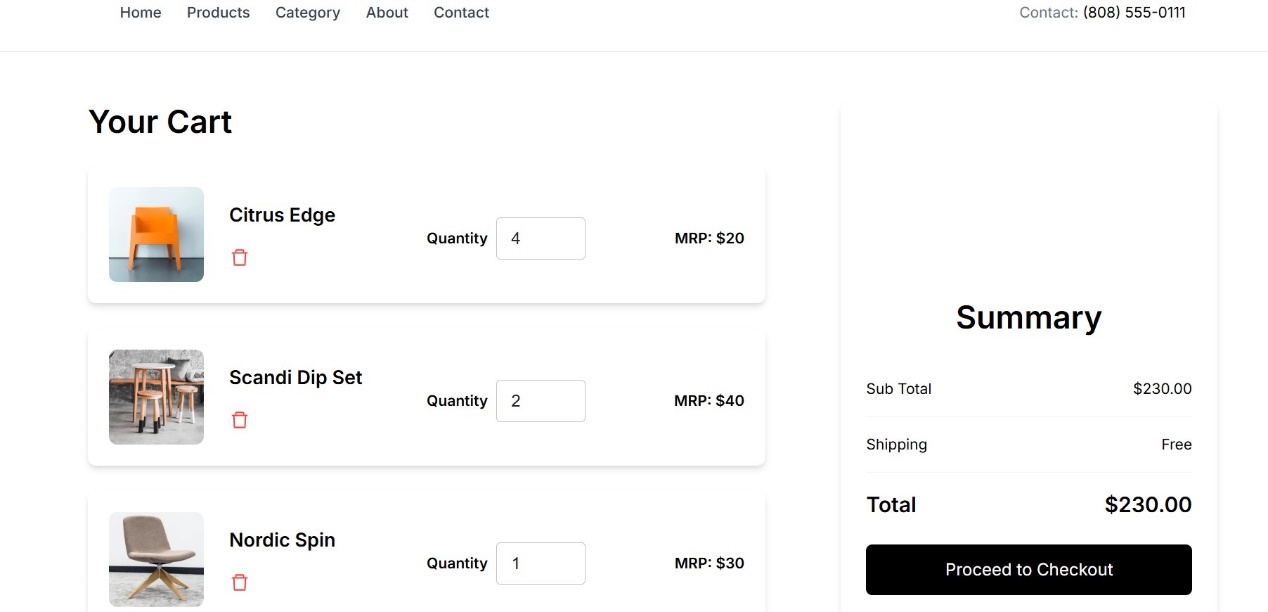
**data rendering**



* **Screen-short of CheckOut:**

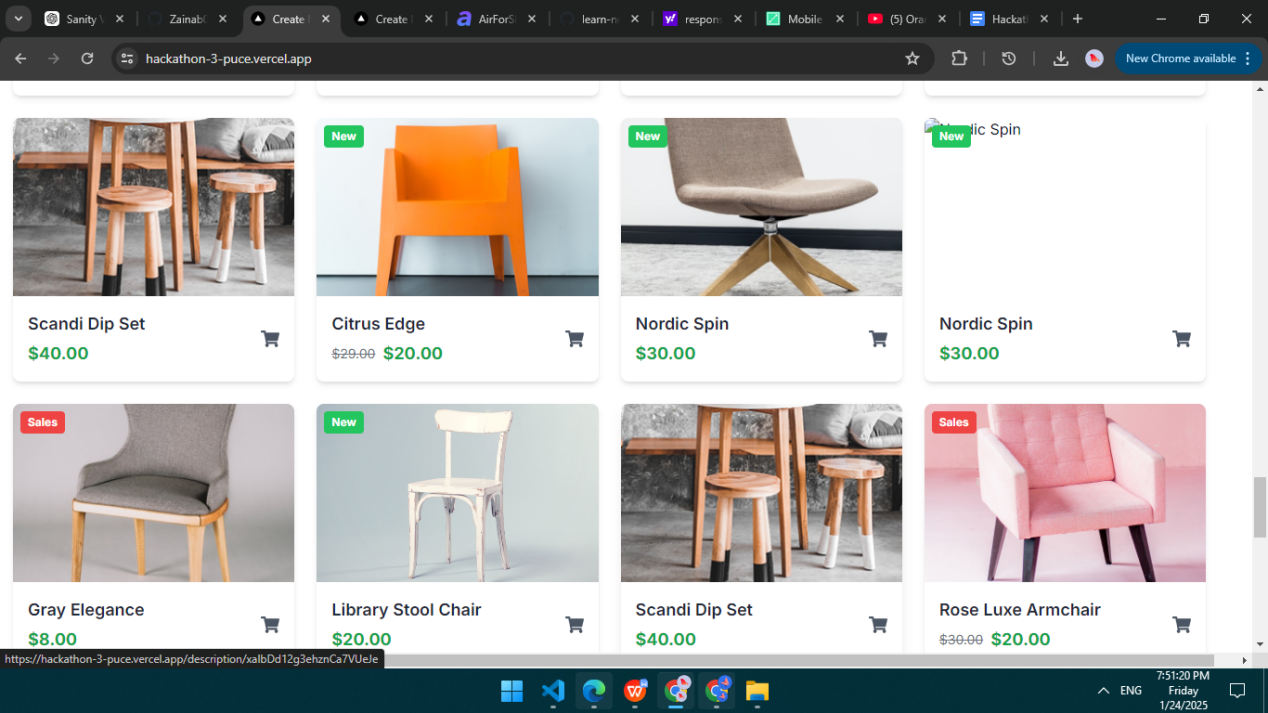
