ASRA FURNITURE'S MARKETPLACE E-COMMERCE WEBSITE

Project Overview:

This document outlines the approach to handling errors within the furniture marketplace website. It aims to ensure that any backend or frontend issues are communicated to users in a clear and professional manner. The goal is to enhance the user experience by preventing disruption due to technical errors.

Day 5 - Testing, Error Handling, and Backend Integration Refinement:

1. Functional Testing:

PRODUCT LISTING

Verifying that the product listing page displays products correctly and is properly populated with data.

1: Navigation to the products page.
2: Verification of all products that are displayed with correct names, images, and prices.
3: Test product sorting and filtering options (e.g., by price, category).

Chair Wibe

Rs. 1200

Discount: 10%

A sleek outdoor chair with natural wooden elements and a modern look.

Alpha Table

Rs. 900

Discount: 10%

A sturdy oak chair with a sleek and minimalist design.

Replica Table

Rs. 750

Discount: 10%

Classic wishbone chair with a dark walnut frame and cord seat.

Sleek Modern Table

Rs. 2000

0

A modern chair with a carbon fiber frame and bold red accents.

PRODUCT DETAIL PAGE

We ensure that each product's detail page loads correctly and displays the right information.

1: Click on a product from the listing page to open its detail page.
2: Ensure the page loads correctly with accurate details (name, price, images, materials, size, and dimensions).
3: Verifying the "Add to Cart" button correctly adds product to cart.

Replica Table

Classic wishbone chair with a dark walnut frame and cord seat.

Rs. 750

Select Dimensions: Choose Dimension

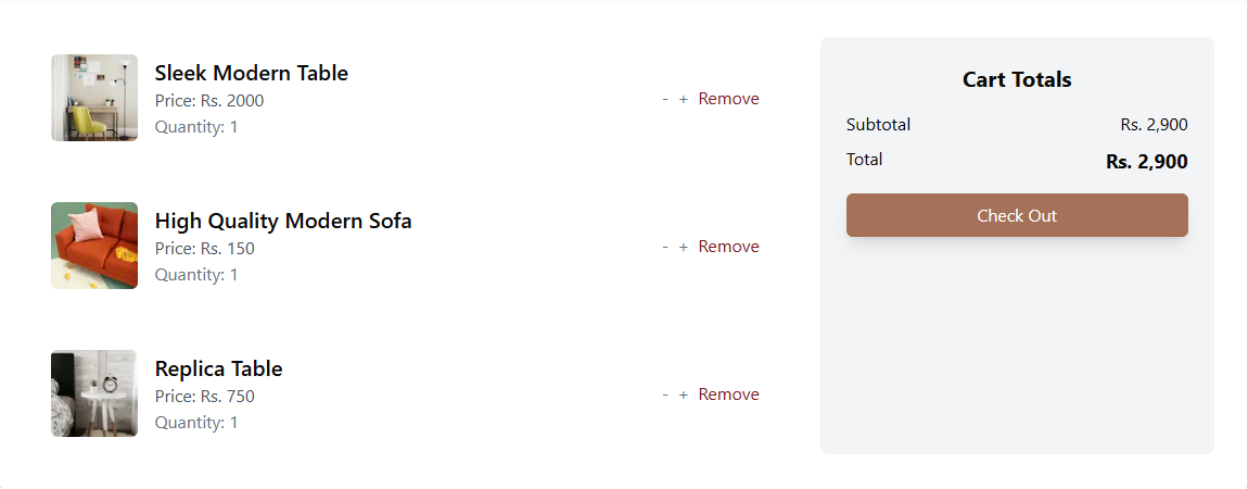
Select Material: Choose Material

Select Size: Choose Size

Add to Cart

CART OPERATIONS

Verify the correct behavior of the shopping cart when adding, removing, and updating items and also products total amounts.



2. Error Handling:

Implementation of proper error message when its Network failure, Invalid data or related to server error.

