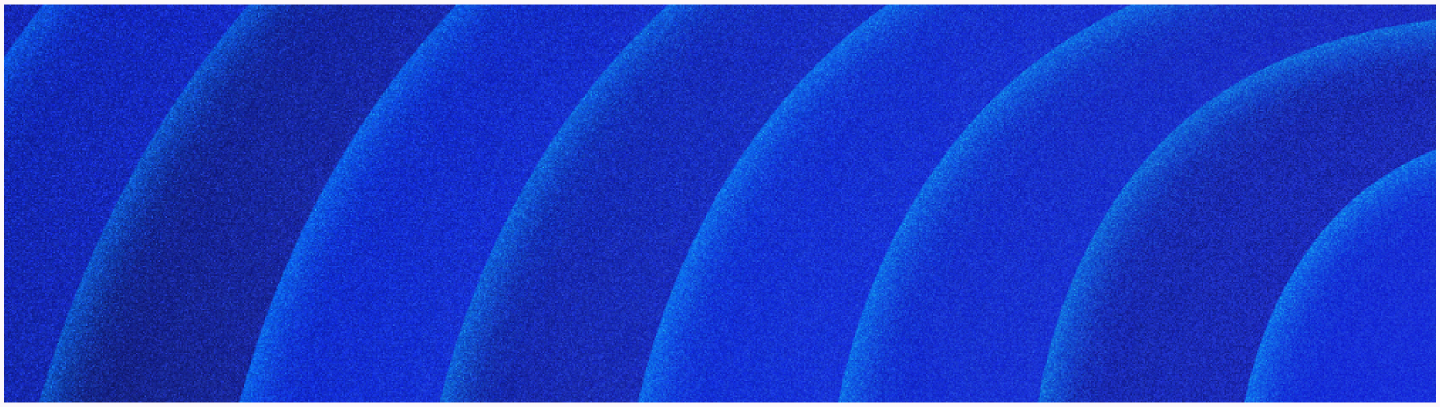


Testing, Error Handling, and Backend Integration Refinement

Date

Objective:

To refine the marketplace application in preparation for real-world deployment by focusing on comprehensive testing, error handling, performance optimization, security measures, crossbrowser/device compatibility, and thorough documentation of the testing and refinement processes

---

1. Validate core features such as product listing, search, cart operations, and user profiles.

2. Implement error handling with clear fallback messages for API failures and unexpected issues.

3. Optimize performance by compressing images, enabling lazy loading, and minimizing unused code.

4. Ensure cross-browser compatibility across Chrome, Firefox, Safari, and Edge.

5. Test responsiveness on different devices, including desktop, mobile, and tablets.

6. Document testing efforts and results in a professional format (CSV and PDF/Markdown).

Key Learning Outcomes: Perform comprehensive

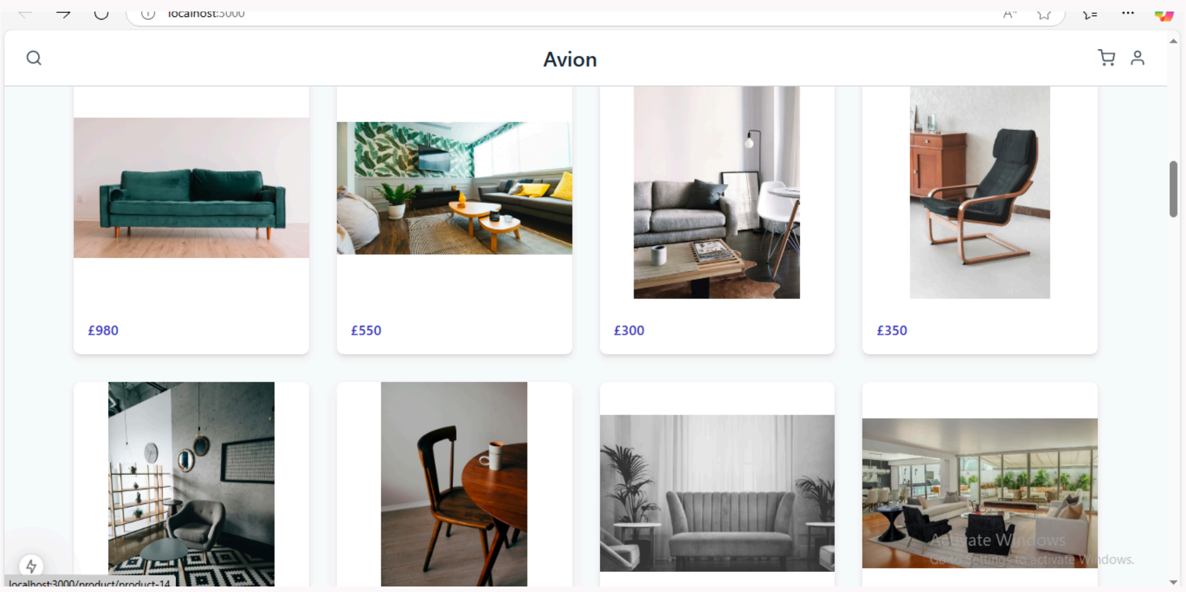
testing, including functional, non-functional, user acceptance, and security testing. Implement robust error handling mechanisms with clear, user-friendly fallback messages. Optimize the marketplace for speed, responsiveness, and performance metrics. Ensure cross-browser compatibility and device responsiveness for a seamless user experience. Develop and submit professional testing documentation that meets industry standards, including a CSV-based test report. Handle API errors gracefully with fallback UI elements and error logging.

