Prepared by: Erum Yousuf

DAY 5 - TESTING, ERROR HANDLING, AND BACKEND INTEGRATION REFINEMENT

Published Date: **January 25, 2025**

Name of the Marketplace : **www.furnitureatdoorstep.com (not confirmed)**

Objective:

On Day 5, we will concentrate on testing activities, error handling, backend integration refinements, and optimizations carried out during Day 5 of the development process. The aim is to ensure that all features of the marketplace application function as expected, are secure, perform optimally, and are user-friendly across multiple devices and browsers.

Key Learning Outcomes:

1. Build dynamic frontend components to display data from Sanity CMS or APIs.
2. Implement reusable and modular component
3. Understand and apply state management techniques.
4. Learn the importance of responsive design and UX/UI best practices.
5. Prepare for real-world client projects by replicating professional workflow

File Attached:

- Test Cases (CSV Report):

Areas of Focus:

1. Functional Testing

Objective:

The core functionality of the marketplace application was tested to ensure all features such as product listing, cart operations, and checkout worked as expected.

Test Cases of Core Features:

Test Case for User Authentication

1. **Test Case ID**: TC001
2. **Test Case Description**: Test login and signup functionality.
3. **Pre-Conditions**: Ensure a user account exists for testing.
4. **Steps**:
   - Attempt login with correct credentials.
   - Attempt login with incorrect credentials.
   - Test the signup process with valid and invalid data.
5. **Expected Result:**
   - Login succeeds with correct credentials and fails with incorrect ones.
   - Signup succeeds with valid data.
6. **Actual Result**: Authentication works as intended.

7. **Severity Level**: High
8. **Assigned To**: [Assign name]
9. **Remarks**: Authentication is secure and functional.

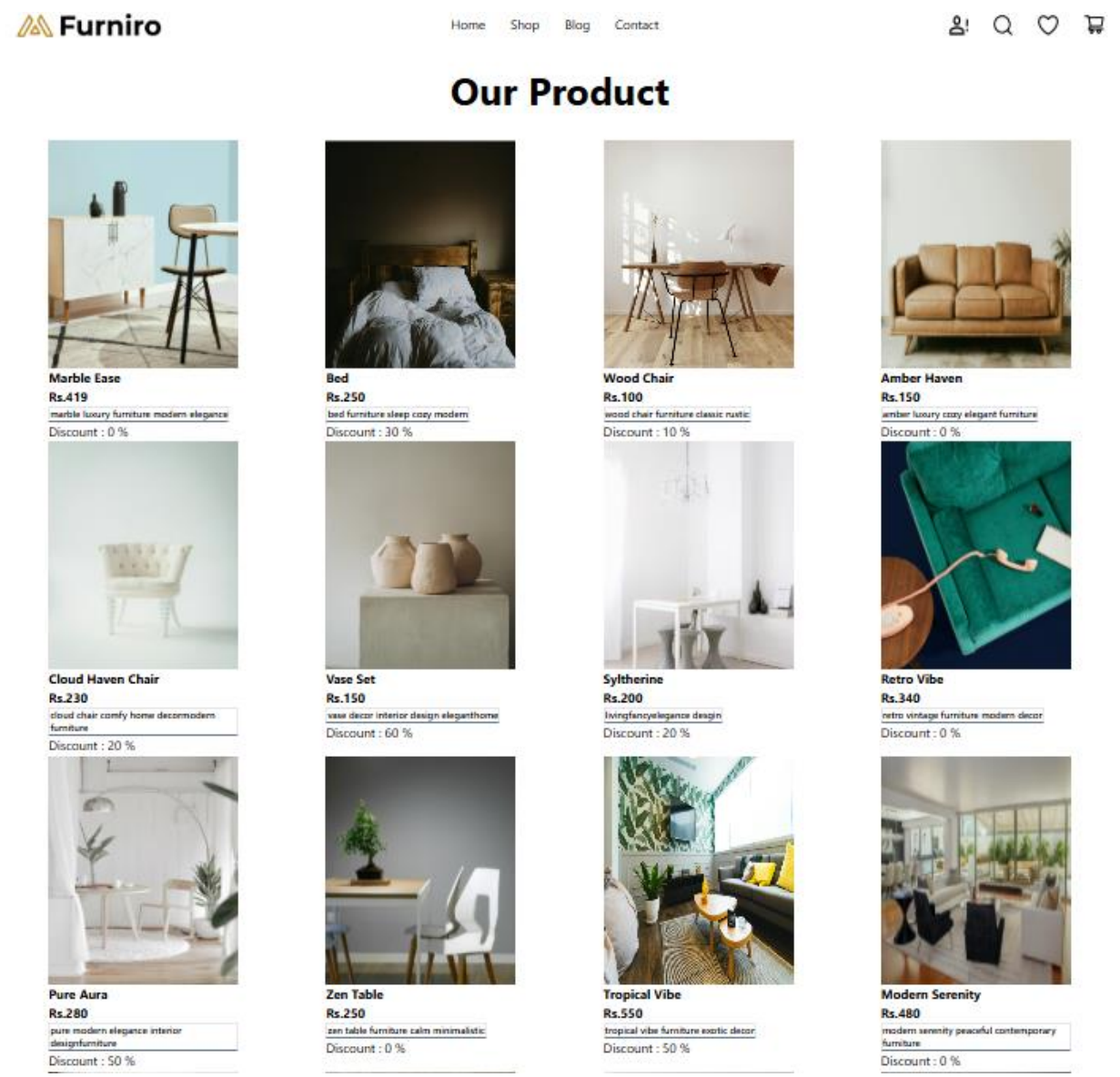
Test Case for Product Listing

1. **Test Case ID**: TC002
2. **Test Case Description**: Validate the product listing page.
3. **Pre-Conditions**: Ensure the database is populated with sample products.
4. **Steps:**
   - Open the products page.
   - Verify that products are displayed with correct titles, images, prices, and availability.
   - Check pagination if applicable.

