2026 February

ICT UPSKILLING PROGRAM

Applied Software Testing & QA

BY MAI TAHA

Contents

Project Overview

In this project, four different software testing techniques were required and applied to ensure software quality and reliability:

- Manual Testing
- Automation Testing
- API Testing
- Performance Testing

SauceDemo

A demo e-commerce web application used to test user workflows such as login, product browsing, cart management, and checkout.

DummyJSON

A mock backend REST API used to test API functionality, request/response handling, and system performance under load.

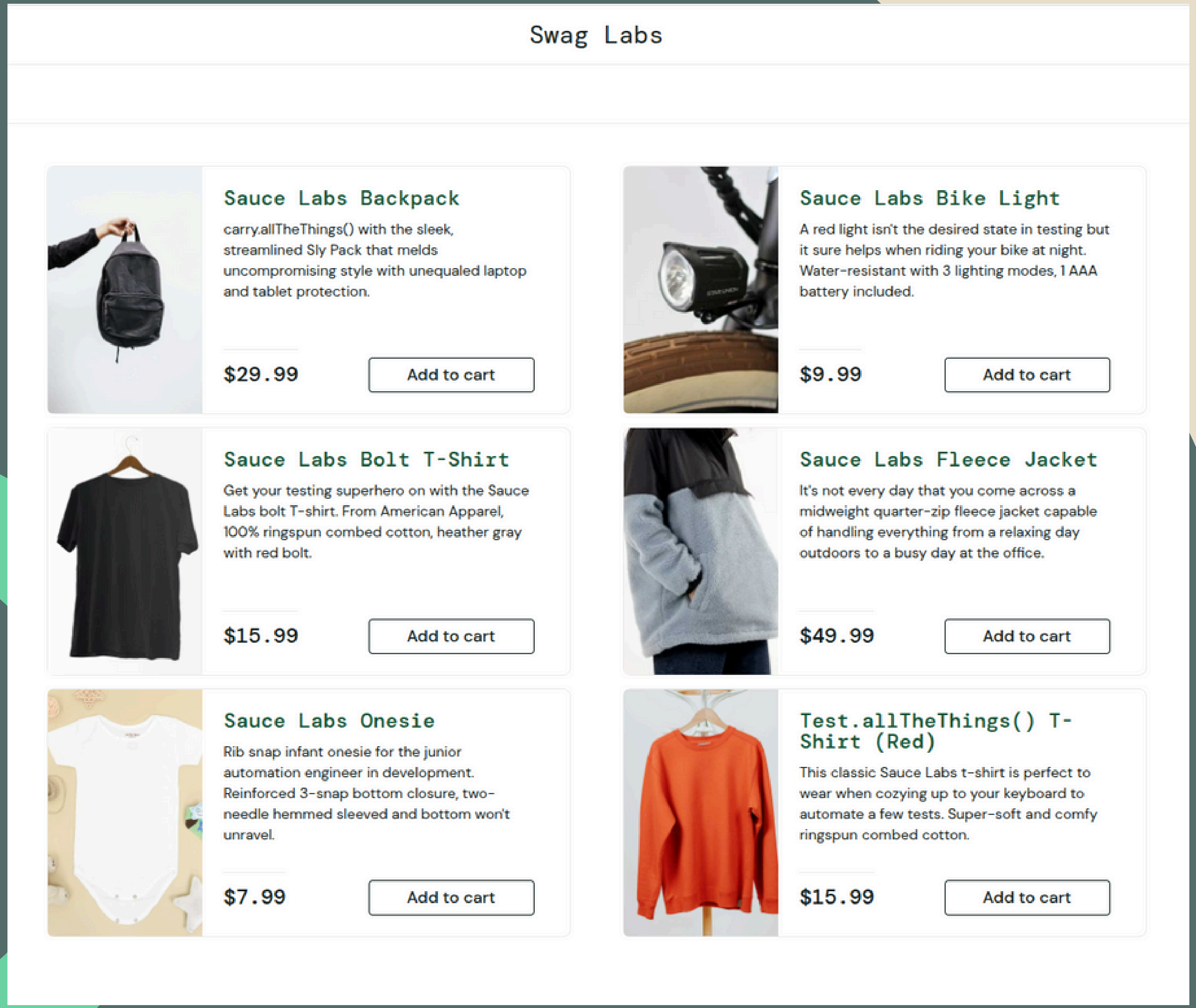
Manual Testing (SauceDemo)

Test Plan

A manual test plan was created for the SauceDemo web application to define the testing scope and objectives.