****

* **AddToCart Function:**

****

1. **Error Handling:**

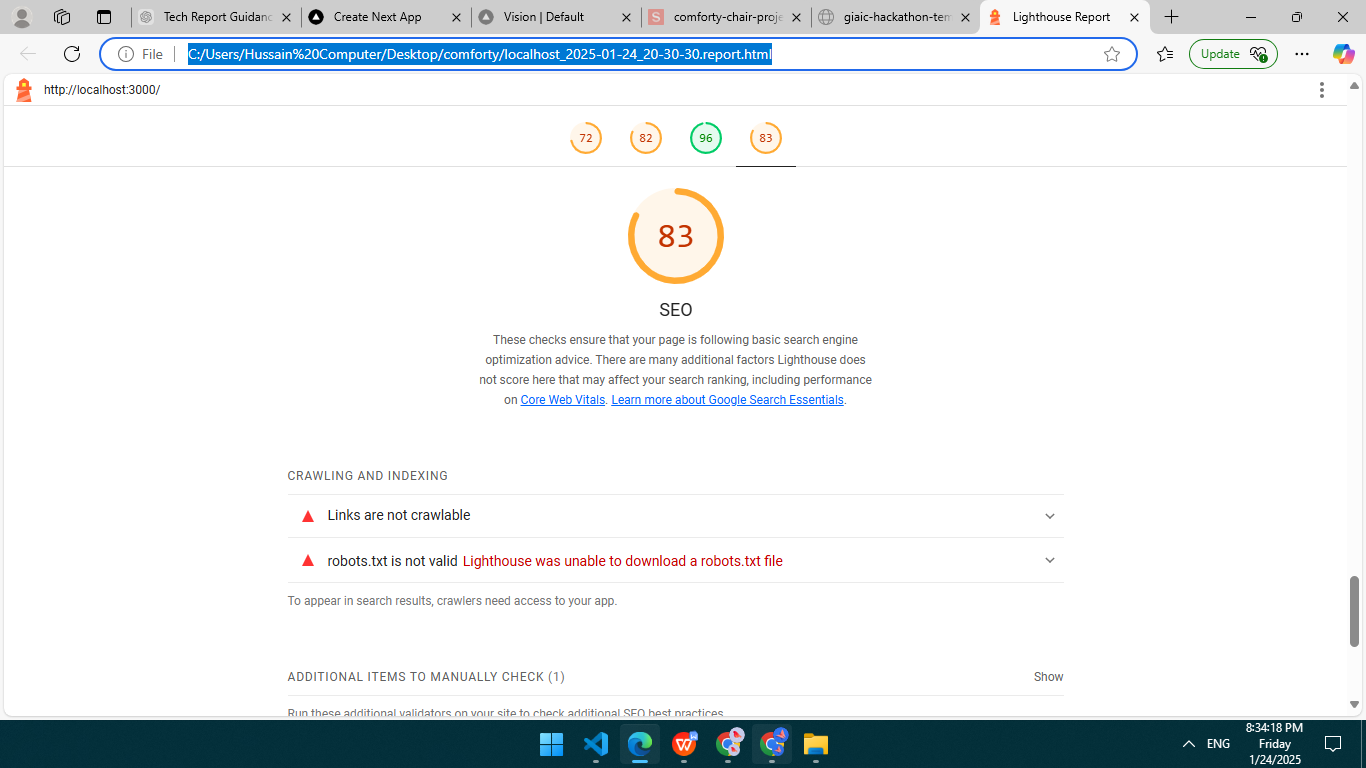