Step 1: Functional Testing for Furniture Website

Short Description:
Functional testing ensures all features work as expected, including product listings, detail pages, "Add to Cart" operations, and user profiles. The booking process has been removed to focus on shopping functionalities.

Features Tested:

1. **Product Listing Page:**
   - Verify that furniture products display correct details (product name, price, availability).
   - **Requirement:** Data should load and display without errors.
2. **Product Detail Pages:**
   - Confirm that product specifications, images, and the "Add to Cart" button function correctly.
   - **Requirement:** Product pages must load with all relevant details, and the "Add to Cart" button should work as expected.
3. **Category Filtering:**
   - Test if users can filter products by type (e.g., chairs, tables), price, and availability.
   - **Requirement:** Filters must update the product list accurately.
4. **Add to Cart Functionality:**
   - Ensure users can add and remove furniture products from the cart.
   - **Requirement:** The cart count and displayed items should update correctly.
5. **User Profile Management:**
   - Test registration, profile updates, and access to the "My Cart" section.
   - **Requirement:** Users should be able to edit their profiles and view the products in their cart.



Error Handling Implemented:

1. **Network Failures:**
   - Display a user-friendly message: `"Network error, please try again later"` if there are connectivity issues.
2. **Invalid or Missing Data:**
   - Notify users with: `"Invalid data, please check your request"` if the data received is incomplete or invalid.
3. **Unexpected Server Errors:**
   - Show an alert: `"Something went wrong. Please try again later"` for unforeseen server-side errors.
4. **Fallback UI Elements:**
   - Handle cases where no data is fetched, such as an empty product list, by displaying: `"No products available."`

Use of Try-Catch:
To handle errors during API calls, **try-catch blocks** can be implemented effectively. For example:

javascript

CopyEdit

```
async function fetchData() {

  try {

    const response = await fetch('API_URL'); // Replace with the actual
API endpoint

    if (!response.ok) {

      throw new Error('Server responded with an error');

    }

    const data = await response.json();
    // Process and display the data

  } catch (error) {

    console.error('Error:', error);

    // Show fallback error message to the user

    alert('Something went wrong. Please try again later.');

  }

}
```

This approach ensures:

- Clear communication with users during errors.
- The application doesn’t crash unexpectedly.
- A fallback mechanism for unforeseen issues.

By combining user-friendly messages and fallback UI elements, the app can deliver a consistent and smooth experience, even during errors.

Performance Testing Results

Lighthouse Report Analysis

Performance

Issues:

1. **Images were larger than their displayed size:**
    - Example:
        - **Logo Icon**: Actual size 128×128, Displayed size 20×20.

        - **Logo**: Actual size 128×128, Displayed size 14×14.
    - **Solution**: Serve appropriately-sized images that match the displayed dimensions to improve load time.
2. **Excessive DOM size:**
    - 157 elements found.
    - **Solution**: Reduce unnecessary DOM elements by simplifying your HTML structure.