The test plan included:

- Testing core e-commerce features such as login, product browsing, cart operations, and checkout
- In-scope and out-of-scope areas
- Assumptions and dependencies
- Test approach based on system-level functional testing



Swag Labs

Sauce Labs Backpack
carry.allTheThings() with the sleek, streamlined Sly Pack that melds uncompromising style with unequaled laptop and tablet protection.
$29.99 — Add to cart

Sauce Labs Bike Light
A red light isn't the desired state in testing but it sure helps when riding your bike at night. Water-resistant with 3 lighting modes, 1 AAA battery included.
$9.99 — Add to cart

Sauce Labs Bolt T-Shirt
Get your testing superhero on with the Sauce Labs bolt T-shirt. From American Apparel, 100% ringspun combed cotton, heather gray with red bolt.
$15.99 — Add to cart

Sauce Labs Fleece Jacket
It's not every day that you come across a midweight quarter-zip fleece jacket capable of handling everything from a relaxing day outdoors to a busy day at the office.
$49.99 — Add to cart

Sauce Labs Onesie
Rib snap infant onesie for the junior automation engineer in development. Reinforced 3-snap bottom closure, two-needle hemmed sleeved and bottom won't unravel.
$7.99 — Add to cart

Test.allTheThings() T-Shirt (Red)
This classic Sauce Labs t-shirt is perfect to wear when cozying up to your keyboard to automate a few tests. Super-soft and comfy ringspun combed cotton.
$15.99 — Add to cart