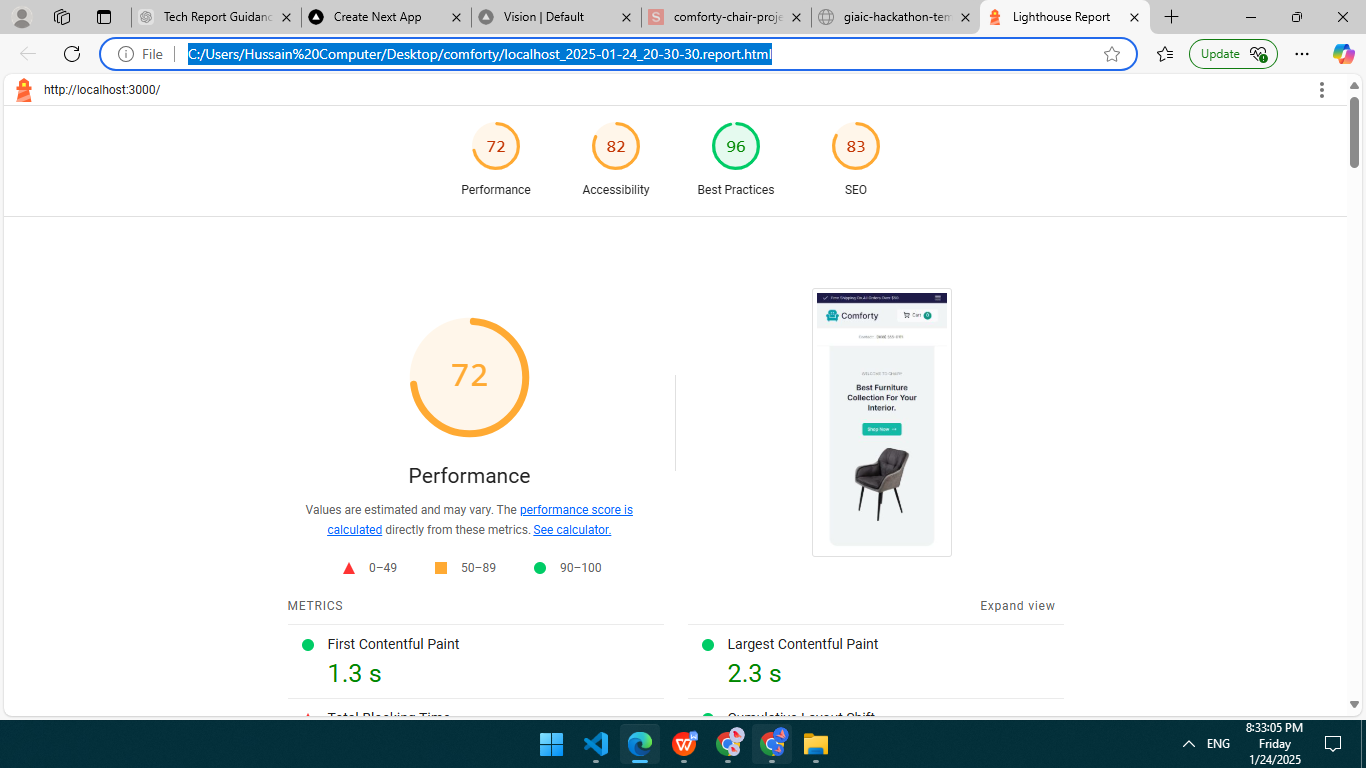
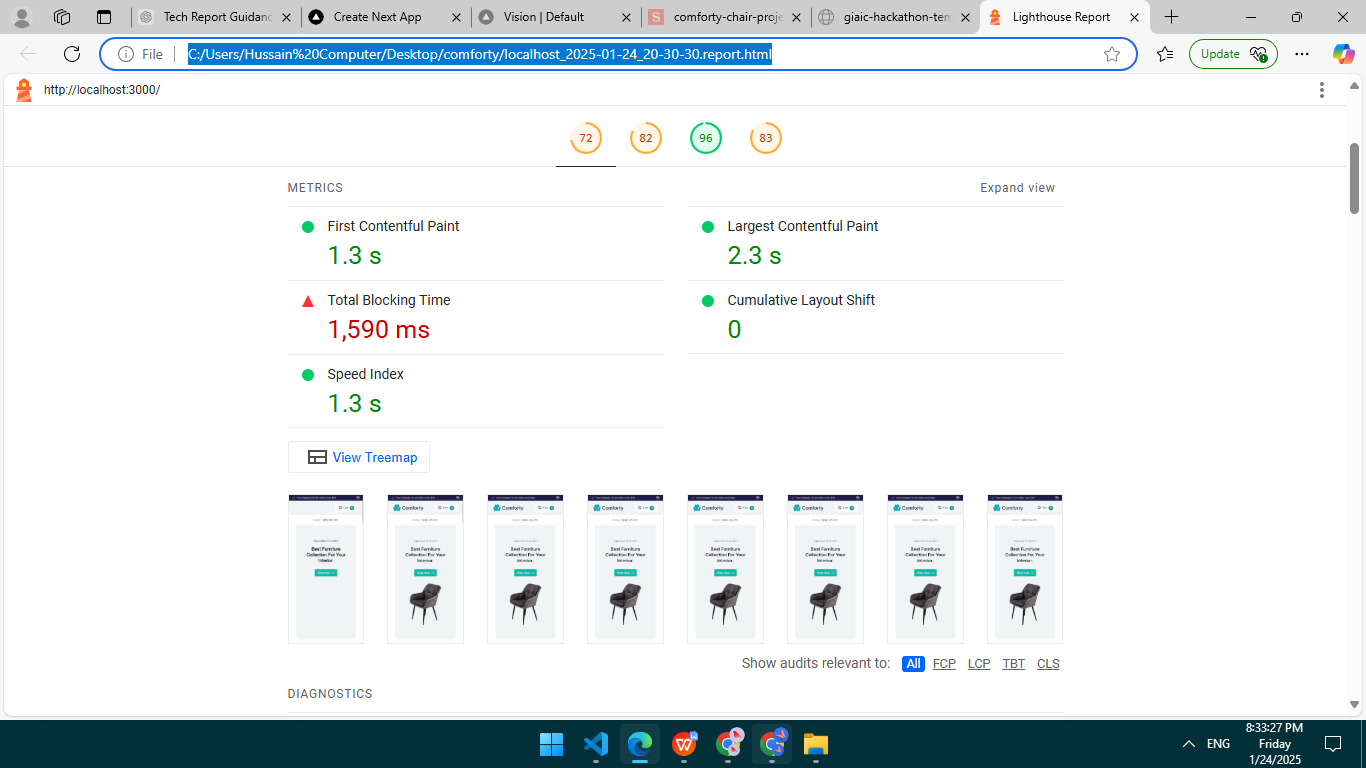
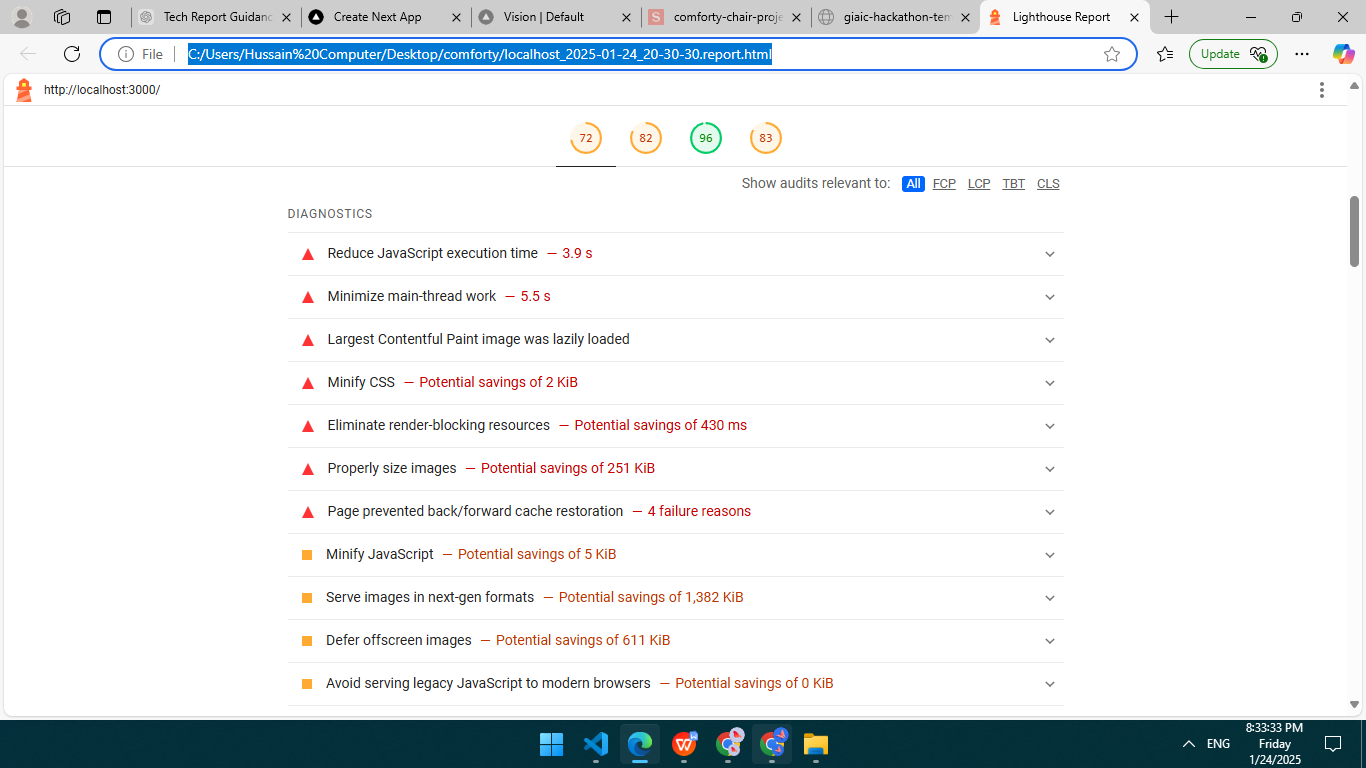
