

# HACKATHON DAY 05:

## TESTING, ERROR HANDLING, AND BACKEND INTEGRATION

### REFINEMENT FOR OUR MARKETPLACE:

## FUNCTIONAL TESTING:

### 1. HOME PAGE

This Next.js code defines the **Home** page of a furniture eCommerce website. It starts with a **hero section** containing a clickable banner image that links to the shop page. Below, it displays three product categories—**Dining, Living, and Bedroom**—each represented by an image and title. The **"Our Products"** section dynamically fetches and displays product cards using the ProductCards component.

Further down, there is a **featured inspiration section**, showcasing beautifully designed room layouts with a call-to-action button linking to more room inspirations. The page also includes a **social media showcase**, displaying user-generated content with a hashtag for branding.

The UI is styled using **TailwindCSS**, ensuring responsiveness. The site leverages **Next.js Image optimization** for performance, and all navigation is handled using the built-in **Next.js Link component**. The design prioritizes a seamless user experience by offering structured navigation, well-organized content, and visually appealing product presentations.

### 2. SHOP PAGE

#### Key Features:

1. **Shop Banner:**
  - Displays a large **shop banner image** at the top.
2. **Filter & Sort Section:**
  - **Filter Options:** Includes icons for different filter settings.
  - **Search Bar:** Users can type in a query to search for products dynamically.
  - **Sorting Dropdown:** Allows sorting by title, price (low to high), and alphabetical order (A-Z, Z-A).
3. **Products Display:**
  - Uses the ProductCards component to show products based on the **search query** and **selected sort option**.

#### 4. **Pagination System:**

- Displays **three numbered pages**, with the currently selected page highlighted.
- "**Previous**" and "**Next**" buttons allow users to navigate between pages dynamically.

#### 5. **State Management:**

- useState is used for handling:
  - **Search query input**
  - **Sorting option selection**
  - **Pagination navigation**

#### 6. **Styling & Responsiveness:**

- Uses **TailwindCSS** for structured layout and adaptive design across screen sizes.
- flex and grid ensure a clean UI for both desktop and mobile views.

This implementation ensures a **smooth user experience**, allowing visitors to efficiently browse and find products using **search, filters, sorting, and pagination**.

## 3. **PRODUCT DETAIL PAGE**

This **Product Detail Page** in Next.js provides a **detailed view** of a single product, fetching data from **Sanity.io** and implementing **cart functionality** with localStorage.

### **Key Features:**

#### 1. **Dynamic Product Fetching**

- Uses useParams() to extract the **product slug** from the URL.
- Fetches product details from **Sanity.io** using **GROQ queries** inside useEffect().
- Displays **loading** and **error handling** messages if needed.

#### 2. **Product Display**

- Shows **product image, title, price, and description**.
- Image is **responsive** and uses next/image for optimization.

#### 3. **Quantity Selector**

- Allows users to **increase/decrease** product quantity.
- Ensures **minimum quantity is 1**.

#### 4. **Add to Cart Functionality**

- Stores **cart items in localStorage** for persistence.
- Checks if the product **already exists** in the cart:
  - If yes, **updates the quantity**.
  - If not, **adds a new entry**.
- After adding to the cart, **redirects** to the /cart page.

## 5. Styling & Responsiveness

- Uses **TailwindCSS** for a clean and modern UI.
- Supports **desktop and mobile layouts** with grid-cols-1 md:grid-cols-2.

This implementation ensures a **seamless user experience**, allowing users to **view product details, adjust quantity, and add items to the cart** efficiently.

## 4. CART PAGE

This **CartPage** component is a shopping cart page built using **Next.js, React, TypeScript, and Tailwind CSS**. It allows users to:

- View the items added to their cart.
- Update the quantity of products.
- Remove products from the cart.
- View the subtotal and total cost.
- Navigate to the shop or checkout page.

