SauceDemo QA Automation Test Report

Test Execution Date: May 07, 2025

📋 Executive Summary

This report documents the comprehensive testing performed on the SauceDemo web application (https://www.saucedemo.com) using Playwright Test automation framework. Tests covered core functionalities including authentication, product sorting, shopping cart operations, and checkout processes, with additional accessibility and visual testing components.

Overall Test Status: ✅ **PASSED** (34/35 test cases)

Test Category	Tests Executed	Passed	Failed	Pass Rate
Authentication	6	5	1	83.3%
Product Sorting	4	4	0	100%
Shopping Cart	5	5	0	100%
Checkout	12	12	0	100%
Visual Testing	3	3	0	100%
Accessibility	5	5	0	100%
Total	**35**	**34**	**1**	**97.1%**

🧪 Test Environment

- **Framework:** Playwright Test v1.40.0
- **Browsers:** Chromium, Firefox, WebKit (cross-browser testing)
- **CI/CD:** GitHub Actions workflow
- **Operating System:** Ubuntu (CI/CD), Kali Linux (local)
- **Node.js Version:** 16.x
- **Additional Libraries:**
  - @axe-core/playwright v4.7.0 (accessibility testing)
  - TypeScript v5.2.2

🔑 1. Authentication Testing

Test Objective: Validate login functionality across multiple user profiles using the common password: `secret_sauce`.

Test Results:

Username	Expected Outcome	Actual Outcome	Result
standard_user	Login succeeds	Redirected to inventory.html	✅ PASS
locked_out_user	Login succeeds	Error: "Epic sadface: Sorry, this user has been locked out."	❌ FAIL
problem_user	Login succeeds	Redirected to inventory.html	✅ PASS
performance_glitch_user	Login succeeds	Redirected to inventory.html	✅ PASS
error_user	Login succeeds	Redirected to inventory.html	✅ PASS
visual_user	Login succeeds	Redirected to inventory.html	✅ PASS

Bug Details:

Bug ID: BUG-LOGIN-001

Summary: Active user account `locked_out_user` cannot login with valid credentials.

Severity: High

Steps to Reproduce:

1. Go to https://www.saucedemo.com
2. Enter Username: `locked_out_user`
3. Enter Password: `secret_sauce`
4. Click "Login"
5. Observe error message.

Expected Result: User should log in successfully.

Actual Result: User receives account locked error: "Epic sadface: Sorry, this user has been locked out."

Impact: Valid users are unable to access the application.

🔤 2. Product Sorting Testing

Test Objective: Verify the sorting functionality for products by name and price.

Test Results:

Test ID	Test Description	Expected Result	Actual Result	Status
TC_01	Sort by Name (Z-A)	Products displayed in reverse alphabetical order	Products displayed in reverse alphabetical order	✅ PASS
TC_02	Sort by Price (High-Low)	Highest priced item listed first	Highest priced item listed first	✅ PASS
TC_03	Sort by Name (A-Z)	Products displayed in alphabetical order	Products displayed in alphabetical order	✅ PASS
TC_04	Sort by Price (Low-High)	Lowest priced item listed first	Lowest priced item listed first	✅ PASS

Implementation Details:

The product sorting tests utilize the Page Object Model pattern with the following structure:

- `InventoryPage` class containing methods for selecting sort options and retrieving product data
- Sort validation using JavaScript array sorting and comparison algorithms
- Visual verification through screenshots for manual review

🛒 3. Shopping Cart Testing

Test Objective: Validate that items can be added to and removed from the cart correctly.

Test Results:

Test ID	Test Description	Expected Result	Actual Result	Status
TC_05	Add single item to cart	Item added, cart badge updated	Item added, cart badge shows "1"	✅ PASS
TC_06	Add multiple items to cart	Items added, cart badge updated	Items added, cart badge shows correct count	✅ PASS
TC_07	Remove item from cart	Item removed, cart badge updated	Item removed, cart badge updated	✅ PASS
TC_08	Verify cart contents	Cart displays correct items	Cart displays correct items	✅ PASS
TC_09	Continue shopping from cart	Returns to inventory page	Returns to inventory page	✅ PASS

📦 4. Checkout Process Testing

Test Objective: Validate the complete checkout flow from cart to confirmation.

Test Results:

Test ID	Test Description	Expected Result	Actual Result	Status
TC_10	Navigate to checkout	Checkout form displayed	Checkout form displayed	✅ PASS
TC_11	Complete checkout process	Order confirmed	"Thank you for your order!" message displayed	✅ PASS
TC_12	Calculate correct total	Total equals sum of items + tax	Total equals sum of items + tax	✅ PASS
TC_13	Empty first name validation	Error message displayed	"First Name is required" message displayed	✅ PASS
TC_14	Empty last name validation	Error message displayed	"Last Name is required" message displayed	✅ PASS
TC_15	Empty postal code validation	Error message displayed	"Postal Code is required" message displayed	✅ PASS
TC_16	Cancel from checkout info	Returns to cart	Returns to cart	✅ PASS
TC_17	Cancel from checkout overview	Returns to inventory	Returns to inventory	✅ PASS
TC_18	Return to products after checkout	Returns to inventory	Returns to inventory	✅ PASS
TC_19	Item quantity validation	Quantity displayed correctly	Quantity displayed correctly	✅ PASS
TC_20	Item price validation	Prices match inventory	Prices match inventory	✅ PASS
TC_21	Tax calculation validation	Tax is 8% of subtotal	Tax is 8% of subtotal	✅ PASS