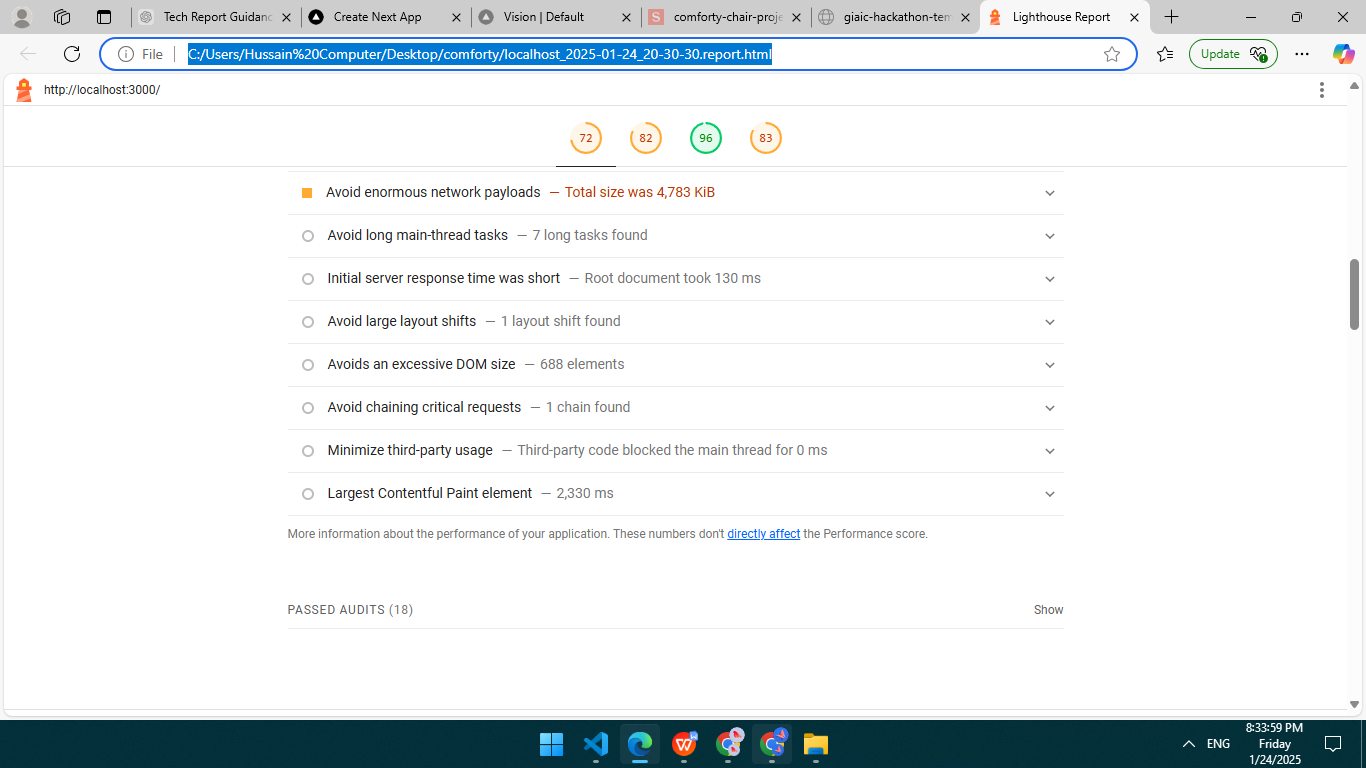
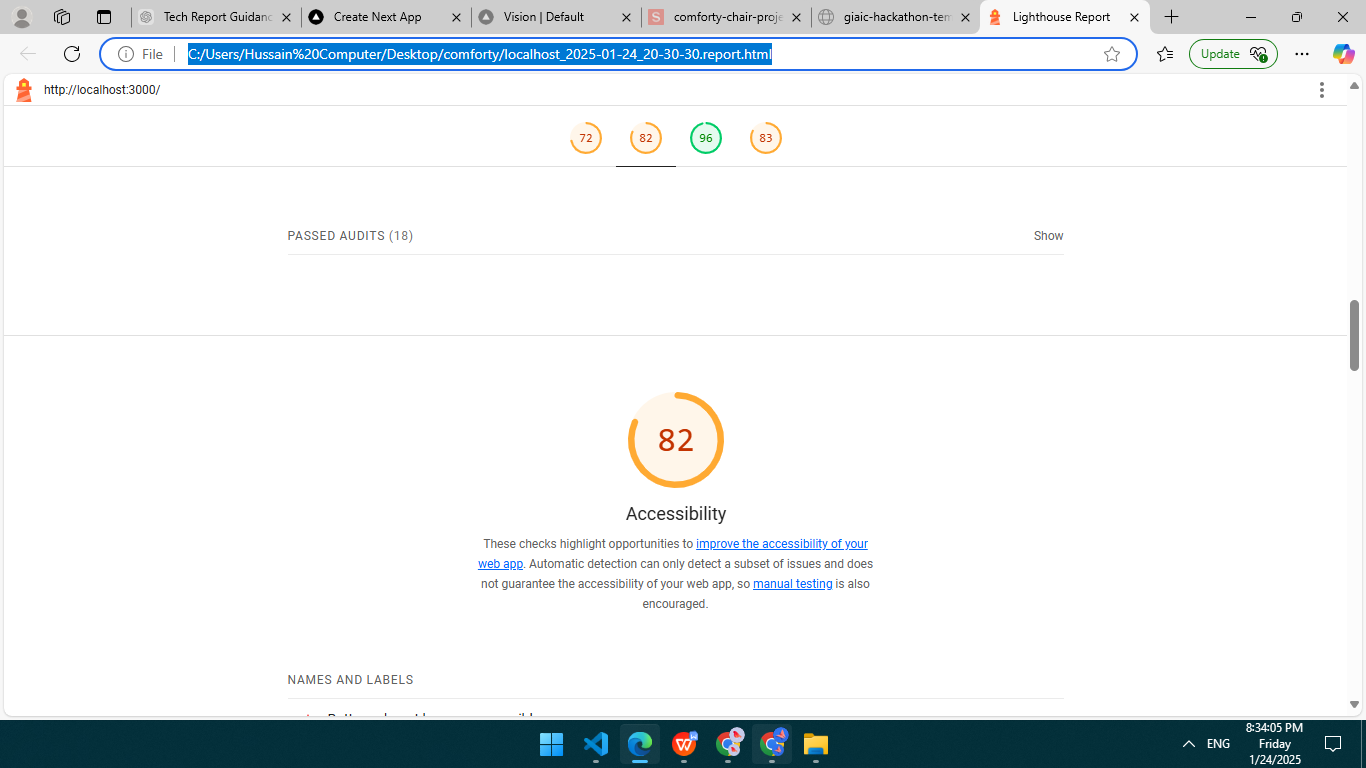
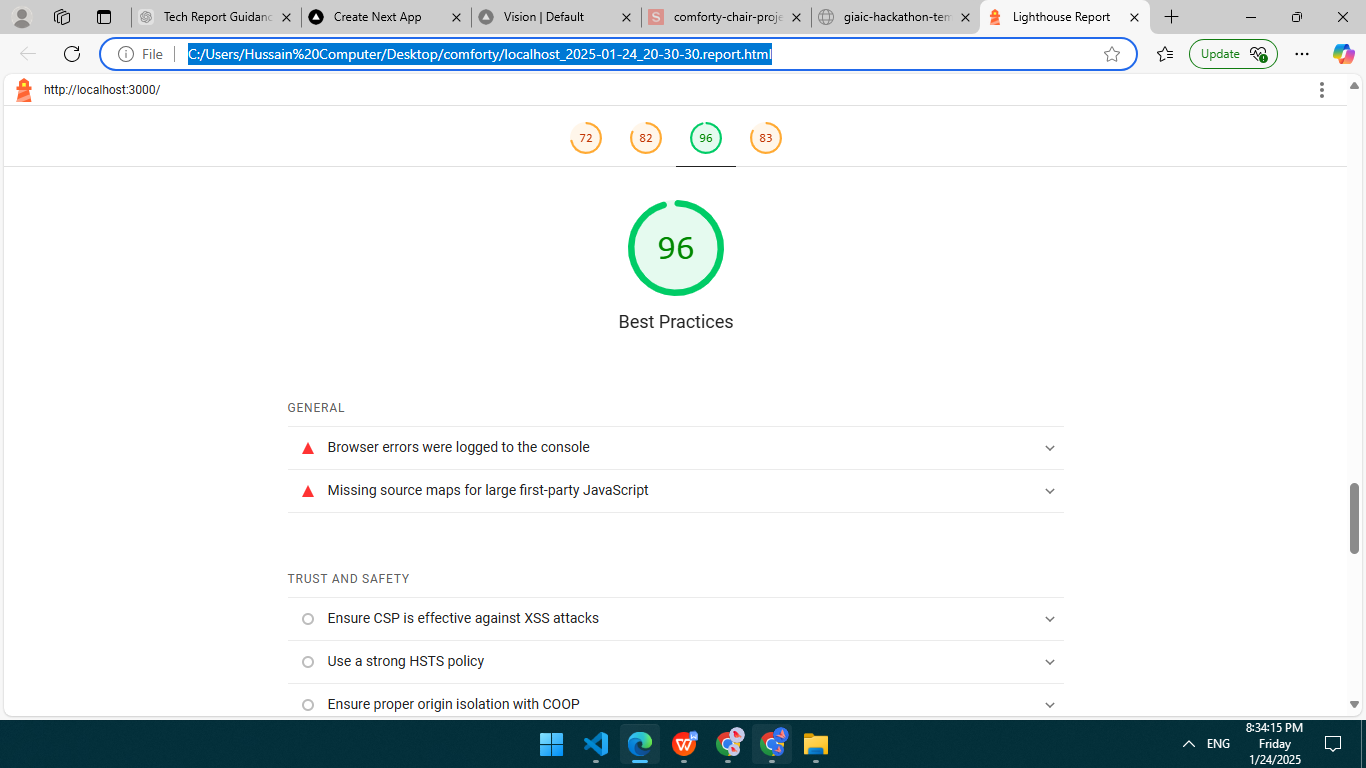
* **Error Handling:**