5 **Expected Result**: Products display correctly with all relevant details.

6. **Actual Result**: Products displayed as expected.
7. **Severity Level**: Low
8. **Assigned To**: [Assign name]
9. **Remarks**: No issues detected.

Product Listing Snippet:

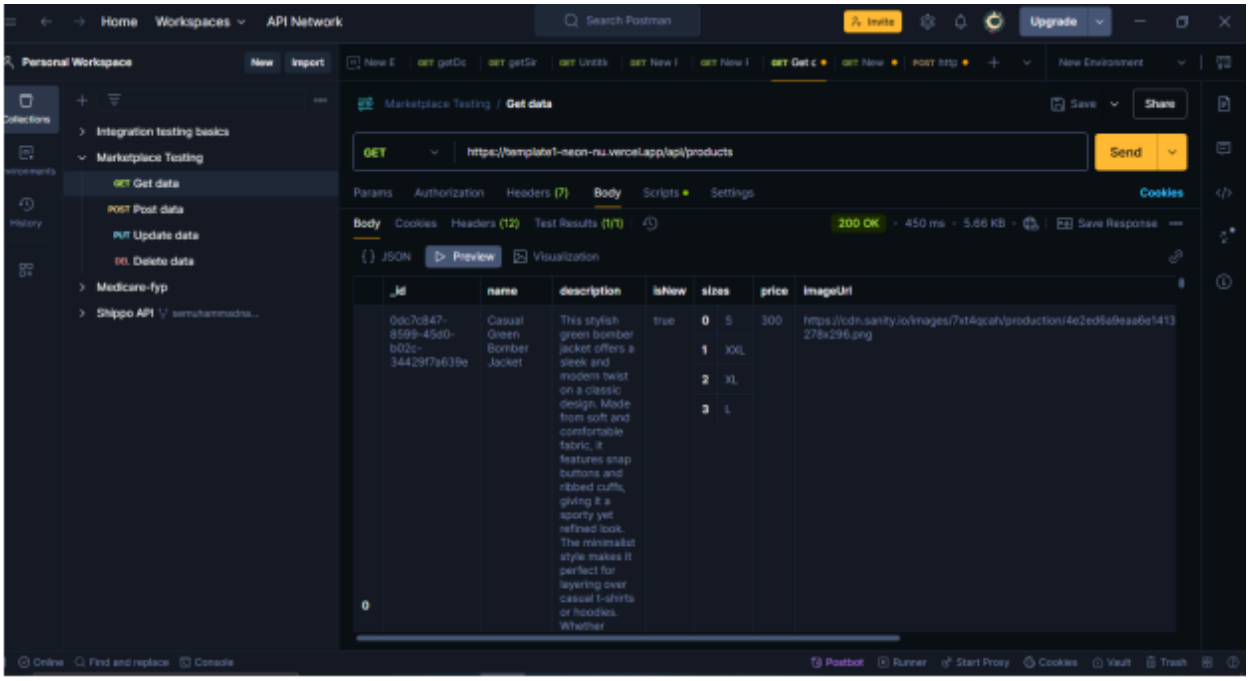


Test Case for API Error Handling

1. **Test Case ID:** TC003
2. **Test Case Description:** Validate API error handling for product data fetch.

3. **Pre-Conditions:** Simulate API unavailability or timeout.
4. **Steps:**
    - Disconnect the API or simulate failure.
    - Refresh the products page.
5. **Expected Result:** An error notification is displayed on the UI.
6. **Actual Result:** Displays appropriate error notification
7. **Severity Level:** Medium
8. **Assigned To:** [Assign name]
9. **Remarks:** Handles errors gracefully.

API Response Snippet:

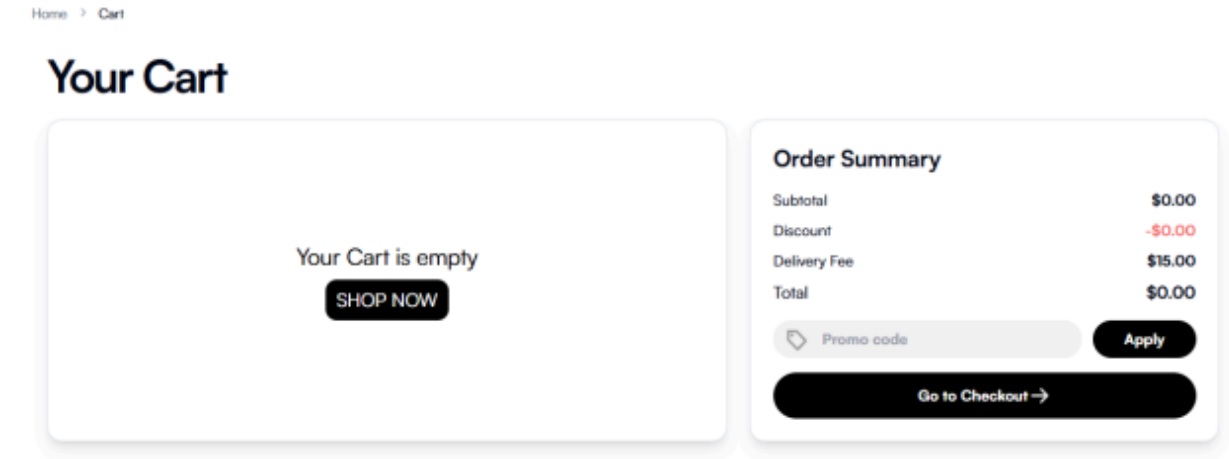


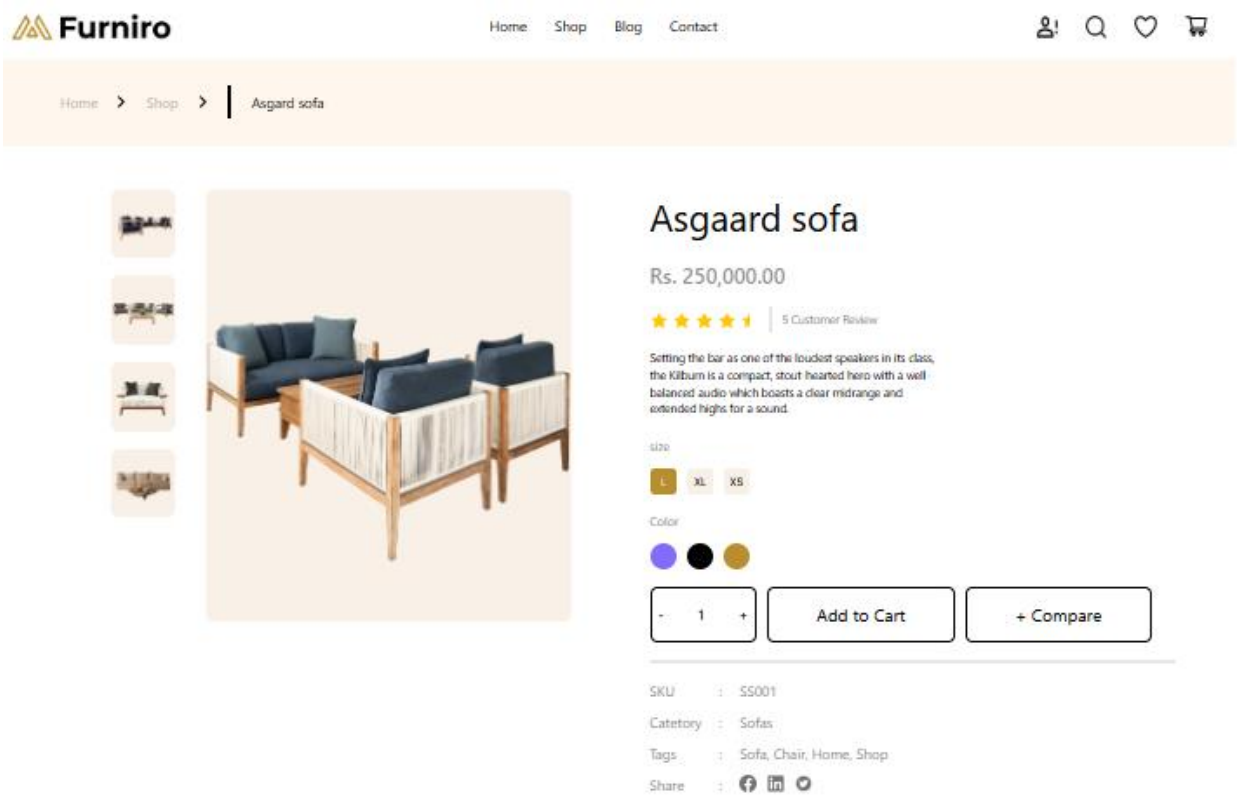
Test Case for Cart Functionality

1. **Test Case ID**: TC004
2. **Test Case Description:** Verify the functionality of adding items to the cart.
3. **Pre-Conditions:** Ensure products are listed and the cart is empty.
4. **Steps:**
    - Add a product to the cart.
    - Navigate to the cart page and verify the item is listed.
    - Update the quantity or remove the item from the cart.
5. **Expected Result:** Cart updates accurately based on user actions.
6. **Actual Result:** Cart functions as expected

7. **Severity Level:** Medium
8. **Assigned To:** [Assign name]
9. **Remarks:** Executed successfully. Cart