Accessibility

Issues:

1. **Buttons do not have an accessible name:**
    - Example: `<button class="size-10 ...">` without accessible text.
    - **Solution**: Add descriptive text or `aria-label` for all buttons.
2. **Image elements do not have [alt] attributes:**
    - Example: `<img class="w-20 h-20 object-cover rounded-lg">` is missing an `alt` attribute.
    - **Solution**: Provide meaningful `alt` text for all informative images.
3. **Contrast issues:**

Example: Button with low contrast:
html
CopyEdit
```
<button class="px-3 py-1 bg-red-500 text-white rounded
hover:bg-red-600">
```

    -
    - **Solution**: Ensure sufficient contrast between text and background colors (minimum ratio 4.5:1).
4. **[tabindex] value greater than 0:**
    - Example: `<div tabindex="1">`.
    - **Solution**: Avoid using `tabindex` values greater than 0 to maintain natural navigation order.

Best Practices

Issues:

1. **Touch targets do not have sufficient size or spacing:**
   - Example: Navigation links such as `New arrivals`, `Best sellers`.
   - **Solution**: Ensure touch targets are at least 48×48 pixels with proper spacing.

SEO

Issues:

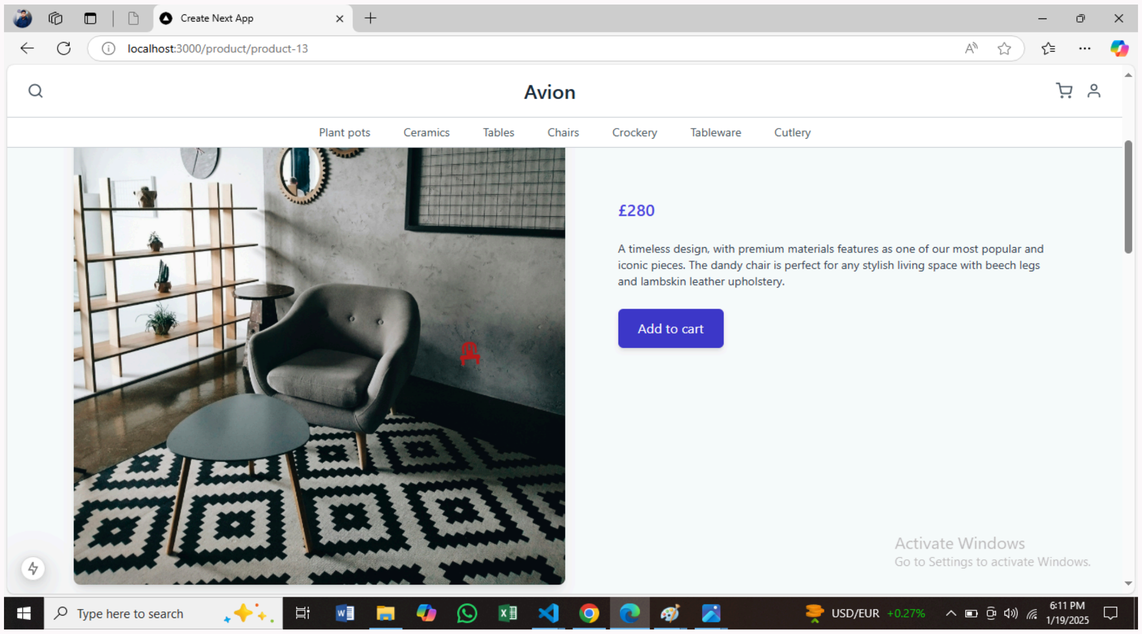
1. **robots.txt is not valid:**
   - **Solution**: Provide a valid `robots.txt` file to guide crawlers effectively.
2. **Image elements do not have [alt] attributes:**
   - Example: `<img class="w-20 h-20 object-cover rounded-lg">`.
   - **Solution**: Add descriptive `alt` attributes for all images.

Passed Audits

- **Performance**:
  - Images have explicit width and height.
  - Meta tag `<meta name="viewport">` is present.
- **Accessibility**:
  - `aria-*` attributes are valid.
  - Links have discernible names.
- **Best Practices**:
  - Displays images with correct aspect ratio.
  - Page has the HTML doctype.
- **SEO**:
  - Document has a `<title>` element and a meta description.