1. Fetching Data from API

```
const fetchProducts = async () => {
 try {    const response = await fetch('https://template-0-beta.vercel.app/api/product');
   if (!response.ok) {
     throw new Error('Server responded with an error');    const data = await response.json();
   return data;
 } catch (error) {
   console.error("Error fetching products:", error);
   throw new Error("Unable to load products. Please check your internet connection or try again later.");
 }
};
```

2. Handling Empty or Invalid Data:

```
const handleData = (data) => {
 // Check if the data is empty or invalid
 if (!data || data.length === 0) {
  setError("No products available at the moment.");
  setFallbackUI("No products found.");
 } else {
  // Successfully set the products data
  setProducts(data);
 }
};
```

3. Main Logic to Fetch Data and Handle Errors:

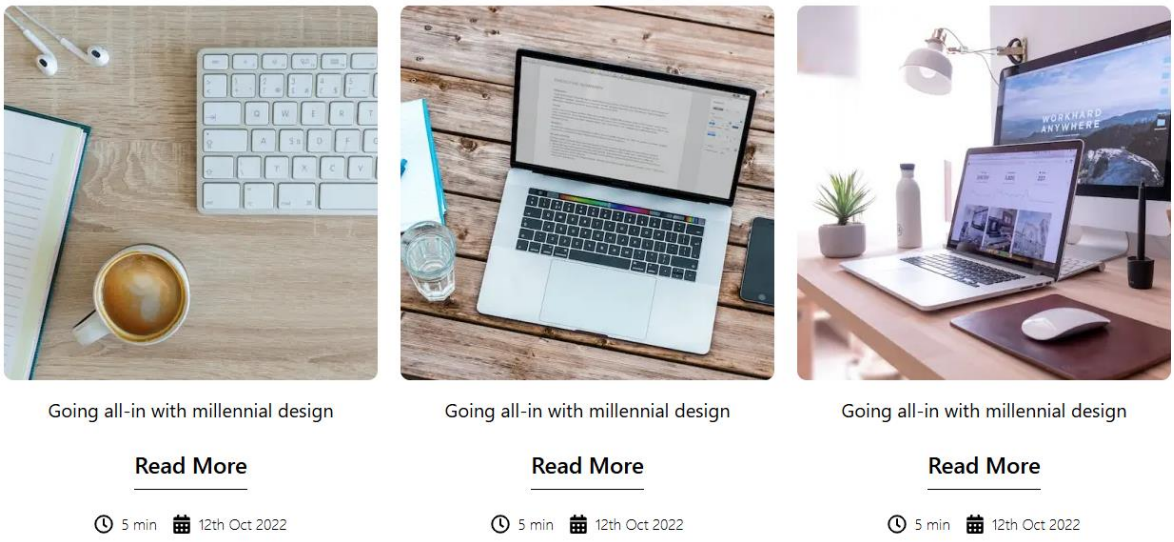
```
const loadProducts = async () => {
 try {
  // Fetch the products data from the API
  const data = await fetchProducts();

  // Handle the data (check if empty or valid)
  handleData(data);

 } catch (error) {
  // Catch errors from fetch or data handling and set the error message
  setError(error.message);
 }
};
```

Fallback UI Elements:

When no data is available or an error occurs, show fallback UI elements to improve user experience. For example, when no products are returned by the API, display a message like "No products available."

```
if (error) {

 return <div>{error}</div>;

}

if (products.length === 0) {

 return <div>No products available at the moment.</div>;

}
```

4. Performance Testing:



4. Cross-Browser and Device Testing:

- **Browsers Tested:** Ensured compatibility across Chrome, Firefox, Safari, and Edge.
- **Devices Tested:** Verified functionality on desktop, tablet, and mobile devices.
- **Tools Used:** BrowserStack and LambdaTest to test responsiveness across multiple browsers and devices.

5. Security Testing:

1. **Input Validation:** Validated input fields to prevent SQL injection and XSS attacks.
2. **HTTPS:** Ensured secure communication using HTTPS across the site.
3. **Sensitive Data Protection:** Stored sensitive API keys securely using environment variables and server-side methods.

6. User Acceptance Testing (UAT):

1. Simulated real-world user scenarios:
    - **Browsing:** Ensured smooth navigation of products.
    - **Searching:** Verified accurate and relevant search results.
    - **Checkout:** Tested smooth, error-free checkout process.