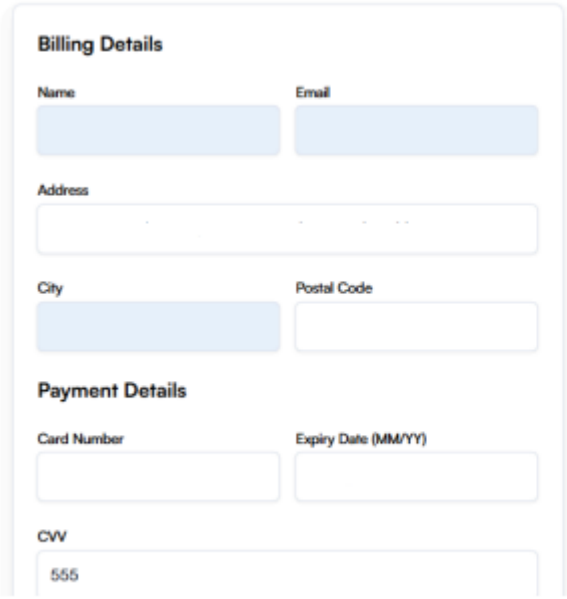
Functionality Snippets

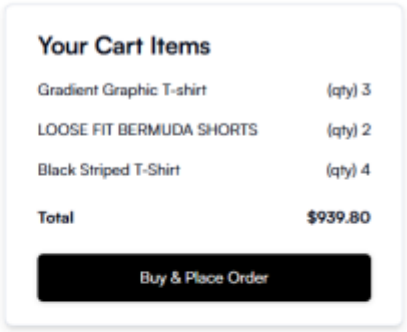


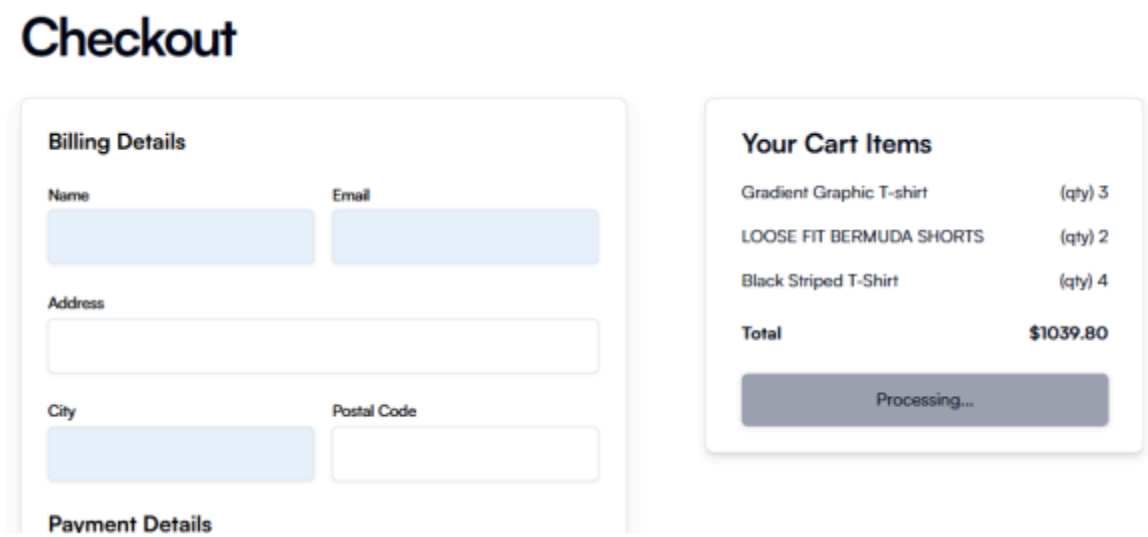


Test Case for Checkout

1. **Test Case ID**: TC005
2. **Test Case Description:** Validate the checkout process.
3. **Pre-Conditions:** Ensure at least one item is in the cart.
4. **Steps:**
    - Proceed to checkout.
    - Enter delivery and payment details.
    - Confirm the order
5. **Expected Result:** Order is successfully placed, and confirmation is displayed.
6. **Actual Result:** Order placement works as expected.
7. **Severity Level**: High
8. **Assigned To**: [Assign name]
9. **Remarks:** Checkout flow is functional.

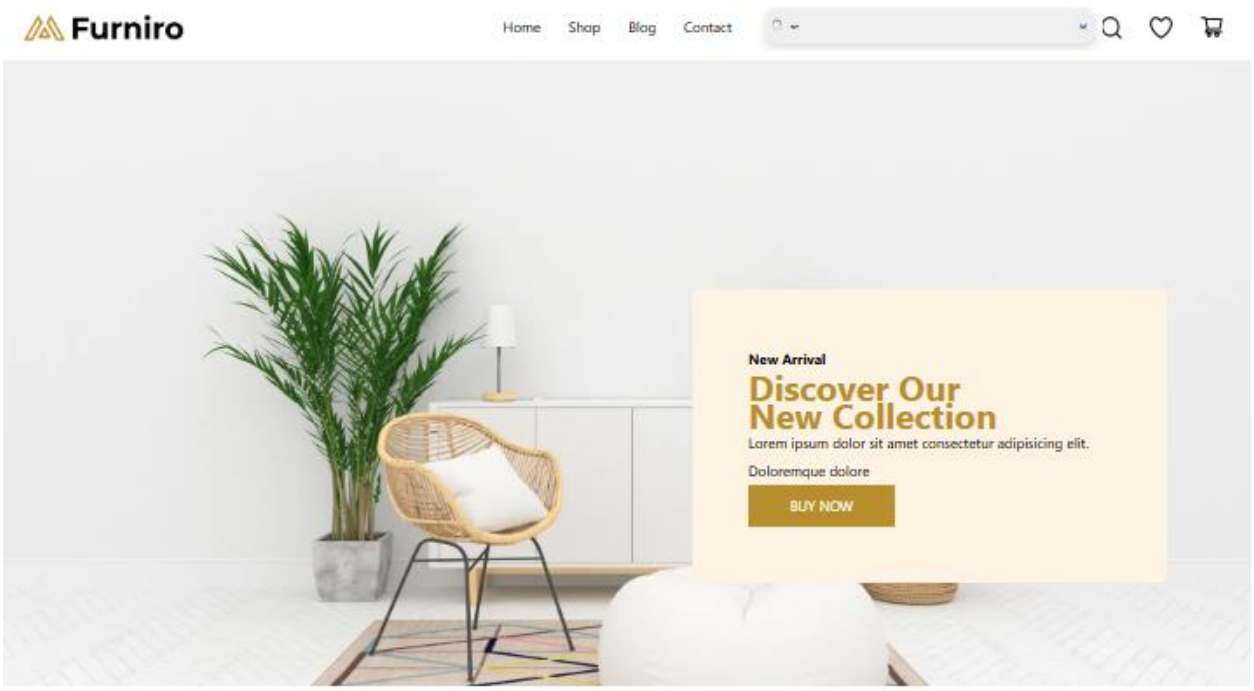






Test Case for Search Bar

1. **Test Case ID**: TC006
2. **Test Case Description:** Validate the functionality of the search bar.
3. **Pre-Conditions:** Products are available in the database.
4. **Steps:**
   - Enter a keyword in the search bar.
   - Verify if the correct products are listed.
   - Test for invalid searches or no results
5. **Expected Result:** Displays items matching the search criteria or shows a "No results found" message.
6. **Actual Result:** Search functionality is accurate.
7. **Severity Level:** High
8. **Assigned To**: [Assign name]
9. **Remarks:** No issues found