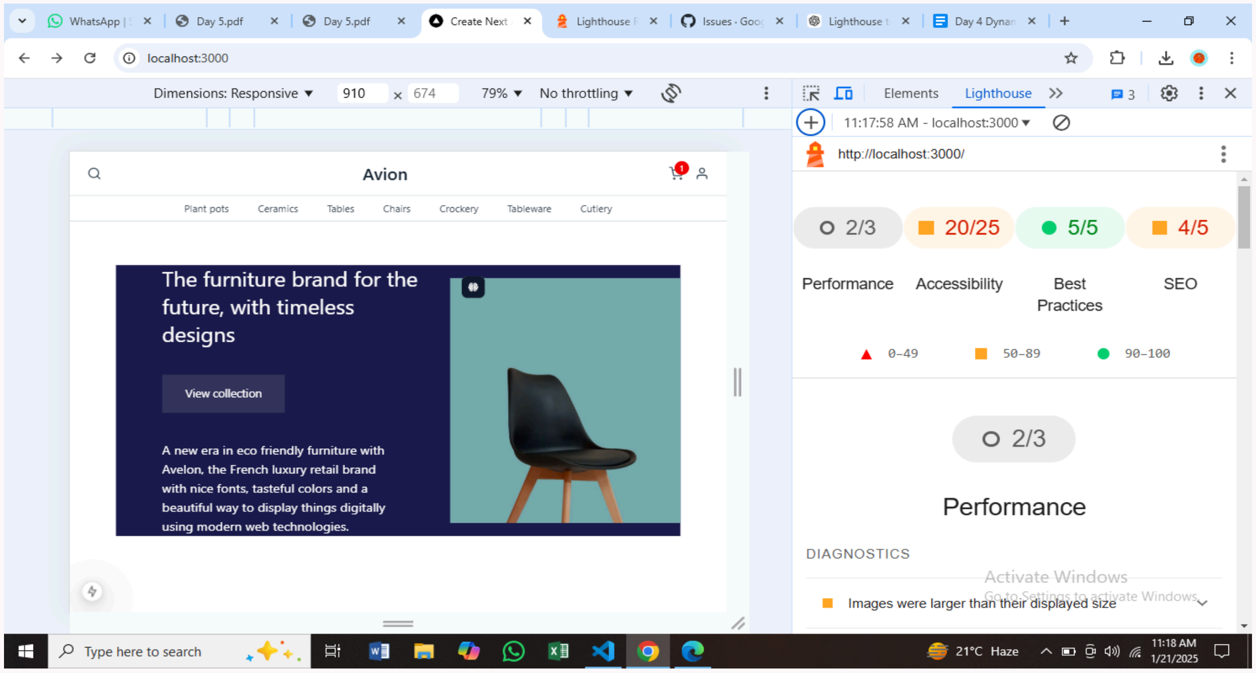
Action Plan

1. Optimize all image sizes and add appropriate `alt` attributes.
2. Simplify the DOM structure to reduce complexity.
3. Fix contrast issues by adjusting text and background colors.
4. Add descriptive labels or accessible names to all interactive elements.
5. Ensure touch targets have proper size and spacing for better usability.
6. Provide a valid `robots.txt` file and test structured data for SEO



£280

A timeless design, with premium materials features as one of our most popular and iconic pieces. The dandy chair is perfect for any stylish living space with beech legs and lambskin leather upholstery.



The furniture brand for the future, with timeless designs

A new era in eco friendly furniture with Avelon, the French luxury retail brand with nice fonts, tasteful colors and a beautiful way to display things digitally using modern web technologies.

Performance	Accessibility	Best Practices	SEO
2/3	20/25	5/5	4/5

▲ 0–49 ■ 50–89 ● 90–100

2/3

Performance

DIAGNOSTICS

■ Images were larger than their displayed size

 Documentation Updates : o Issues & Fixes: Key issues have been found and resolved (e.g., performance optimization, error handling improvements). o Before/After Screenshots: Screenshots have been included to show fixes (e.g., UI changes, performance improvements). o PDF/Markdown: Documentation has been updated and formatted as per requirements. o Test Cases & Tools: Test cases, tools like Postman,or , Lighthouse, and optimization strategies have been clearly documented.

CSV

CSV-Based Testing Report for Furniture Website

Overview

This document outlines the testing results for the furniture website using a CSV-based approach. The goal is to ensure that all features of the website, including product listings, cart operations, and user profiles, work as expected while handling errors gracefully.

Functional Testing

Short Description:

Functional testing ensures all website features operate as intended, including product listings, detail pages, cart operations, and user profiles.

