Chairify Platform: Testing, Error Handling, and Backend Integration Refinement

Objective

This document outlines the testing, error handling, and backend integration strategies for the Chairify platform. The goal is to ensure Chairify is fully functional, optimized for performance, and deployment-ready. Testing covers functional validation, error handling mechanisms, performance improvements, cross-browser/device compatibility, and user acceptance testing.

Key Areas of Focus

1. Comprehensive functional testing of core features.
2. Robust error handling mechanisms.
3. Performance optimization and accessibility compliance.
4. Cross-browser and device compatibility.
5. Professional documentation of findings and test cases.

Implementation Steps

Step 1: Functional Testing

Core Features to Test

1. **Product Listings:** Ensure all products are displayed correctly with accurate details such as name, price, description, and images.
2. **Filters and Search:** Validate that filters (e.g., category, price range) and the search function return accurate results.
3. **Shopping Cart Operations:** Test adding, updating, and removing items from the cart.
4. **Product Details Page:** Verify dynamic routing and the correct display of individual product details.
5. **User Account Management:** Validate user registration, login, and profile updates.
6. **Checkout Process:** Ensure smooth navigation through the checkout process, including payment gateway integration.

Tools Used

- **Postman:** API testing and response validation.
- **React Testing Library:** Component behavior testing.
- **Cypress:** End-to-end testing.

Test Cases

Test Case ID	Description	Steps	Expected Result	Actual Result	Status	Severity	Remarks
TC001	Test product listing display	Navigate to the product page. Verify product details are correct.	Products display with accurate details and images.	Products display with accurate details and images.	Pass	Medium	All product images and details are displayed correctly.
TC002	Validate filters	Apply price and category filters. Check displayed results.	Filters return accurate results.	Filters return accurate results as per criteria.	Pass	High	Filters are working as expected.
TC003	Test search functionality	Enter a search term. Verify returned results match the term.	Accurate search results are displayed.	Results match the search term entered by the user.	Pass	High	Search results match the term accurately.
TC004	Test add to cart	Add multiple items to the cart. Check cart contents.	Items are correctly added and displayed in the cart.	Items are added to the cart, and the cart displays the correct details.	Pass	High	Cart update reflects added items correctly.
TC005	Validate checkout	Complete the checkout process with valid payment details.	Successful order placement and confirmation message.	Checkout completes successfully, and an order confirmation message is shown.	Pass	Critical	Order is placed successfully without errors.
TC006	Test login and registration	Register a new user and log in.	User can register and log in successfully.	User registration and login work correctly without errors.	Pass	High	Login functionality works as expected.

TC007	Test product detail page	Click on a product. Verify routing and displayed details.	Correct product details page is loaded.	Product details page loads dynamically with accurate information.	Pass	High	Details page loads dynamically as expected.

Step 2: Error Handling

Strategies

Try-Catch Blocks: Use try-catch to handle API errors and provide meaningful messages to users.

```
 try {
   const data = await fetchProducts();
   setProducts(data);
} catch (error) {
   console.error("Failed to fetch products:", error);
   setError("Unable to load products. Please try again later.");
}
```

1. **Fallback UI:** Provide alternate content when data is unavailable.
    - Example: Display "No items found" for an empty product list.
2. **Error Logging:** Log errors for debugging.

Test Cases for Error Handling

Test Case ID	Description	Steps	Expected Result	Actual Result	Status	Severity	Remarks
TC008	Handle API failure gracefully	Simulate API failure and observe UI behavior.	Fallback message is displayed instead of crashing.	"Unable to load products. Please try again later." is displayed.	Pass	Critical	Fallback message appears correctly without crashing.
TC009	Test invalid login attempt	Enter incorrect credentials. Verify error message.	"Invalid credentials" message is shown.	"Invalid credentials" message is displayed correctly.	Pass	High	Error message displays as expected.

Step 3: Performance Optimization

Actions Taken

1. **Image Optimization:** Compressed all images using TinyPNG.
2. **Lazy Loading:** Implemented lazy loading for large images.
3. **Minification:** Minified CSS and JavaScript files.
4. **Performance Audit:** Used Google Lighthouse to identify bottlenecks.

Lighthouse Scores Before and After Optimization

Metric	Before	After
Performance	65	90
Accessibility	85	100
Best Practices	70	85
SEO	75	92

Test Case ID	Description	Steps	Expected Result	Actual Result	Status	Severity	Remarks
TC010	Test image optimization	Verify all images are compressed and load efficiently.	Images are optimized, and page load speed improves.	Images are compressed, and load time is reduced.	Pass	Medium	Images load faster after optimization.
TC011	Test lazy loading of images	Navigate through product pages with large images.	Images load only when in the viewport.	Images load dynamically when they enter the viewport.	Pass	Medium	Lazy loading works for large images.

Step 4: Cross-Browser and Device Testing

Browsers Tested

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Apple Safari

Devices Tested

- Desktop (Windows and macOS)
- Mobile (iOS and Android)
- Tablet

Tools Used

- **BrowserStack:** Simulated different devices and browser environments.

Test Cases for Cross-Browser Testing

Test Case ID	Description	Steps	Expected Result	Actual Result	Status	Severity	Remarks
TC012	Test rendering on Chrome	Open the platform on Chrome. Verify layout and functionality.	Layout renders correctly, and all functionalities work.	The layout and functionality are perfect in Chrome.	Pass	High	Platform renders as expected on Chrome.
TC013	Test responsive design	Resize browser window. Verify UI adjusts to screen size.	Responsive design adapts for all viewports.	The UI adjusts well for all tested screen sizes.	Pass	Medium	UI adjusts as expected for different screen sizes.

Step 5: Security Testing

Strategies

1. **Input Validation:**
    - Validate email and phone inputs using regular expressions.
    - Prevent SQL injection attacks by sanitizing inputs.
2. **Secure API Calls:**
    - Ensure all API calls use HTTPS.
    - Store sensitive keys in .env files.
3. **Tools Used:**
    - OWASP ZAP: Automated vulnerability scanning.
    - Burp Suite: Advanced penetration testing.

Step 6: Documentation Updates

Guidelines

1. Summarize test results, including key findings and resolutions.
2. Include a table of contents for easy navigation.

3. Format the document professionally.

Submission Format

- **Functional Deliverables:** Screenshots or recordings showcasing functionalities.
- **Testing Report:** Submit a detailed CSV-based report.
- **Documentation:** Compile a professionally formatted PDF or Markdown report.
- **GitHub Repository:** Upload all updated files with a clear folder hierarchy.

Conclusion

The Chairify platform has undergone rigorous testing, error handling implementation, and performance optimization. Remaining areas for improvement include enhancing SEO scores and conducting periodic security audits. This documentation and CSV report provide a clear roadmap for final deployment.