Test Case for Responsiveness

1. **Test Case ID:** TC007
2. **Test Case Description:** Verify responsiveness across devices.
3. **Pre-Conditions:** Use tools like Chrome DevTools for testing.
4. **Steps:**
   - Inspect the page.
   - Simulate different device viewports (mobile, tablet, desktop).
   - Verify UI elements adapt to screen sizes.
5. **Expected Result:** Content adjusts seamlessly for different screen sizes.
6. **Actual Result:** Fully responsive UI.
7. **Severity Level:** Medium
8. **Assigned To:** [Assign name]
9. **Remarks:** Fully responsive design confirmed.

Wood Chair
Rs.100
wood chair furniture classic rustic
Discount : 10 %

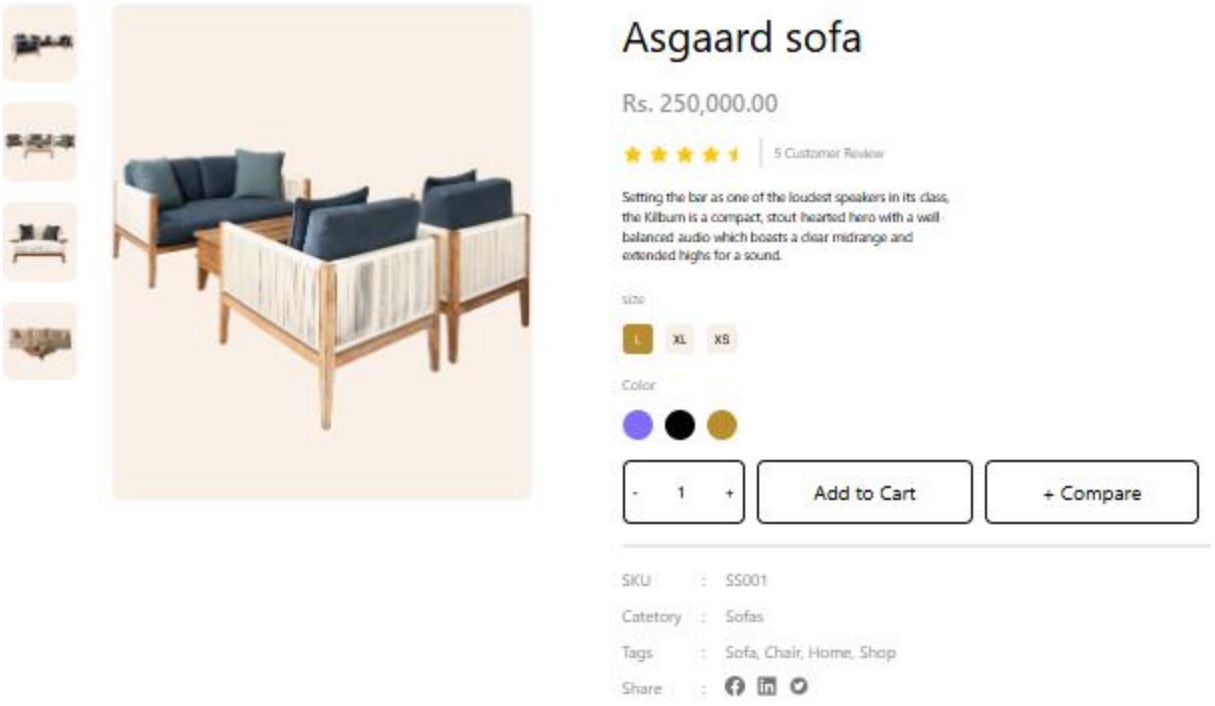
Amber Haven
Rs.150
amber luxury cozy elegant furniture
Discount : 0 %

Syltherine
Rs.200
livingfancyelegance design
Discount : 20 %

Retro Vibe
Rs.340
retro vintage furniture modern decor
Discount : 0 %

Test Case for Individual Product Details

1. **Test Case ID:** TC008
2. **Test Case Description:** Verify individual product detail pages.
3. **Pre-Conditions**: Products must have associated detail pages.
4. **Steps:**
   - Open the products page.
   - Click on a product to open its detail page.
   - Verify all details (e.g., image, description, price, stock).
5. **Expected Result:** Product detail page loads with accurate information.
6. **Actual Result**: Detail page works as expected.
7. **Severity Level**: Low
8. **Assigned To**: [Assign name]
9. **Remarks:** Works perfectly.

