# Final Report – SauceDemoAutomationSuite

---

## Part 1: UI Testing – SauceDemo

### Selected Features

1. **Login Functionality**
2. **Cart Functionality**

### Test Coverage Overview

The test suite covers two key UI features each targeting critical paths a real user would follow. The automated scenarios focus on validating both success and failure states without overloading the suite with low-value tests. Full test case details are documented separately.

### Framework Architecture (UI)

- **Framework:** Playwright with C#
- **Design Pattern:** Page Object Model (POM)
- **Structure:**
    - Pages/: Encapsulates UI logic
    - Tests/: Functional test classes
    - Models/: Data abstraction for products
    - Utils/: Test lifecycle setup (TestBase.cs)
- **Locator Strategy:** data-test attributes + dynamic XPath
- **Reusable Logic:** Dynamic methods like AddItemToCart(string productName)
- **Test Runner:** NUnit

## Part 2: API Testing – Swagger Petstore

### Selected Endpoints

1. POST /pet -> Add Pet
2. GET /pet/{petId} -> Retrieve Pet

### Test Coverage Overview

API tests were created for two endpoints: POST /pet and GET /pet/{petId}. The suite includes tests for valid requests and negative cases like schema violations and missing data. Tests target realistic failure scenarios to ensure API behavior is consistent and predictable. Full test case details are documented separately.

### Framework Architecture (API)

- **Framework:** C# using HttpClient (within Playwright solution)
- **Design:**
    - ApiClient: Reusable wrapper for API calls
    - Models/: Pet.cs used for serialization
- **Request Types:** POST, GET
- **Validation:** Status code + JSON response parsing via Newtonsoft.Json
- **Edge Case Handling:** PostRawAsync() used for schema-violating tests

**Note:** Due to quirks in the Swagger Petstore API, some invalid or incomplete inputs return 404 Not Found instead of the expected 400 Bad Request. The tests and reports reflect this behavior to match real outcomes.

## Design Highlights

- Used **clean architecture** principles and POM for UI
- API and UI tests are structured as separate projects under a single solution
- Avoided redundant automation, only **high-value tests were implemented**
- Created **data abstraction models** (ProductItem) for clarity
- Test logic is reusable and scalable for future cases

---