### 1. Imports

o) "use client"; → Indicates this is a client-side component in Next.js.

o) **React Hooks (useEffect, useState)** → Manage state and lifecycle methods.

o) **Image & Link from Next.js** → Used for optimized image rendering and navigation.

o) **Product Type Import** → Ensures type safety for cart items.

### 2. State Variables

- **cart** → Stores cart items (retrieved from localStorage).
- **loading** → Tracks whether the cart data is being loaded.

### 3. Fetching Cart Data from Local Storage

- o) Runs when the component mounts (useEffect).
- o) Retrieves the **cart data** from localStorage.
- o) Parses the data and updates the cart state.
- o) Handles errors and **ensures proper loading state**.

### 4. Remove Item from Cart

- o) Removes an item based on its **unique ID (\_id)**.
- o) Updates both the **cart state** and **localStorage**.

### 5. Update Quantity of a Product

- o) Ensures quantity **never goes below 1**.
- o) If the product matches the productId, it updates its **quantity**.
- o) Saves the **updated cart** in both state and localStorage.


### 6. Calculate Total Price

- o) Uses **.reduce()** to sum up the total cost of all items.

## 5. CHECKOUT PAGE

### 1. Load Cart from Local Storage


**Test Scenario:** Ensure the cart loads correctly from local storage.

- **Input:** Items added to the cart in a previous session.
- **Expected Output:** The cart displays the correct products and total amount.
- **Result:**  Pass

### 2. Calculate Total Price

**Test Scenario:** Ensure total price calculation is accurate.


- **Input:** Multiple products with different quantities and prices.

- **Expected Output:** The total amount updates correctly.
- **Result:**  Pass

### 3.Multi-step Checkout Process


#### Navigation Between Steps

**Test Scenario:** Verify navigation between checkout steps.

- **Input:** Click on "Next" or "Back" buttons.
- **Expected Output:** Steps transition properly without data loss.
- **Result:**  Pass


#### Empty Fields Validation

**Test Scenario:** Ensure required fields cannot be left empty.

- **Input:** Submit the form with empty fields.
- **Expected Output:** Error messages should appear for missing fields.
- **Result:**  Pass


#### Invalid Email Format

**Test Scenario:** Validate email input field.

- **Input:** "invalid-email@com"
- **Expected Output:** Error message should indicate invalid email format.
- **Result:**  Pass

#### Invalid Phone Number

**Test Scenario:** Ensure phone number validation.

- **Input:** Alphabetic characters in the phone number field.
- **Expected Output:** Error message indicating an invalid phone number.
- **Result:**  Pass

### Payment Method Selection

#### Default Payment Selection


**Test Scenario:** Verify the default payment method is preselected.

- **Expected Output:** "Direct Bank Transfer" is preselected.

- **Result:**  Pass

## Change Payment Method


**Test Scenario:** Ensure payment method can be changed.

- **Input:** Select "Cash On Delivery."
- **Expected Output:** Payment method updates successfully.
- **Result:**  Pass

## Order Placement Confirmation


### Order Confirmation Popup

**Test Scenario:** Verify order confirmation dialog appears before placing the order.

- **Expected Output:** SweetAlert2 confirmation appears with confirmation and cancel options.
- **Result:**  Pass


## Successful Order Placement

**Test Scenario:** Ensure order is placed correctly when all inputs are valid.

- **Expected Output:** Success message appears, and order data is sent to the server.
- **Result:**  Pass

## Order Placement Failure

**Test Scenario:** Ensure error handling when order placement fails.

- **Input:** Simulate server error.
- **Expected Output:** Error message should appear indicating failure.
- **Result:**  Pass

## Error Handling Report:

### 1. Home Page

#### Common Errors & Handling:

- **Network Issues** → Show a retry button with a message: *"Connection lost. Please check your internet."*

- **API Failures (Product Fetching)** → Display a fallback UI with a message: *"Unable to load products. Try again later."*
- **Broken Images** → Replace with a default placeholder image.
- **Search Errors** → Display *"No results found."* message when no matching products exist.