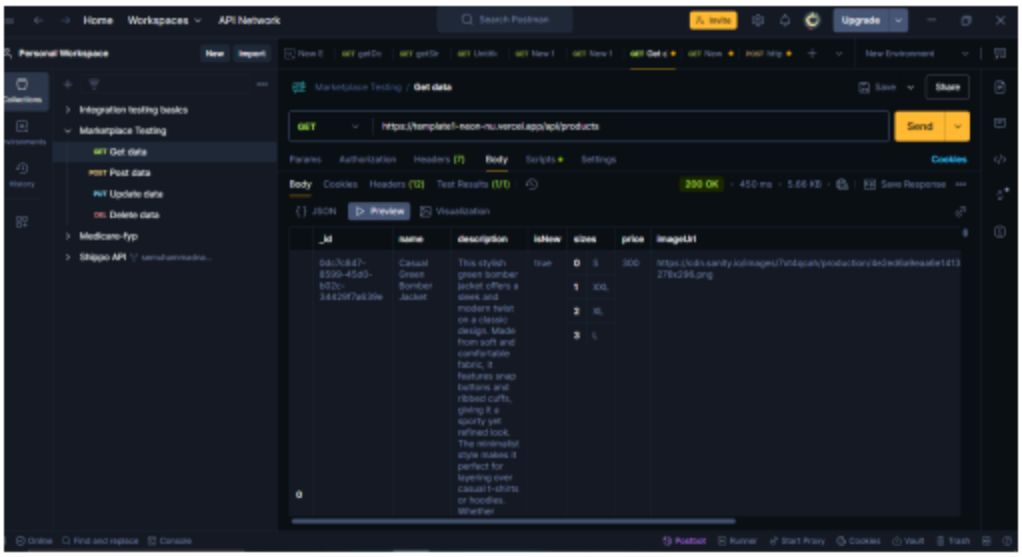


Tools Used:

- Postman: For API endpoint testing.
- React Testing Library: For unit testing of React components (will conduct later).
- Cypress: For end-to-end testing.

Test Coverage:

1. API End-Point (Postman)



2. Error Handling

Objective:

Ensure that proper error handling and fallback mechanisms are in place for API failures or unexpected errors.

1. Input Validation

In our marketplace project, we implemented robust input validation to ensure user-provided data is safe and correctly formatted:

- **Sanitization:**

  All input fields (e.g., product names, descriptions, user reviews) a sanitized to prevent SQL injection and Cross-Site Scripting (XSS) attacks.

- **Validation:**

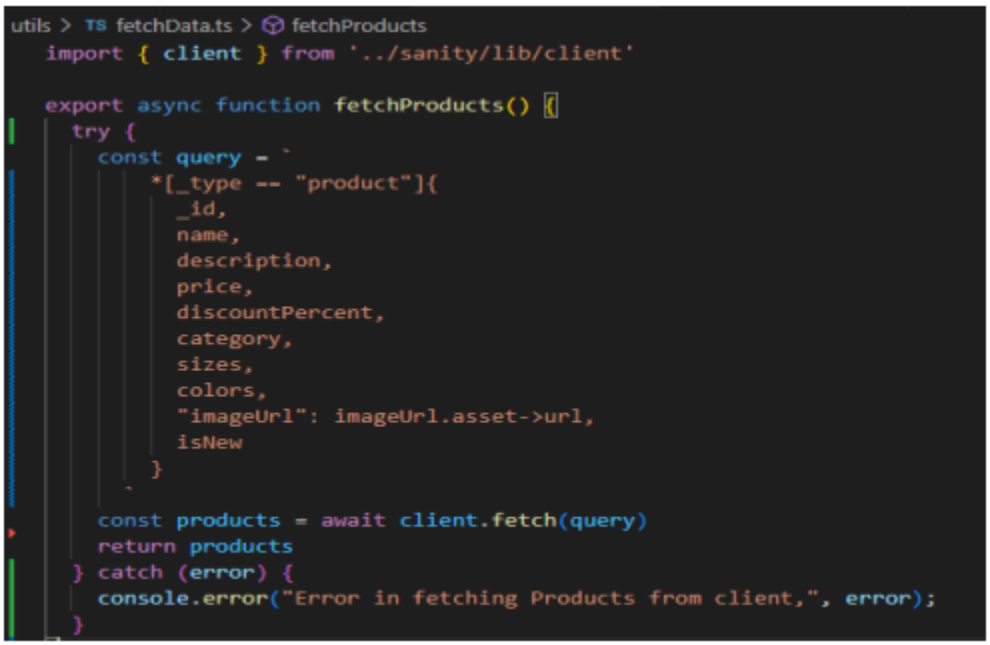
  Specific fields are validated as foll
  - Email: Ensured the format includes "@" and a valid domain.
  - Phone Numbers: Limited to numeric values and enforced specific count formats.
  - Product Prices: Restricted to numeric inputs with defined ranges prevent invalid data.

- User-Friendly Messages:

Errors like invalid input are communicated with clear and actionable messages. Example: If a user enters an invalid email during registration, they see: "Invalid Email Address"

- Hiding System Information:

Critical system details, such as stack traces, are not displayed to users in error messages to prevent exploitation of vulnerabilities



```
rc > sanity > TS env.ts > ...
1   export const apiVersion =
2     process.env.NEXT_PUBLIC_SANITY_API_VERSION ||
3
4   export const dataset = assertValue(
5     process.env.NEXT_PUBLIC_SANITY_DATASET,
6     'Missing environment variable: NEXT_PUBLIC_SA
7   )
8
9   export const projectId = assertValue(
10    process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
11    'Missing environment variable: NEXT_PUBLIC_SA
12  )
13
14  function assertValue<T>(v: T | undefined, error
15    if (v === undefined) {
16      throw new Error(errorMessage)
17    }
18
19    return v
20  }
```