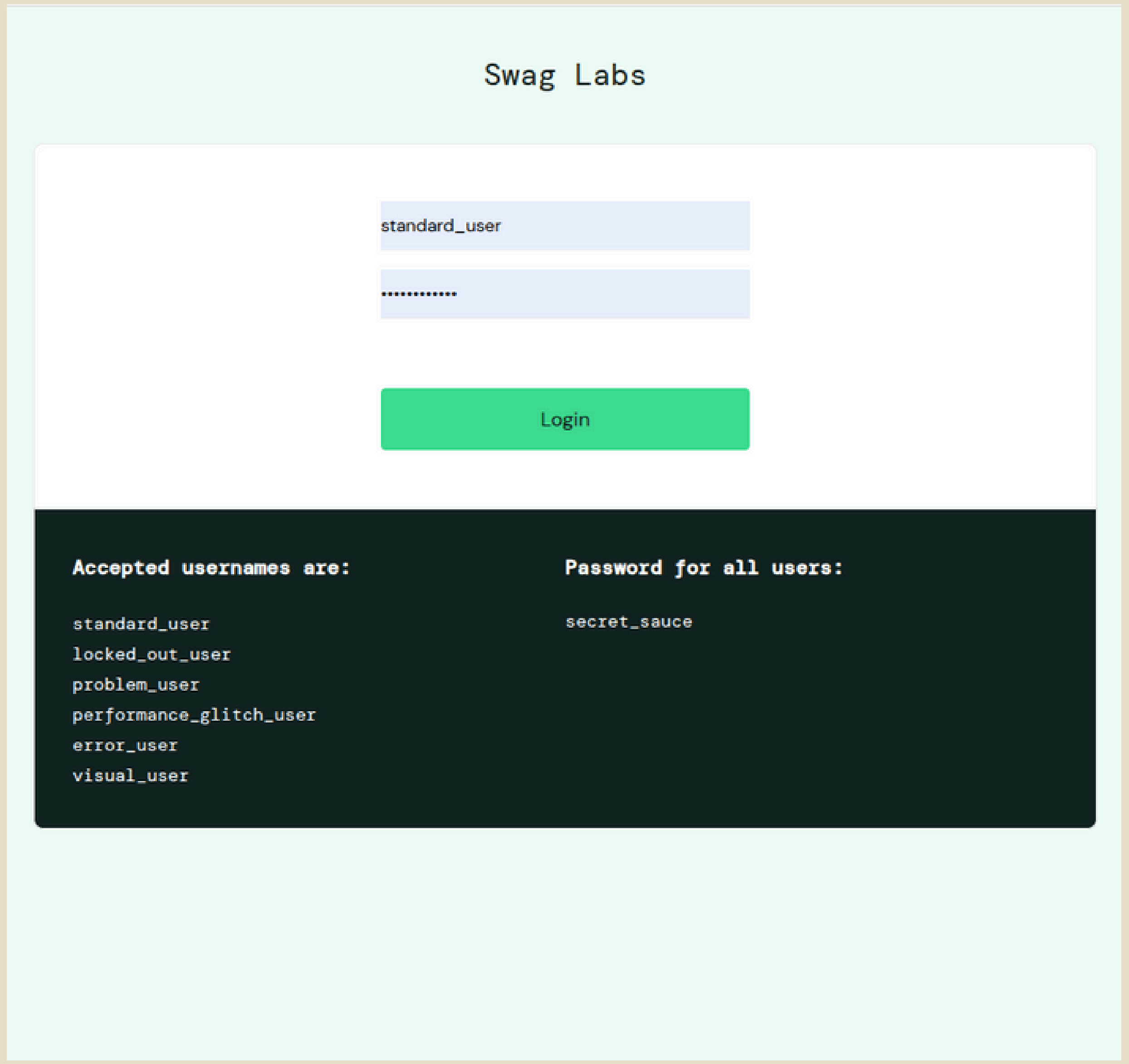
Test Execution
Manual Testing

After finalizing the test plan, manual testing was executed by running test scenarios that simulate real user behavior.

Test cases were designed and executed to cover:

- Happy path scenarios using valid inputs
- Negative scenarios using invalid credentials and incorrect actions

During execution, test results were recorded, defects were reported with clear reproduction steps, and retesting was performed to verify defect fixes.

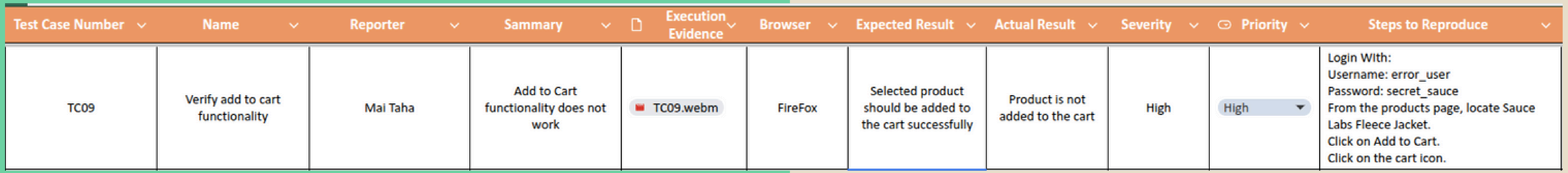


Accepted usernames are:

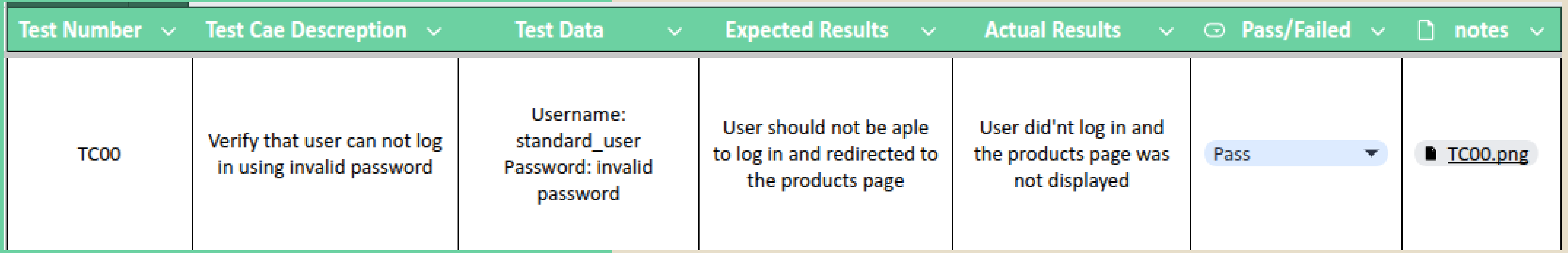
standard_user
locked_out_user
problem_user
performance_glitch_user
error_user
visual_user

Password for all users:

secret_sauce



Test Case Number	Name	Reporter	Sammary	Execution Evidence	Browser	Expected Result	Actual Result	Severity	Priority	Steps to Reproduce
TC09	Verify add to cart functionality	Mai Taha	Add to Cart functionality does not work	TC09.webm	FireFox	Selected product should be added to the cart successfully	Product is not added to the cart	High	High	Login With: Username: error_user Password: secret_sauce From the products page, locate Sauce Labs Fleece Jacket. Click on Add to Cart. Click on the cart icon.