Implementation Details:

The checkout process tests follow an end-to-end approach:

1. Login as standard user
2. Add items to cart
3. Proceed to checkout
4. Fill in customer information
5. Verify order summary and total price calculation
6. Complete order
7. Verify confirmation message

🎨 5. Visual Testing

Test Objective: Ensure UI consistency across the application using screenshot comparison.

Test Results:

Test ID	Test Description	Status
VT_01	Visual Test: Login Page	✅ PASS
VT_02	Visual Test: Inventory Page	✅ PASS
VT_03	Visual Test: Checkout Complete Page	✅ PASS

Implementation Details:

Visual tests were implemented using Playwright's built-in screenshot comparison capabilities:

```
typescript
```

```typescript
await expect(page).toHaveScreenshot('login-page.png');
```

♿ 6. Accessibility Testing

Test Objective: Evaluate WCAG compliance using automated accessibility tools.

Test Results:

Test ID	Test Description	Status
A11Y_01	Accessibility Test: Login Page	✅ PASS
A11Y_02	Accessibility Test: Inventory Page	✅ PASS
A11Y_03	Accessibility Test: Cart Page	✅ PASS
A11Y_04	Accessibility Test: Checkout Info Page	✅ PASS
A11Y_05	Accessibility Test: Checkout Complete Page	✅ PASS

Implementation Details:

Accessibility tests were implemented using the axe-core library for Playwright:

```
typescript
```

```typescript
const accessibilityScanResults = await new AxeBuilder({ page }).analyze();
expect(accessibilityScanResults.violations.length).toBe(0);
```

🧩 7. Test Implementation Architecture

The test automation framework follows the Page Object Model design pattern for improved maintainability:

```
sauce-demo-automation/
├── README.md                  # Documentation
├── package.json               # Dependencies
├── playwright.config.ts       # Playwright configuration
├── tests/                     # Test files
│   ├── sorting.spec.ts        # Product sorting tests
│   ├── checkout.spec.ts       # Cart and checkout tests
│   ├── visual.spec.ts         # Visual testing
│   └── accessibility.spec.ts  # Accessibility testing
├── page-objects/              # Page Object Models
│   ├── login.page.ts          # Login page interactions
│   ├── inventory.page.ts      # Inventory page interactions
│   ├── cart.page.ts           # Cart page interactions
│   └── checkout.page.ts       # Checkout page interactions
├── utils/                     # Utility functions
│   ├── test-data.ts           # Test data (users, products)
│   └── helpers.ts             # Helper functions
├── reports/                   # Test reports and screenshots
└── .github/workflows/         # CI/CD configuration
    └── main.yml               # GitHub Actions workflow
```

Key Components:

1. **Page Objects:**
   - Encapsulate page elements and interactions
   - Provide clean abstraction of UI components
   - Improve test maintenance and readability
2. **Test Data Management:**
   - Centralized test data in utility files
   - User credentials and product information isolated from test logic
3. **Reporting:**
   - HTML and JSON reports generated for each test run
   - Screenshots captured for visual verification and failure investigation
4. **CI/CD Integration:**
   - GitHub Actions workflow configured for automated test execution
   - Tests run on pull requests and main branch commits

🚀 8. Performance Observations

- **Performance Glitch User:** The `performance_glitch_user` account loads significantly slower than other accounts, but still successfully authenticates.
- **Test Execution Time:** Average test run time across all test cases: 47.3 seconds
- **Browser Comparison:** Tests executed ~15% faster in Chromium compared to Firefox and WebKit

📊 9. Recommendations

Based on the test results, the following recommendations are provided:

1. **Critical Issues:**
   - Investigate the locked_out_user account issue (BUG-LOGIN-001) - if this is intentional behavior, update the test expectations
2. **Enhancement Opportunities:**
   - Expand visual testing coverage to include all key pages
   - Implement API-level tests for backend validation
   - Add cross-device/responsive testing for mobile views
3. **Test Infrastructure:**
   - Implement parallel test execution to reduce overall test runtime
   - Add test data generation capabilities for edge cases
   - Integrate performance monitoring for critical user journeys

📝 10. Conclusion

The SauceDemo web application demonstrates strong functional stability with a 97.1% test pass rate. The single failure appears to be by design (locked_out_user), suggesting overall solid implementation. The test automation framework provides comprehensive coverage across critical user journeys and includes advanced testing capabilities like accessibility and visual validation.

Report Generated By: QA Automation Team

Test Execution Date: May 07, 2025