3. Cross-Browser and Device Testing

1. Browser Testing

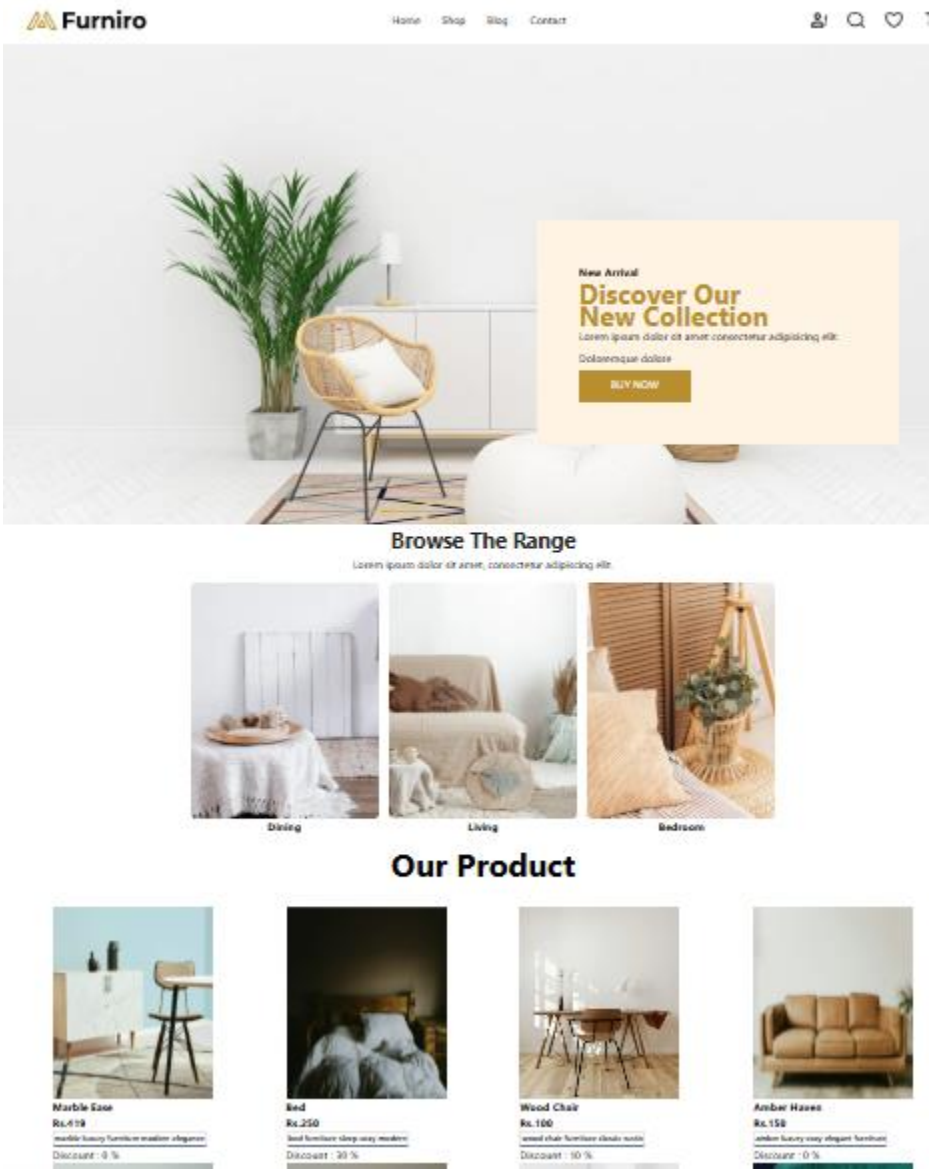
To ensure a seamless user experience, comprehensive browser testing is conducted across popular browsers to validate consistent rendering, navigation, and functionality. Key areas tested include:

- **Layout Alignment:**
  Ensuring that the design and positioning of elements are consistent across browsers.

- **Script Execution:**
  Validating that JavaScript runs smoothly and without errors.

- **User Interaction Responsiveness:**
  Testing buttons, forms, and interactive elements for consistent behavior.

CHROME:



2. Device Testing

Device compatibility is essential for ensuring a mobile-friendly, responsive experience for all users.

Approaches Used:

1. Responsive Design Tools:

- Tools like Chrome Dev Tools are used to simulate multiple devices, screen sizes, and resolutions, such as smartphones, tablets, and desktops.
- Simulate scenarios like slow network speeds and different operating systems for comprehensive testing.

2. Manual Testing on Physical Devices:

- A physical mobile device is tested to replicate real-world performance.
- Special attention is given to touch responsiveness, scrolling behavior, and hardware limitations (e.g., memory and CPU).

3. Performance and Optimization

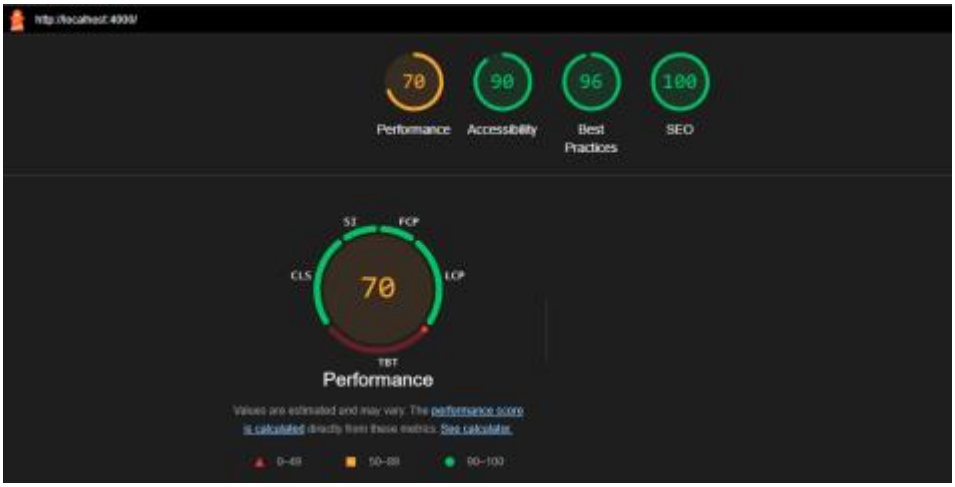
Performance Score: 70 (Needs Improvement) Key