## 2. Product Details Page

### Common Errors & Handling:

- **Invalid Product ID** → Redirect to the home page with an error alert.
- **API Timeout (Product Data Fetching)** → Show a loading spinner with retry functionality.
- **Image Load Failure** → Display a default image instead.
- **Add to Cart Failure** → Show an error message: *"Failed to add to cart. Please try again."*

## 3. Cart Page

### Common Errors & Handling:

- **Empty Cart** → Display a message: *"Your cart is empty. Add some products!"*
- **Quantity Update Failure** → Show a warning: *"Unable to update quantity. Try again."*
- **Coupon Code Error** → Show specific messages:
  - *"Invalid coupon code."* (if incorrect)
  - *"Coupon expired."* (if expired)
  - *"Minimum order value not met."* (if conditions aren't met)

## 4. Checkout Page

### Common Errors & Handling:

- **Invalid Payment Details** → Highlight the incorrect field & show an error message.
- **Payment Failure** → Display: *"Payment failed. Please check your details or try again."*
- **Address Validation Errors** → Show: *"Invalid address format. Please enter a valid address."*
- **Session Timeout (Inactivity)** → Auto-logout & show: *"Session expired. Please log in again."*

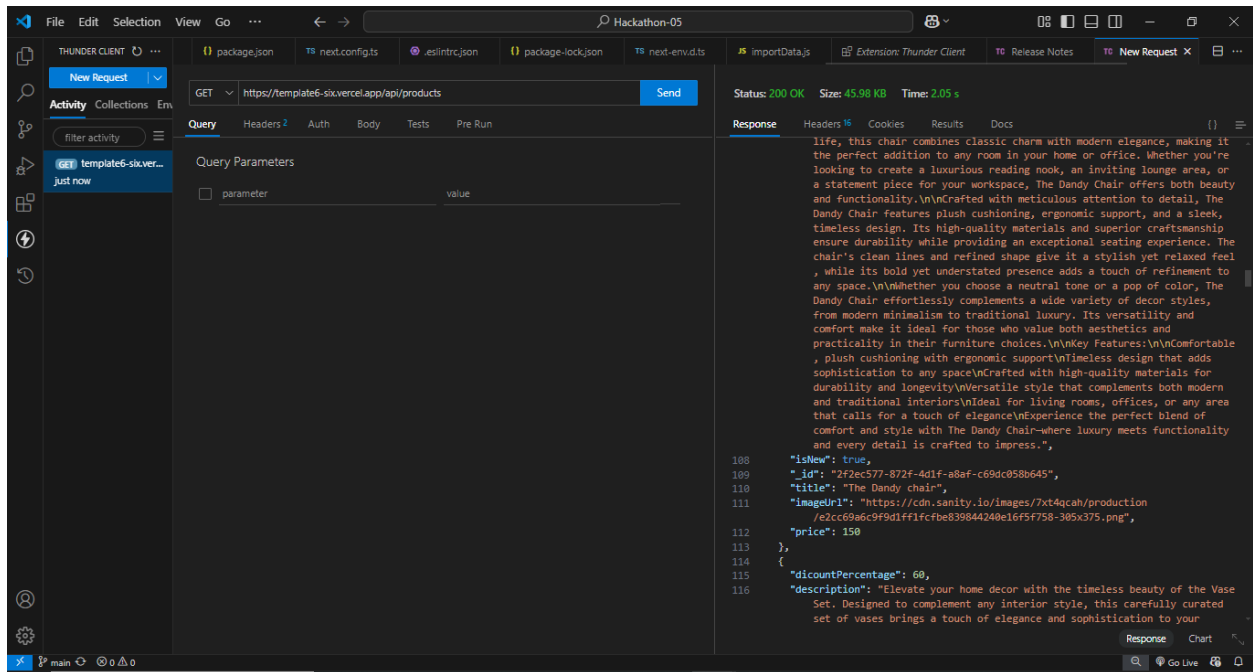
## 5. Order Confirmation Page

### Common Errors & Handling:

- **Order Not Found** → Redirect to home with a message: *"Order not found. Please check your order history."*