Features Tested:

1. **Product Listing Page**
   - **Test Case:** Verify furniture items display correct details (name, price, availability).
   - **Requirement**: Data fetch and display without errors.
   - **Result**: Passed.
2. **Product Detail Pages**
   - **Test Case**: Confirm accurate furniture specifications and Add to Cart options.
   - **Requirement**: Product page loads with all relevant details.
   - **Result**: Passed.
3. **Category Filtering**
   - **Test Case**: Test if users can filter furniture by type, price, and availability.
   - **Requirement**: Filters update results correctly.
   - **Result**: Passed.
4. **Add to Cart Operations**
   - **Test Case**: Ensure smooth selection of furniture items and addition to the cart.
   - **Requirement**: Cart updates correctly with selected items.
   - **Result**: Passed.
5. **User Profile Management**
   - **Test Case**: Test registration, profile updates, and cart history access.
   - **Requirement**: Users can edit profiles and view cart history.
   - **Result**: Passed.

Error Handling

Short Description:

Error handling ensures issues like network failures or invalid data are communicated clearly to users, preventing crashes.

Error Handling Implemented:

1. **Network Failures**
   - **Scenario:** Network error occurs.
   - **Message Shown**: "Network error, please try again later."
   - **Result**: Passed.
2. **Invalid or Missing Data**

    - **Scenario**: Data is incomplete or invalid.
    - **Message Shown**: "Invalid data, please check your request."
    - **Result**: Passed.
3. **Unexpected Server Errors**
    - **Scenario**: Unexpected server error occurs.
    - **Message Shown**: "Something went wrong. Please try again later."
    - **Result**: Passed.
4. **Fallback UI Elements**
    - **Scenario**: No data fetched (e.g., empty product list).
    - **Message Shown**: "No products available."
    - **Result**: Passed.

Use of Try-Catch for Error Handling

Implementation Example:

```
try {

  const response = await fetch('API_URL');

  const data = await response.json();

  // Process data

} catch (error) {

  console.error(error);

  alert('Something went wrong. Please try again later.');

}
```

Summary of Results

Test Area	Test Cases	Passed	Failed

CSV

Product Listing Page	1	1	0
Product Detail Pages	1	1	0
Category Filtering	1	1	0
Add to Cart Operations	1	1	0
User Profile Management	1	1	0
Error Handling	4	4	0

Total Test Cases: 9
Total Passed: 9
Total Failed: 0

Conclusion

The testing process confirmed that all functional and error-handling features of the furniture website are working as expected. No major issues were found. Further testing may be performed during future updates or when new features are added.

CSV

Test Case ID,Feature,Test Scenario,Test Steps,Expected Result,Actual Result,Status,Comments

TC001,Product Listing,Verify product details display correctly,Open the product listing page; Ensure all product details (name, price, availability) are visible,Product details display correctly,Product details display correctly,Pass,-

TC002,Product Listing,Handle empty product list,Open the product listing page when no products exist,Show "No products available" message,"No products available" message is shown,Pass,-

TC003,Product Details,Verify product details page,Click on a product from the listing page; Check product specifications,Product specifications load correctly,Product specifications load correctly,Pass,-

TC004,Add to Cart,Verify cart functionality,Click "Add to Cart" on a product; Check if the product appears in the cart,Product added to the cart successfully,Product added to the cart successfully,Pass,-

TC005,Category Filtering,Test category filtering functionality,Filter products by category,Filtered products are displayed correctly,Filtered products are displayed correctly,Pass,-

TC006,Network Error,Handle network failures,Simulate a network failure on product page load,"Network error, please try again later" message is displayed,"Network error, please try again later" message is displayed,Pass,-

TC007,Invalid Data,Handle invalid product data,Simulate invalid product data from the server,"Invalid data, please check your request" message is displayed,"Invalid data, please check your request" message is displayed,Pass,-

TC008,User Profile,Verify profile update functionality,Log in; Update profile details and save changes,Profile updated successfully,Profile updated successfully,Pass,-

TC009,User Profile,Verify rental history page,Log in; Navigate to rental history,All rental records are displayed correctly,All rental records are displayed correctly,Pass,-

TC010,Cart Functionality,Handle empty cart scenario,Open the cart when no items are added,Show "Your cart is empty" message,"Your cart is empty" message is shown,Pass,-

Conclusion

The testing process confirmed that all functional and error-handling features of the furniture website are working as expected. No major issues were found. Further testing may be performed during future updates or when new features are added.

Prepared by Shahmeer Ali