****

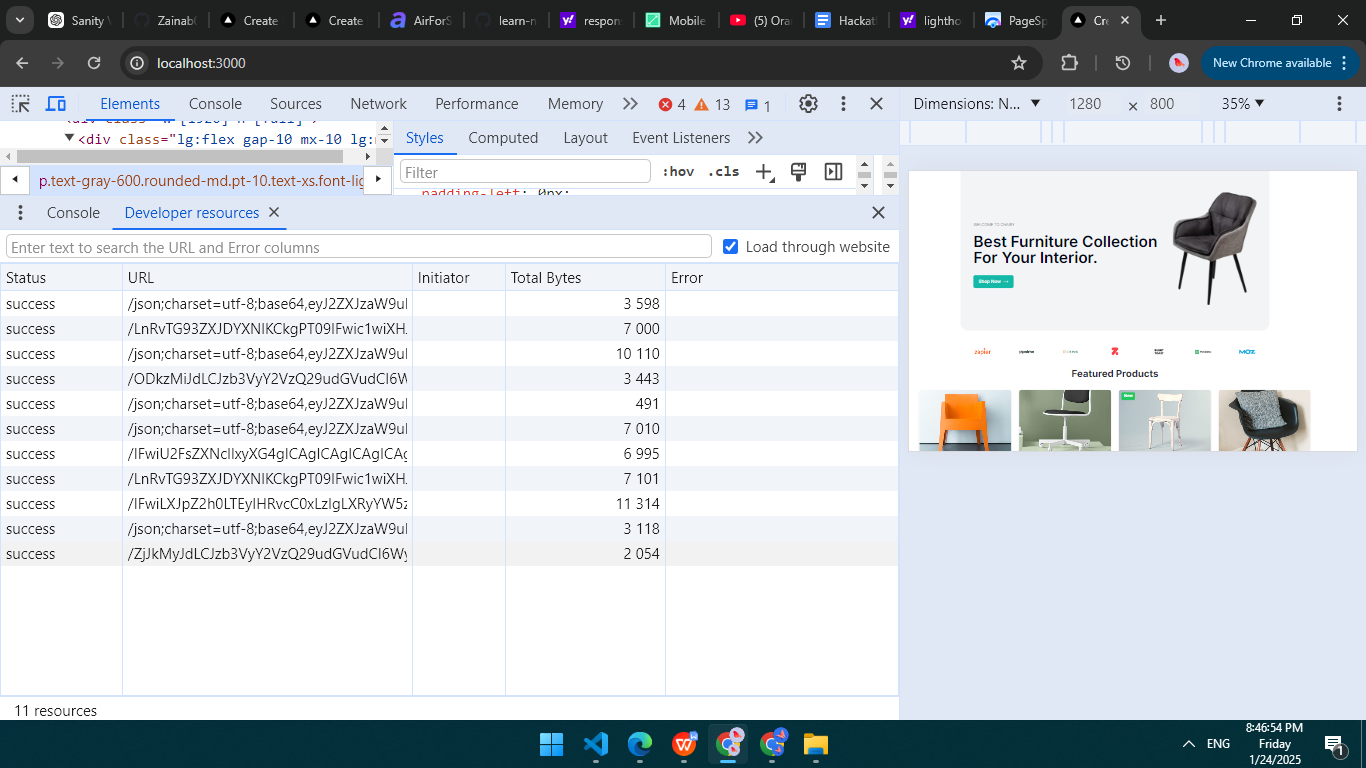
1. **Testing Report::**

| **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 | High | - | 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 |  |  | |

