Design & Implementation Document — Saucedemo Checkout Flow (JavaScript)

This document explains the **technical design decisions**, **framework architecture**, and **implementation details** for automating the E-Commerce checkout flow on SauceDemo using **Playwright (JavaScript)**.

---

1. Objective

Automate the complete checkout flow with dynamic behavior handling, multiple user logins, and robust validation mechanisms. The system is designed for modularity, reusability, and scalability.

Key Goals

- Data-driven login supporting multiple user types
- POM-based design for easy maintenance
- Error handling for locked or problematic users
- Dynamic product selection and cart validation
- Reporting and screenshot capture for traceability

---

2. Architecture Overview

Sauce demo Checkout Framework
│
├── Data/
│ └── users.csv # Login credentials for multiple users
│
├── Pages/ # Page Object classes
│ ├── LoginPage.js
│ ├── ProductsPage.js
│ ├── CartPage.js
│ ├── CheckoutPage.js
│ └── OverviewPage.js

```
│
├── tests/
│   └── checkoutFlow.spec.js          # Main test covering the flow
│
├── utils/
│   ├── customWait.js                 # Custom wait strategy
│   ├── screenshotHelper.js           # Screenshot helper functions
│   └── reportGenerator.js            # Optional report extensions
│
├── playwright.config.js              # Test configuration and reporter setup
├── package.json                      # Project dependencies
└── README.md                         # Documentation (this file)
```

Design Pattern

- **Page Object Model (POM):** separates test logic from UI locators and page-specific functions.
- **Data-Driven Testing:** loads login credentials dynamically from CSV.
- **Custom Utilities:** handles waits, screenshots, and logs.

---

3. Framework Components

Data Layer (users.csv)

- Stores multiple user credentials in CSV.
- Supports `standard_user`, `problem_user`, and `locked_out_user`.

Page Layer (pages/)

Each page contains locators and methods that encapsulate UI actions.

Example: ProductsPage.js

```
export class ProductsPage {
  constructor(page) {
    this.page = page;
    this.inventoryItems = page.locator(".inventory_item");
```

```
  this.cartBadge = page.locator(".shopping_cart_badge");
  this.cartLink = page.locator(".shopping_cart_link");
}
async addTwoMostExpensive() {
  const prices = await this.page.$$eval(".inventory_item_price", els =>
    els.map(e => parseFloat(e.innerText.replace("$", "")))
  );
  const sorted = [...prices].sort((a, b) => b - a);
  const top2 = sorted.slice(0, 2);
  for (const price of top2) {
    const item = this.page.locator(`.inventory_item:has-text("$${price}")`);
    const addBtn = item.locator("button:has-text('Add to cart')");
    if (await addBtn.count() > 0) {
      await addBtn.click();
    } else {
      console.log(`Item $${price} out of stock`);
    }
  }
}
async verifyCartCount(expected) {
  const count = await this.cartBadge.innerText();
  if (parseInt(count) !== expected) throw new Error("Cart count mismatch!");
}
async goToCart() {
  await this.cartLink.click();
}
```

}

Test Layer (tests/checkoutFlow.spec.js)