Performance Metrics:

- First Contentful Paint (FCP): 0.4s (Excellent)
- Largest Contentful Paint (LCP): 0.6s (Excellent)
- Total Blocking Time (TBT): 2,580ms (Critical Issue)
- Cumulative Layout Shift (CLS): 0 (Perfect)
- Speed Index (SI): 0.7s (Excellent)

Key Findings:

- The performance score of 70 suggests that there are significant opportunities for optimization. Although critical render paths such as FCP and LCP are fast, TBT needs immediate attention, as high blocking time can negatively impact user experience.
- CLS is perfect, which indicates that layout shifts do not occur during page load, improving visual stability.
- Speed Index reflects a fast page load, but there are still several areas where performance could be improved.



Optimization Recommendations:

1. Reduce JavaScript Execution Time:

The page spends a significant amount of ti executing JavaScript (3.6s). Consider optimizing scripts or loading them asynchronously.

2. Minimize Main-Thread Work:

The main thread is congested with a total of 5.4s of processing time. Break long-running tasks into smaller, asynchronous operations.

3. Eliminate Render-Blocking Resources:

Investigate critical CSS/JS files that blo rendering and try deferring non-essential resources.

4. Minify JavaScript and CSS:

Consider minifying JavaScript and CSS to reduce resource size (especially JavaScript, where savings of 5 KiB are suggested).

5. Serve Images in Next-Gen Formats (WebP):

This could save up to 643 KiB, making image loading faster, especially on mobile devices.

6. Defer Offscreen Images:

Implement lazy loading for offscreen images to save up to 460 KiB of unnecessary data.

7. **Reduce Unused CSS and JavaScript:**
Minimize unused resources to save up to 10 KiB of CSS and 588 KiB of JavaScript.

Network Payloads:

- Avoid enormous network payloads; the current total size is 6,541 KiB, which could be reduced by optimizing images, JavaScript, and other assets.

Server Response Time

- The server response time is acceptable at 160ms, but further reductions could improve performance.

Accessibility Score: 90 (Good, but Needs Attention)

Key Accessibility Issues:

- Contrast: Background and foreground colors have insufficient contrast, affect legibility for users with visual impairments.
- Names and Labels: Links lack discernible names. Providing proper names will improve navigation for screen readers and other assistive technologies.
- Heading Elements: Some heading elements are not in sequential order, which can hinder keyboard navigation.

Recommendations for Accessibility:

1. Improve Contrast Ratios:

Adjust the background and text colors to meet WCAG accessibility standards.

2. **Add Descriptive Text for Links:**

Ensure that all links have a discernible name (via aria-label or title).

3. **Fix Heading Hierarchy:**

Reorganize headings to ensure they follow a sequential order, making it easier for screen reader users to navigate the content.

4. **Test for Keyboard Navigation:**

Manually check to ensure all interactive elements are focusable and accessible via the keyboard.

Best Practices Score: 96 (Excellent)

Key Areas:

- User Experience: Some images have incorrect aspect ratios. Fixing this can enhance the visual consistency of the page.
- Trust and Safety: Ensure that the Content Security Policy (CSP) is configured effective to protect against XSS attacks.
- General: Missing source maps for large first-party JavaScript files could be adde improve debugging and performance during development.

Recommendations:

1. Correct Image Aspect Ratios:

Ensure images maintain correct aspect ratios to improve the user experience

2. **Review and Strengthen CSP:**

Audit the CSP to ensure it's sufficiently restrictive prevent cross-site scripting (XSS) attacks.

3. **Add Source Maps for JavaScript:**

This will improve maintainability and debugging of JavaScript code.

SEO Score: 100 (Perfect)

Key SEO Best Practices:

- The site meets the essential SEO guidelines, including:
    - Properly configured meta title and meta description
    - Alt attributes are present for all image elements, ensuring search engines can properly index visual content.

- Structured data is valid, ensuring rich snippets can be displayed in search results.
- The robots.txt file is valid, and no content is being blocked from indexing

SEO Recommendations:

- Ensure that the structured data is consistently maintained across all pages for improved SEO and richer search result snippets.

Final Recommendations:

1. **Focus on TBT:**

The most significant performance bottleneck is Total Blocking Tim Immediate actions to reduce JavaScript execution time, defer tasks, and split large tasks into smaller ones should be prioritized.

2. **Optimize Images:**

Transitioning images to next-gen formats and deferring offscreen images will reduce payload and improve load times.

3. **Address Accessibility Issues:**

Make accessibility improvements to ensure a better experience for all users, including those with disabilities.

4. **SEO Optimization:**

While SEO is performing perfectly, maintaining structured data and ensuring crawlability will continue to yield better search rankings

4. User Acceptance Testing (UAT)

Objective:

Conduct User Acceptance Testing (UAT) to ensure the application is ready for the end-users.

Testing Scenarios:

1. **Search Functionality**
   Test Case: Verify users can search for products.
   Expected Results:
   - The search bar should return relevant product results based on user input.

2. **Add to Cart**
   Test Case: Verify users can add products to their shopping cart.
   Expected Results:
   - The product should be added to the cart with the correct quantity.

3. **Checkout Process Test Case:**
   Verify users can complete the checkout and payment process.

   Expected Results:
   - Users should be able to input shipping details and process payments without errors.

   Expected Results:
   - All user workflows are intuitive and functional
   - No issues were encountered during the simulated user sessions.

   Actual Results:
   - No critical issues were found. Users were able to complete all workflow successfully