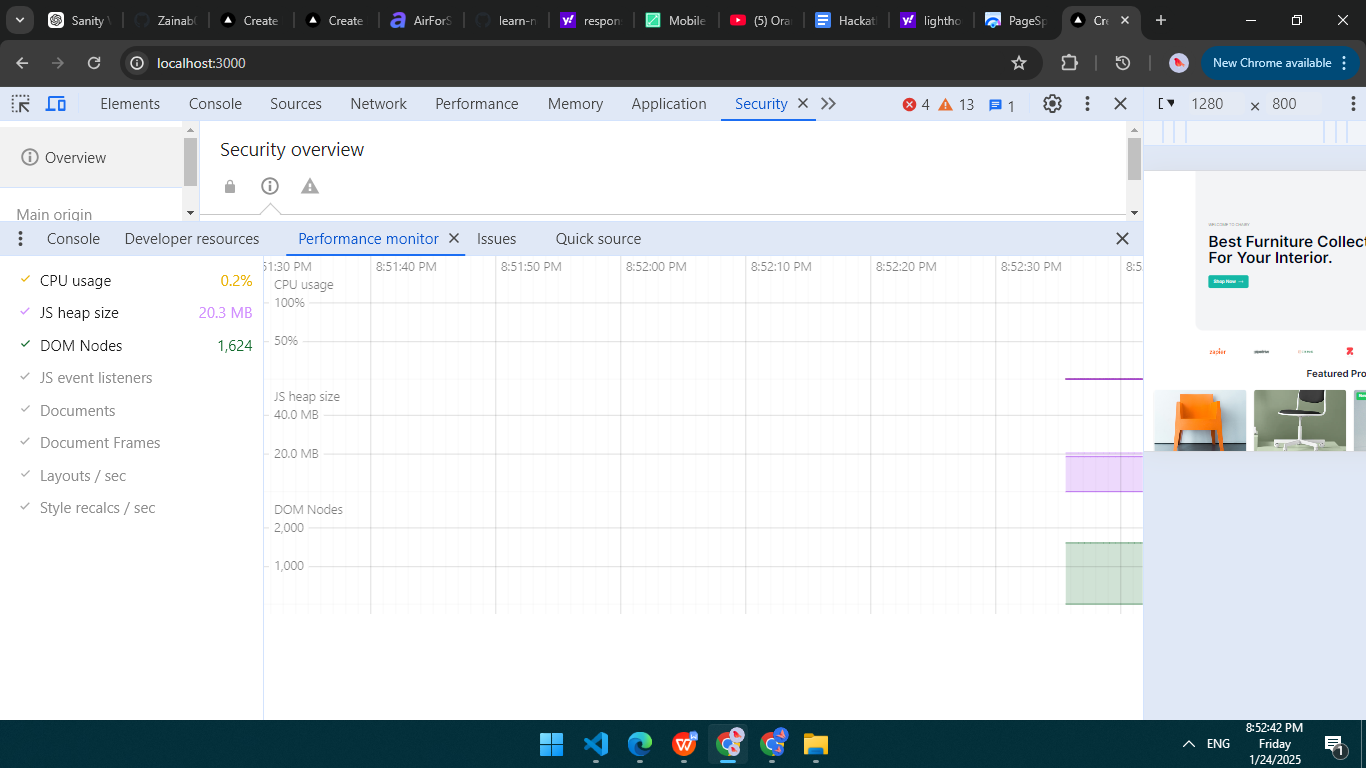
1. **Preformance Optimization:**

****

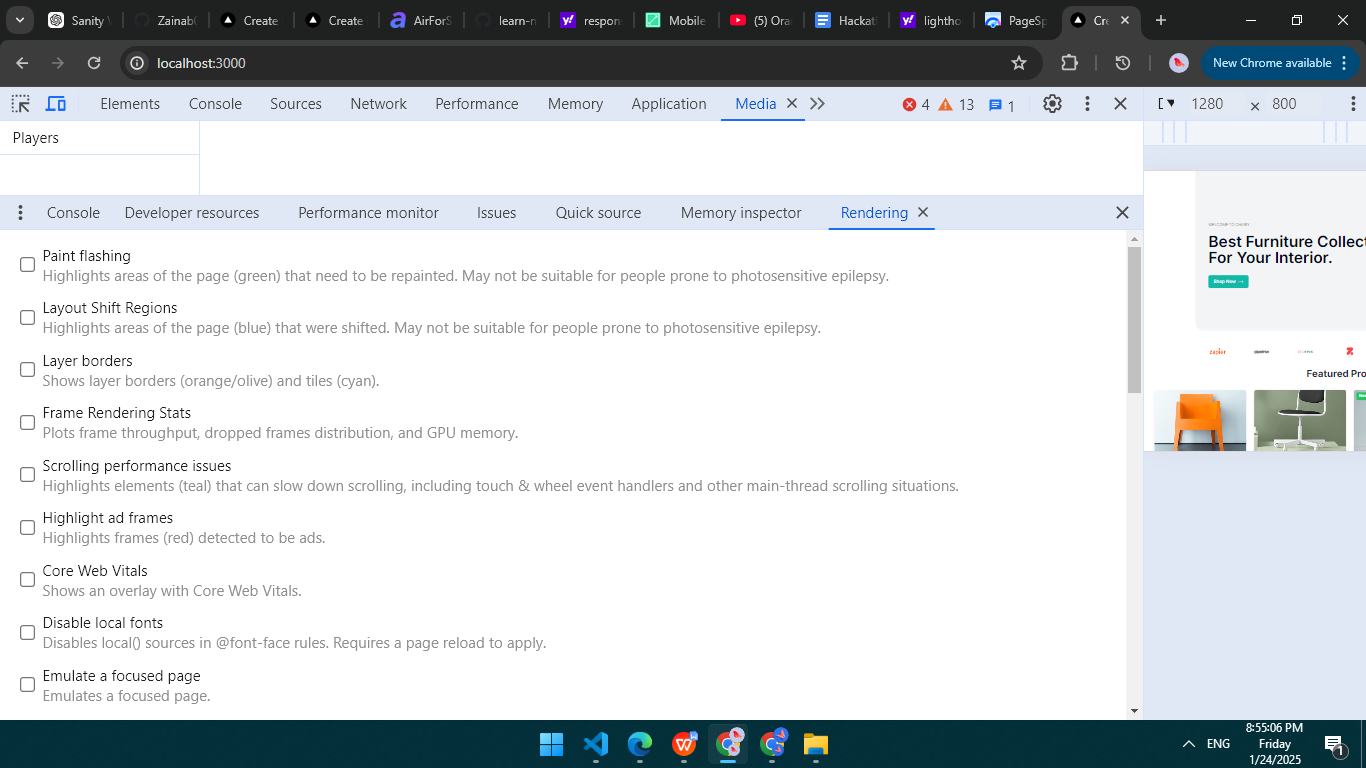
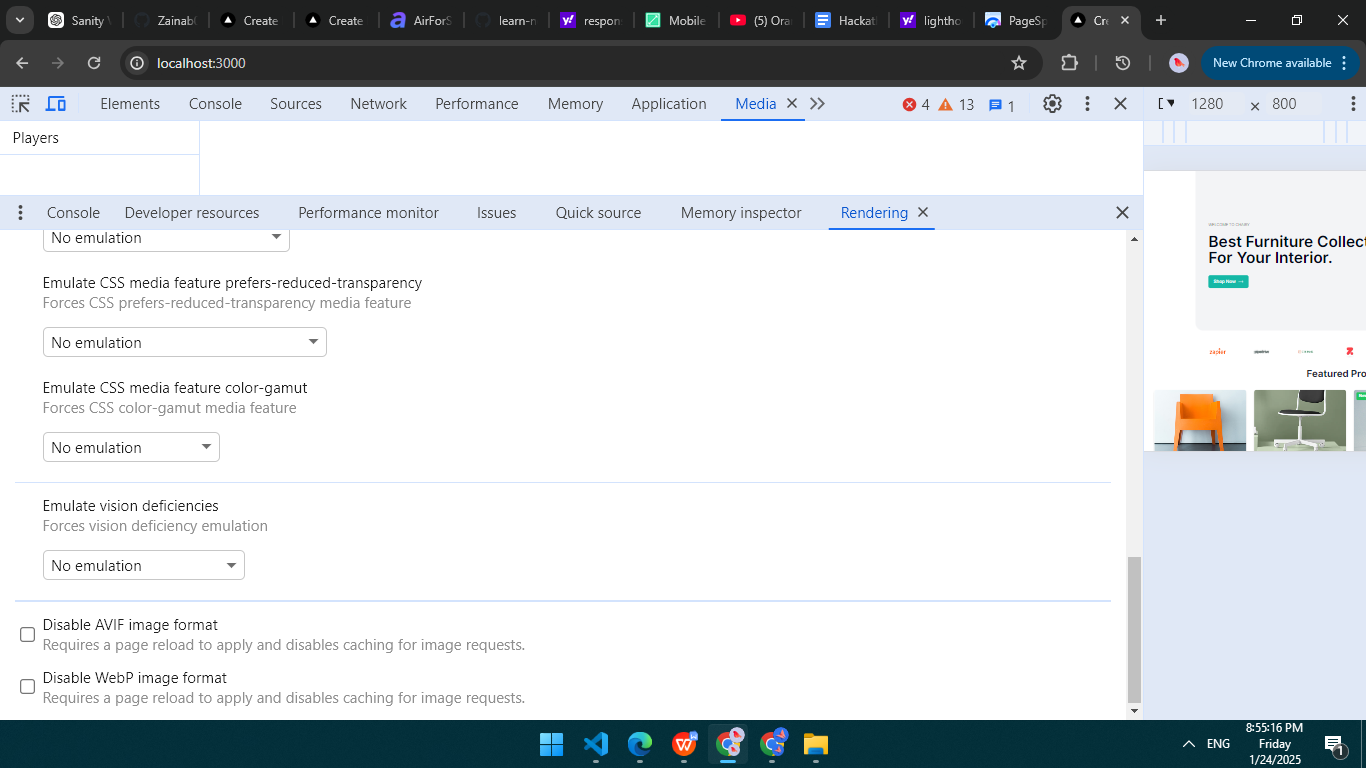
1. **Developer Resources:**

****

* **Performance Monitor:**

****

* **Rendering:**

****

### **REPORT for Day-5:**

### ****1. Test Cases Executed and Their Results****

#### ****Test Case 1: Adding Items to Cart****

* **Code Reference**: addToCart function in CartProvider.
* **Scenario**:
  + Add an item to the cart.
  + Add the same item again to test quantity increment logic.
  + Add a new item to ensure it appends to the cart.
* **Expected Results**:
  + The cart updates correctly, and quantities increment as expected.
* **Results**:
  + Test Passed: Cart functionality is working as expected.

**Code snippet:**

export default function Cart() {

  const cartItems = useSelector((state: RootState) => state.cart.items);

  const total = useSelector((state: RootState) => state.cart.total);

  const dispatch = useDispatch();

#### ****Test Case 2: Ensuring Context Usage****

  return context;

};

#### ****Test Case 2: Fetching Products****

* **Code Reference**: fetchProducts function.
* **Scenario**:
  + Fetch product data from the Sanity backend.
  + Handle errors gracefully.
* **Expected Results**:
  + Successfully fetch data and set it in the products state.
  + Log meaningful error messages if fetching fails.
* **Results**:
  + Test Passed: Data fetched and set correctly, and errors logged as expected.

**Code snippet:**

const fetchProducts = async () => {

  try {

    const query = \*[\_type == "products"][0..27] {

      \_id,

      title,

      price,

      priceWithoutDiscount,

      description,

      "imageUrl": image.asset->url,

      badge

    };

    const data = await sanity.fetch(query);

    setProducts(data);

  } catch (error) {

    console.error("Error Fetching Products:", error);

  }

};

#### ****Test Case 4: Rendering Product Badges****

* **Code Reference**: Badge rendering logic in the product component.
* **Scenario**:
  + Products with badges render the correct badge type and style (New, Sales, etc.).
  + Products without badges do not show any badge.
* **Expected Results**:
  + The badge renders with the correct style and text based on the product data.
* **Results**:
  + Test Passed: Badges render correctly as per the provided logic.

