TEST PLAN DOCUMENT FOR SAUCEDEMO WEB APPLICATION

1. Objectives & Scope

1.1 Objectives

- Ensure the application allows users to **log in** successfully with valid credentials and blocks invalid ones.
- Validate users can **browse products**, **add products to cart**, **remove items**, **checkout**, and **complete orders** accurately.
- Ensure data consistency across user sessions (cart contents persist appropriately, order history if applicable).
- Verify UI is responsive and usable across desktop and tablet/phone screen sizes.
- Confirm that the application behaves correctly under different browsers and degrades gracefully under constrained network conditions.
- Validate that key flows are functional and that the application recovers appropriately from errors (e.g., invalid login, missing field at checkout).

1.2 Scope

Included:

- Functional testing of user flows: login/logout, product listing, product detail, add to cart, cart update/remove, checkout (information, overview, complete), logout.
- UI/UX responsiveness testing for multiple screen sizes (desktop, tablet, mobile).
- Cross-browser compatibility testing (at least Chrome, Firefox, Safari, Edge).
- Network-condition testing: slow connections, intermittent connectivity (where feasible) to verify application handling.
- Automation of regression flows for stable features (login/logout, add/remove cart, checkout).
- Manual exploratory testing for edge cases, negative flows, usability.

Excluded:

- Extensive performance/stress testing with thousands of users (outside current scope).
- Extensive compatibility testing on rare/outdated browsers or devices beyond major ones.

2. Assumptions, Environment & Tools

2.1 Assumptions

- The application is deployed in a staging/QA environment that replicates the production UI/flow of SauceDemo.
- Test data (user credentials, product list) is stable and known in advance for consistent testing.
- The test team has access to necessary browsers/devices or device emulators for responsive testing.
- The network simulation (slow connection) tools or browser dev tools will be available for constrained-network testing.
- Automation scripts will be maintained and updated as the application evolves; changes in UI locators will be managed by the automation engineer.
- Defect management tool (e.g., Jira, GitHub Issues) and test-case management tool (e.g., TestRail) are available and accessible to the test team.

2.2 Environment

- **QA/Staging Environment**: Mirror of production with latest stable build for testing.
- **Browsers/Devices**:
    - Desktop: Chrome (latest), Firefox (latest), Edge (latest), Safari (latest)
    - Mobile/Tablet: Chrome on Android (latest), Safari on iOS (latest), BrowserStack/SauceLabs (if available) for multiple devices
- **Network Condition Simulation**: Browser dev tools or network throttling tool to simulate 2G/3G, intermittent drops.
- **Test Data**: Valid user credentials, locked‑out user, problem‑user (if SauceDemo offers one), invalid credentials, sample product list.
- **Test Environments Setup**: Access to browser console logs, ability to clear cache/cookies, ensure clean sessions for each test.

2.3 Tools

- **Test Case Management**: TestRail (or equivalent) – for documenting test cases, tracking status.
- **Defect/Bug Tracking**: Jira or GitHub Issues – for logging defects with reproduction steps, environment details, screenshots.
- **Automation Framework**:
    - Playwright or Selenium (preferably Playwright for modern features) for browser-based automated test execution.
- **API Testing Tool**: Postman (optional, if any APIs are exposed) used a mocked API from the platform https://dummyjson.com
- **Performance/Network Testing**: Browser dev tools network throttling, or a simple tool like JMeter (if basic performance tests are needed)
- **Collaboration / Reporting**: Slack/Teams for communication, Confluence or shared docs for test plan/reference.

3. Manual & Automated Test Strategy

3.1 Manual Testing Strategy

3.1.1 Approach

- Execute scripted functional test cases covering all core user flows (login/logout, shopping, checkout).
- For each screen size/device category (desktop, tablet, mobile), perform UI/UX verification manually to ensure responsiveness and usability.
- Perform exploratory testing: testers will explore edge cases (e.g., attempting checkout with no items, invalid login attempts, session timeout, navigating back during checkout).
- Network resilience testing manually: using browser dev tools simulate slow connection, intermittent connectivity, and observe system behaviour (timeouts, retry, data integrity).
- Record all test results in TestRail (or equivalent) with status (Pass/Fail), details of any defects, environment, browser, device.
- Defects logged into Jira/GitHub with full reproduction steps, screenshots, browser/device info. Use severity/priority criteria as per defect strategy.

3.1.2 Coverage

- Functional: all user flows plus negative/edge scenarios.
- Non‑functional: usability, compatibility (cross‑browser/device), network resilience.
- Use test matrix:
    - Browser × Device × Flow combinations.
    - Network condition × Flow combinations (especially checkout and add/remove cart).
- Manual testing will focus on new features, complex scenarios, and exploratory sessions. Regression of core flows will also be included but with automation emphasis (see below).

3.2 Automated Testing Strategy

3.2.1 Purpose

- Provide regression safety for core flows, enabling faster feedback with each build.
- Reduce manual effort for repetitive scenarios (login, add/remove cart, checkout).
- Facilitate cross-browser automation to cover major browsers/devices.

3.2.2 Scope of Automation

- Login/logout flow: valid credentials, invalid credentials.
- Add to cart: select one or multiple products, remove items, verify cart contents.
- Checkout flow: fill in information, overview, complete, verify success message.
- Cross-browser execution: Chrome, Firefox, Edge (and optionally Safari if tool supports).
- Smoke tests: a minimal set of tests executed on each build to validate that the system is testable.

3.2.3 Framework & Implementation

- Use Playwright (preferred) or Selenium with a test automation framework (e.g., JUnit/TestNG for Java or Jest/Mocha for JS).
- Test scripts organised by modules: LoginPage, ProductPage, CartPage, CheckoutPage, etc.
- Use Page Object Model (POM) design pattern for maintainability.
- Integrate into CI/CD pipeline (e.g., when code is committed, build triggers, automation runs).
- Use test data files (e.g., JSON/CSV) to parameterise tests.
- Execution environments: local and remote (e.g., browser cloud, Docker).
- Reports: Generate HTML reports, send summary to QA lead, record results in test management tool if integrated.

3.2.4 Maintenance & Risks

- Locators may change frequently - plan for maintenance of scripts.
- Test data must remain stable or scripts will fail - coordinate with dev/team to manage data resets.
- Cross-browser support may incur flakiness - use retry logic and proper environment stabilisation.

4. Risks & Mitigations

- **Risk:** UI/locators change often → Automation scripts break.
  Mitigation: Use POM and maintain locator library; schedule periodic review.
- **Risk:** Test data instability (users/cart state) → inconsistent results.
  Mitigation: Coordinate with dev for a reset script; use clean profiles for each test.
- **Risk:** Network simulation may not reflect real world.
  Mitigation: Combine browser dev-tool simulation with manual ad-hoc testing on low-connectivity devices.
- **Risk:** Cross-browser/device environment access limited.
  Mitigation: Use browser-cloud services (BrowserStack, SauceLabs) or device emulators as fallback.
- **Risk:** Automation framework integration issues with CI/CD.
  Mitigation: Early integration trial, include smoke tests first, monitor build stability.

5. Test Deliverables

- Test Plan (this document)
- Test Cases (manual and automated)
- Automation Scripts & Reports
- Defect Logs & Summary Report
- Test Execution Report (status, passed/failed, unresolved defects)