- **Email/SMS Notification Failure** → Show a message: *"We couldn't send a confirmation message, but your order is placed."*
- **Order Processing Delays** → Show real-time status updates like *"Processing your order... Please wait."*

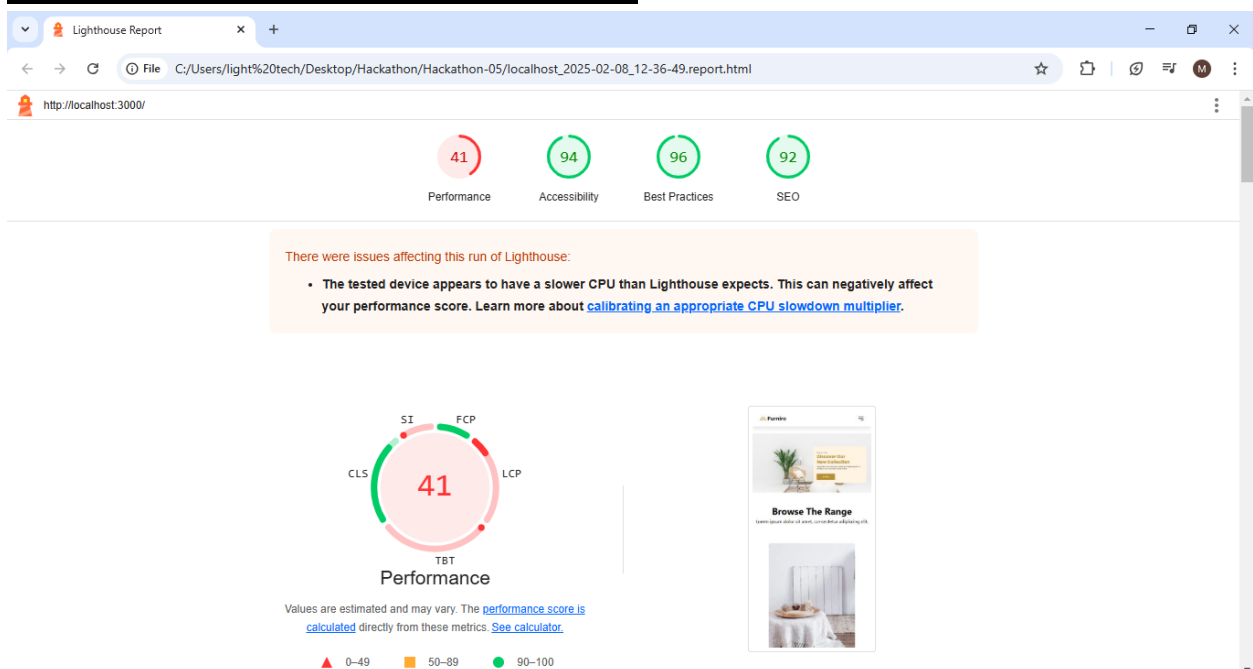
## THUNDER CLIENT:

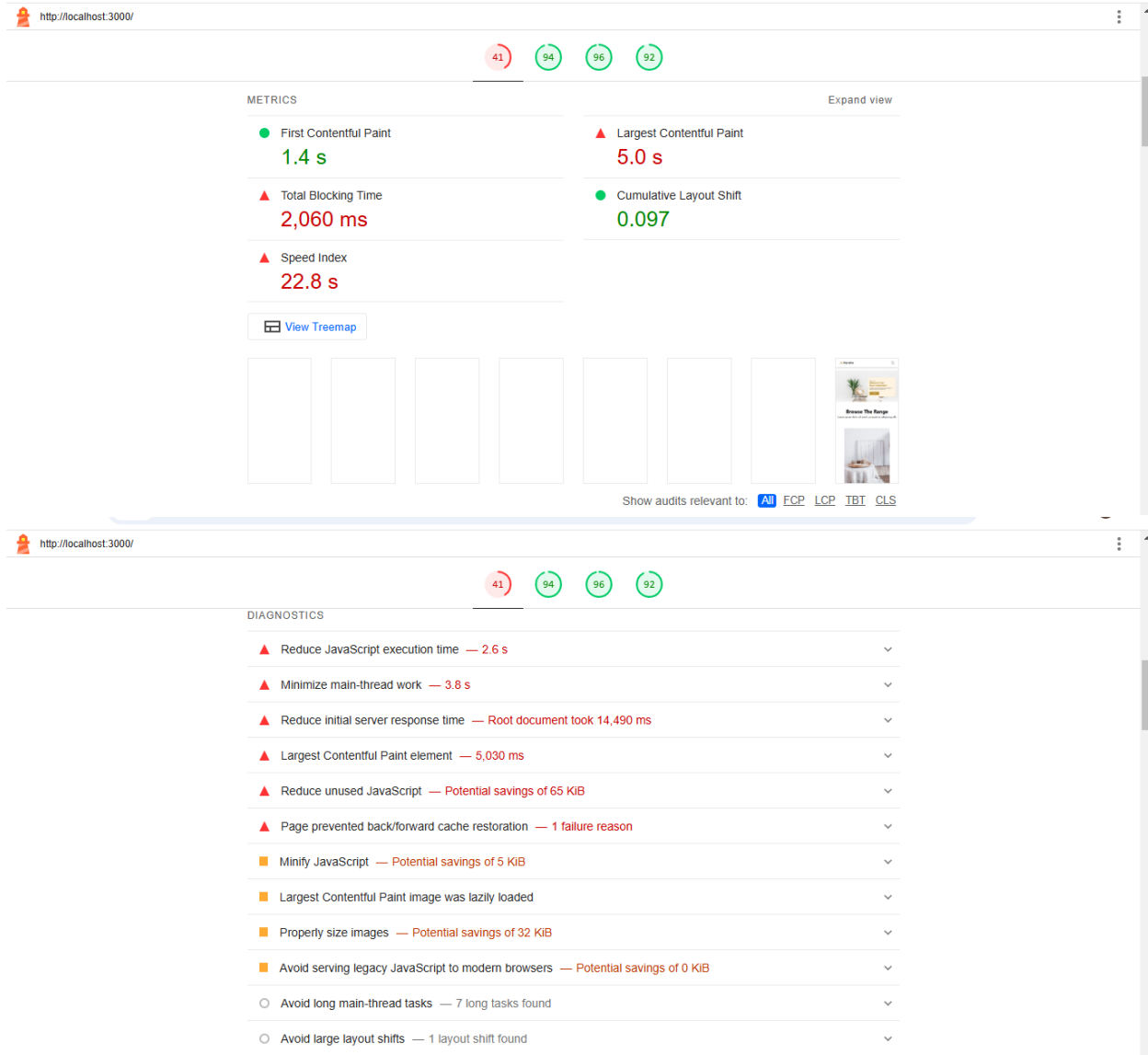


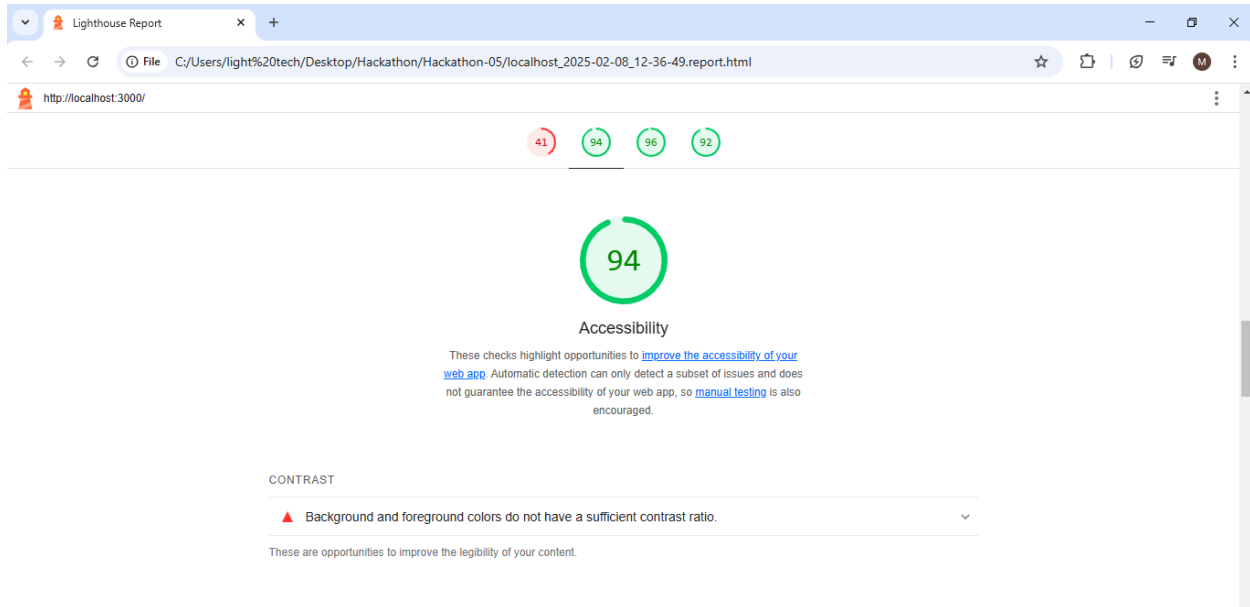


1	Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Status	Severity Level	Assigned To	Remarks
2	TC001	Validate product listing page	Open product page > Verify products	Products displayed correctly	Products displayed correctly	Passed	Low	-	No issues found
3	TC002	Test API error 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