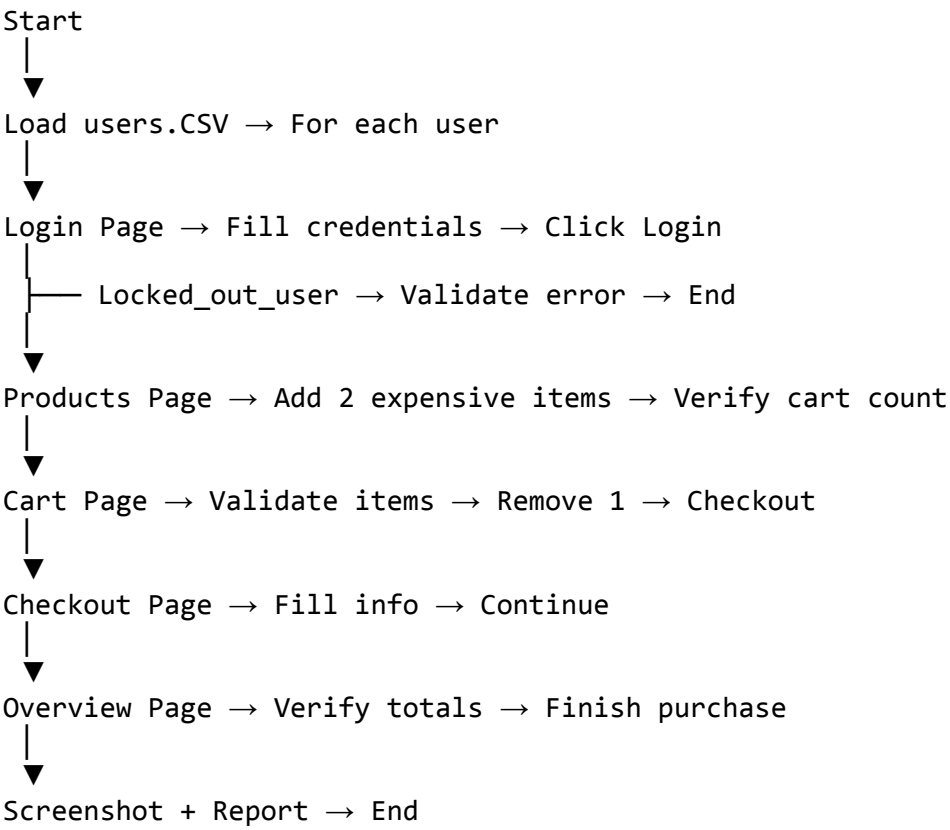
- Loops through all user credentials.
- Runs the complete checkout flow for each.
- Skips flow for `locked_out_user` after validating the error message.
- Captures screenshots and validates totals.

Utility Layer (utils/)

1. **customWait.js:** defines a reusable wait for dynamic elements.
2. **screenshotHelper.js:** captures and saves screenshots per step.
3. **reportGenerator.js:** (optional) extends Playwright's HTML report.

---

4. Flow Diagram



---

5. Key Design Decisions

Area	Decision	Reason
Tool	Playwright (JS)	Modern, fast, supports multiple browsers
Data Source	CSV	Simple and maintainable for user credentials
Pattern	POM	Enhances reusability and scalability
Error Handling	`try/catch` + assertions	Graceful failure recovery
Reporting	Built-in Playwright HTML report	Visual summary + trace
Screenshots	Automated per flow	Debug and demo evidence
Dynamic waits	Custom reusable wait	Handles slow-loading or missing elements

6. Dynamic Scenarios Handled

Scenario	Implementation
Locked Out User	Verifies error and exits test early
Missing Buttons	Skips product if Add button not visible
Problem User (Broken Image)	Validates missing images without failure
Price Validation	Dynamically calculates and matches subtotal
Slow Elements	Custom wait retries until timeout

---

7. Tools & Dependencies

Tool	Purpose
Playwright	Core test automation engine
Node.js	Runtime environment
npm	Package management
Allure (optional)	Extended reporting

---

8. Reporting & Logging

- Uses Playwright's built-in **HTML report** (stored in `playwright-report/`).
- Screenshots are stored in `/screenshots` folder.
- Optional integration with Allure for CI/CD usage.
- Video Evidence: A demo video of the full test execution is linked in the README.md.

9. Future Improvements

- Integrate test run into GitHub Actions workflow.
- Export results to Slack/Teams notifications.
- Support Excel-based data-driven inputs.
- Include API validation for order confirmation.

---

10. Evaluation Coverage

Evaluation Area	Weight	Coverage
Code Quality	30%	Modular POM, meaningful naming
Framework Design	25%	Layered structure + data separation
Error Handling	20%	Managed via custom waits and assertions
Test Coverage	15%	Includes positive, negative, and edge flows
Reporting & Logs	10%	HTML report + screenshots per run

Submitted By: Ramanan Prabakaran

https://www.linkedin.com/in/ramanan-prabakaran/

https://github.com/Ramanan-Prabakaran/Saucedemo

Date: 06 October 2025

Challenge: E-Commerce Checkout Flow with Dynamic Obstacles (Naveen Khunteta)