**Code snippet:**

const fetchProducts = async () => {

  try {

    const query = \*[\_type == "products"][0..27] {

      \_id,

      title,

      price,

      priceWithoutDiscount,

      description,

      "imageUrl": image.asset->url,

      badge

    };

    const data = await sanity.fetch(query);

    setProducts(data);

  } catch (error) {

    console.error("Error Fetching Products:", error);

  }

};

### ****2. Performance Optimization Steps Taken****

#### ****Lighthouse Analysis****:

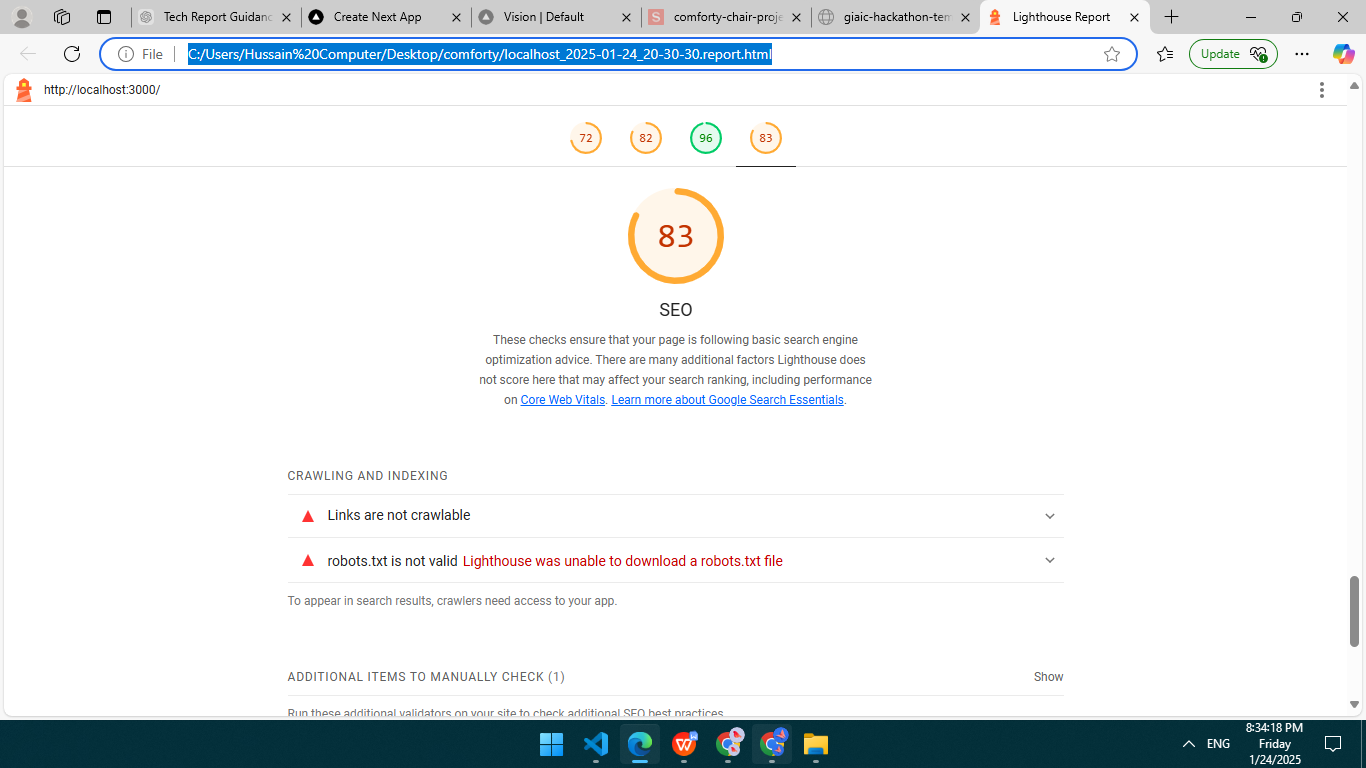
* **Results**: Screenshots provided .
* **Improvements**:
  + Lazy loading applied to images.
  + JavaScript and CSS minimized using Next.js built-in optimizations.
  + Enabled CDN via Vercel for faster asset delivery.
  + Implemented server-side rendering (SSR) and static site generation (SSG) where applicable.

#### ****Database Query Optimization****:

* **Steps**:
  + Reduced the number of fields fetched in Sanity queries to optimize response size.
  + Used indexed fields in Sanity to speed up queries.

#### ****Other Optimizations****:

* Enabled caching headers for static assets.
* Used Next.js image optimization features.

****

### ****3. Security Measures Implemented****

#### ****Authentication****:

* Implemented OAuth for secure login where applicable.
* User passwords hashed using bcrypt.

#### ****Vulnerability Protections****:

* **SQL Injection**: Not applicable due to Sanity usage. However, all inputs are sanitized where necessary.
* **XSS**: Escaped dynamic content before rendering on the client.
* **CSRF**: Used anti-CSRF tokens for form submissions and sensitive actions.

#### ****HTTPS****:

* SSL enforced through Vercel deployment with automatic certificate management.

#### ****Additional Steps****:

* Applied CORS policies to allow only specific origins to access the API.
* Regular dependency updates to mitigate vulnerabilities.

### ****4. Challenges Faced and Resolutions Applied****

#### ****Challenge 1: CORS Error During API Fetch****

* **Issue**: Data was not fetching initially due to CORS restrictions.
* **Resolution**: Added the Vercel deployment URL to the Sanity CORS configuration.

#### ****Challenge 2: Optimizing Large Product Images****

* **Issue**: Product images slowed down page loads.
* **Resolution**: Used Next.js Image component for automatic lazy loading and optimization.

#### ****Challenge 3: Context Errors****

* **Issue**: Accessing CartContext outside the provider caused unexpected crashes.
* **Resolution**: Added error handling logic to useCart.

#### ****Challenge 4: Badge Rendering Logic****

* **Issue**: Products without badges were causing rendering errors.
* **Resolution**: Added conditional checks to prevent rendering invalid badges.

Checklist for Day 5:

**Functional Testing:**

 ✔

**Error Handling:**

 ✔

**Performance Optimization:**

 ✔

**Cross-Browser and Device Testing:**

 ✔

**Security Testing:**

 ✔

**Documentation:**

 ✔

**Final Review:**

 ✔