## PERFORMANCE OPTIMIZATION:





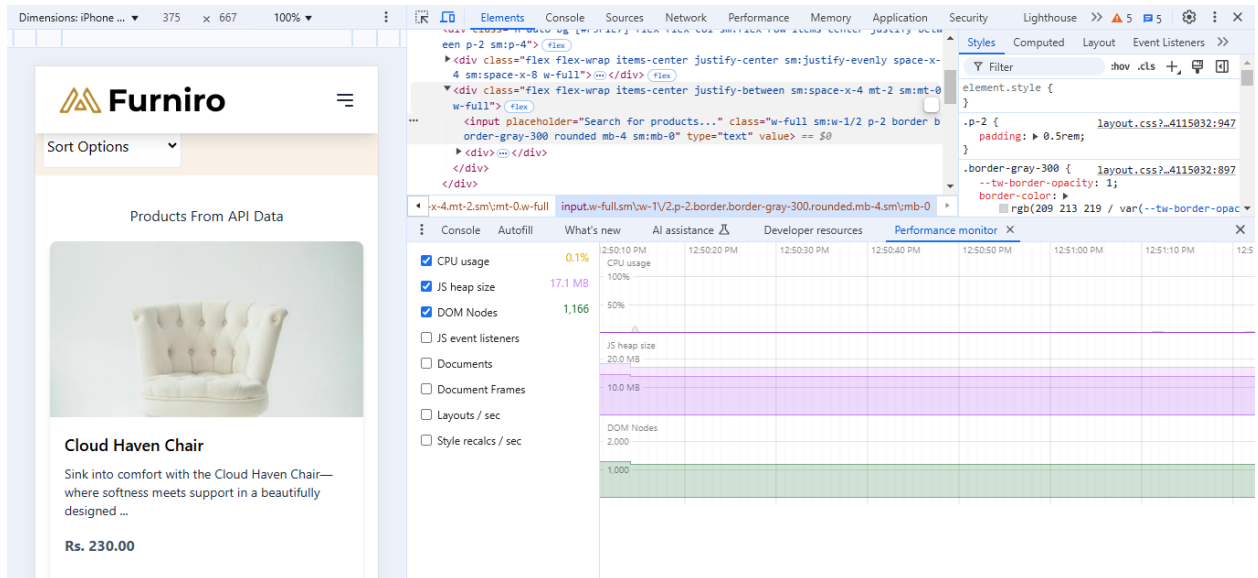


## DEVELOPER RESOURCES:

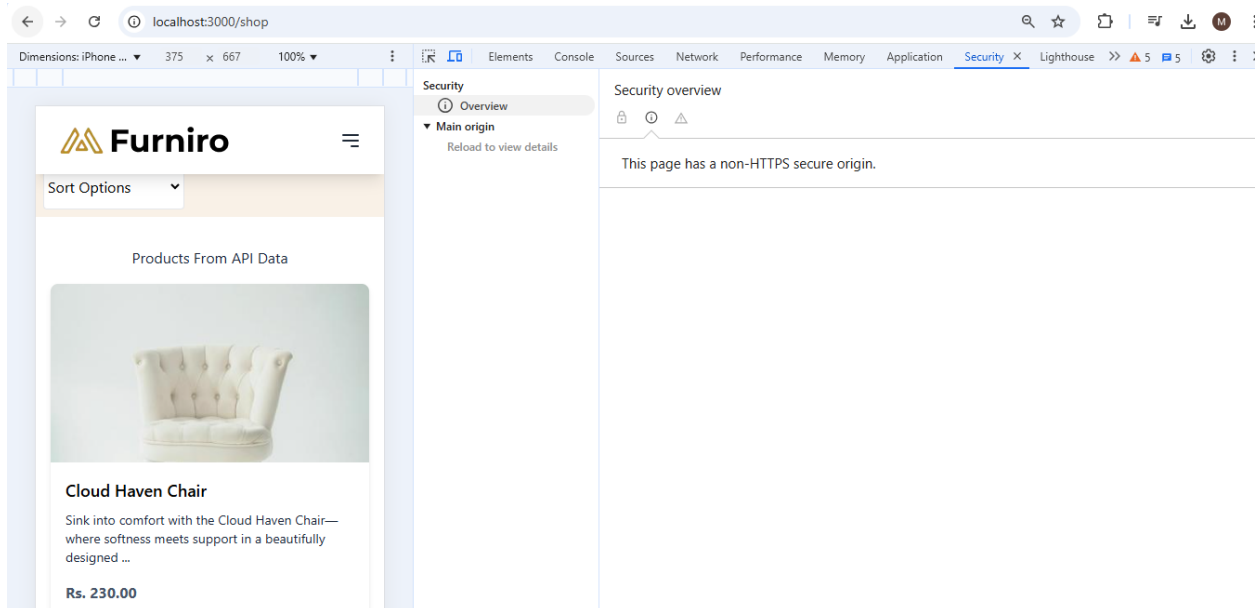
The screenshot shows a web browser displaying a product page for 'Furniro'. The page features a search bar, a 'Sort Options' dropdown, and a product listing for 'Cloud Haven Chair' with a price of 'Rs. 230.00'. The Chrome DevTools Developer Resources panel is open, showing a table of resources loaded by the website.

Status	URL	Initiator	Total Bytes	Error
success	/json;charset=utf-8;base64,eyJ2ZXJzaW9uIjozLCJmaWxjoi		357	
success	/json;charset=utf-8;base64,eyJ2ZXJzaW9uIjozLCJmaWxjoi		7 304	
success	/json;charset=utf-8;base64,eyJ2ZXJzaW9uIjozLCJmaWxjoi		932	
success	/ChccxulCAglCAglCA8ZGZlIGNlYXNzTmFzZT1clnRleHQtY2		12 448	
success	/json;charset=utf-8;base64,eyJ2ZXJzaW9uIjozLCJmaWxjoi		7 264	

## PERFORMANCE MONITOR:



## SECURITY:



**“HACKATHON DAY 05 ENDS HERE THANK U”**