Test Number	Test Cae Descreption	Test Data	Expected Results	Actual Results	Pass/Failed	notes
TC00	Verify that user can not log in using invalid password	Username: standard_user Password: invalid password	User should not be aple to log in and redirected to the products page	User did'nt log in and the products page was not displayed	Pass	TC00.png

(SauceDemo)

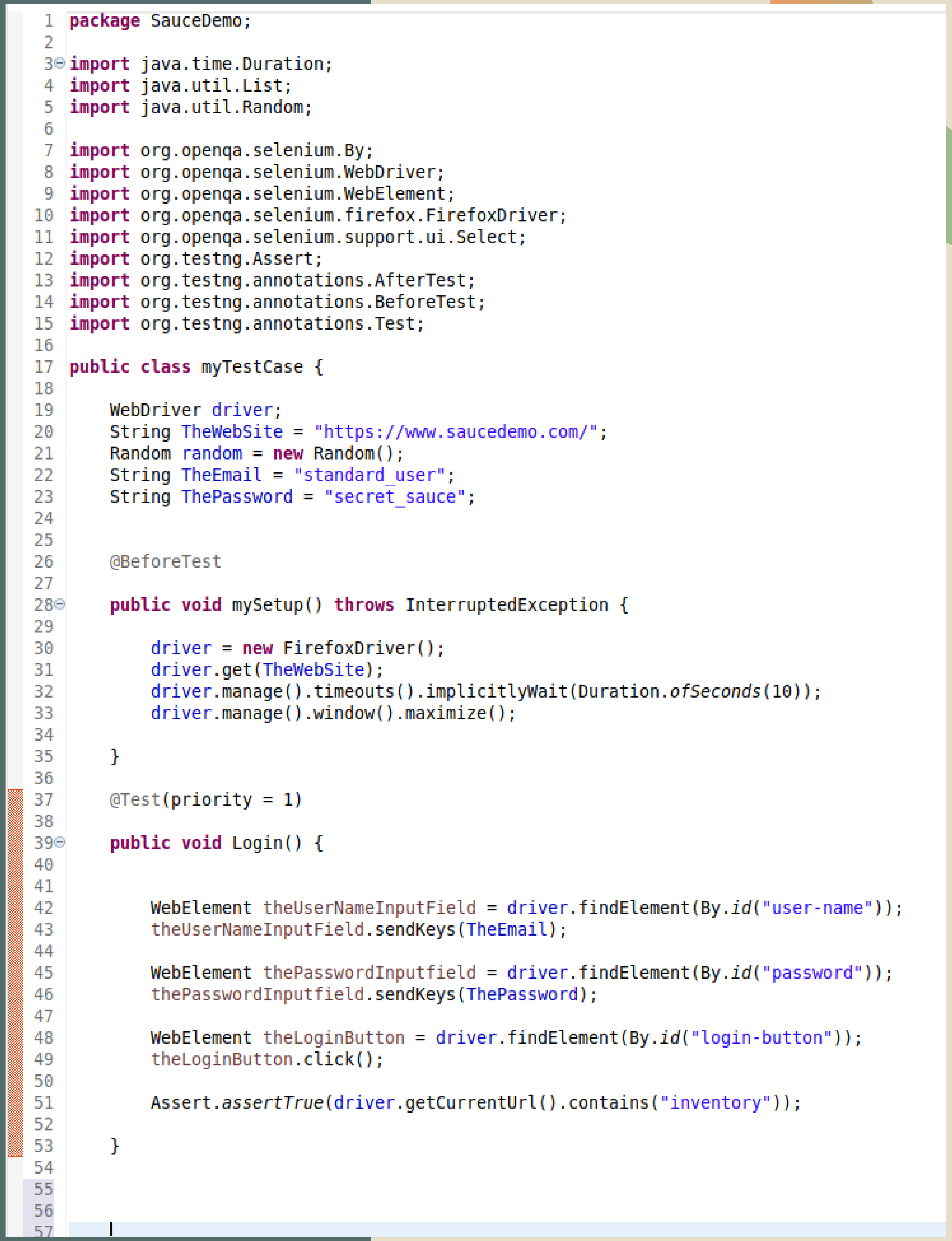
UI Automation Testing

Automation testing was implemented using Selenium WebDriver and TestNG.

Automated test scripts were created to simulate real user actions, including:

- Log in with valid and invalid credentials
- Product sorting
- Adding and removing items from the cart
- Checkout process and logout

Tests were executed in a controlled environment, and assertions were added to validate page navigation, displayed messages, and expected application behavior.



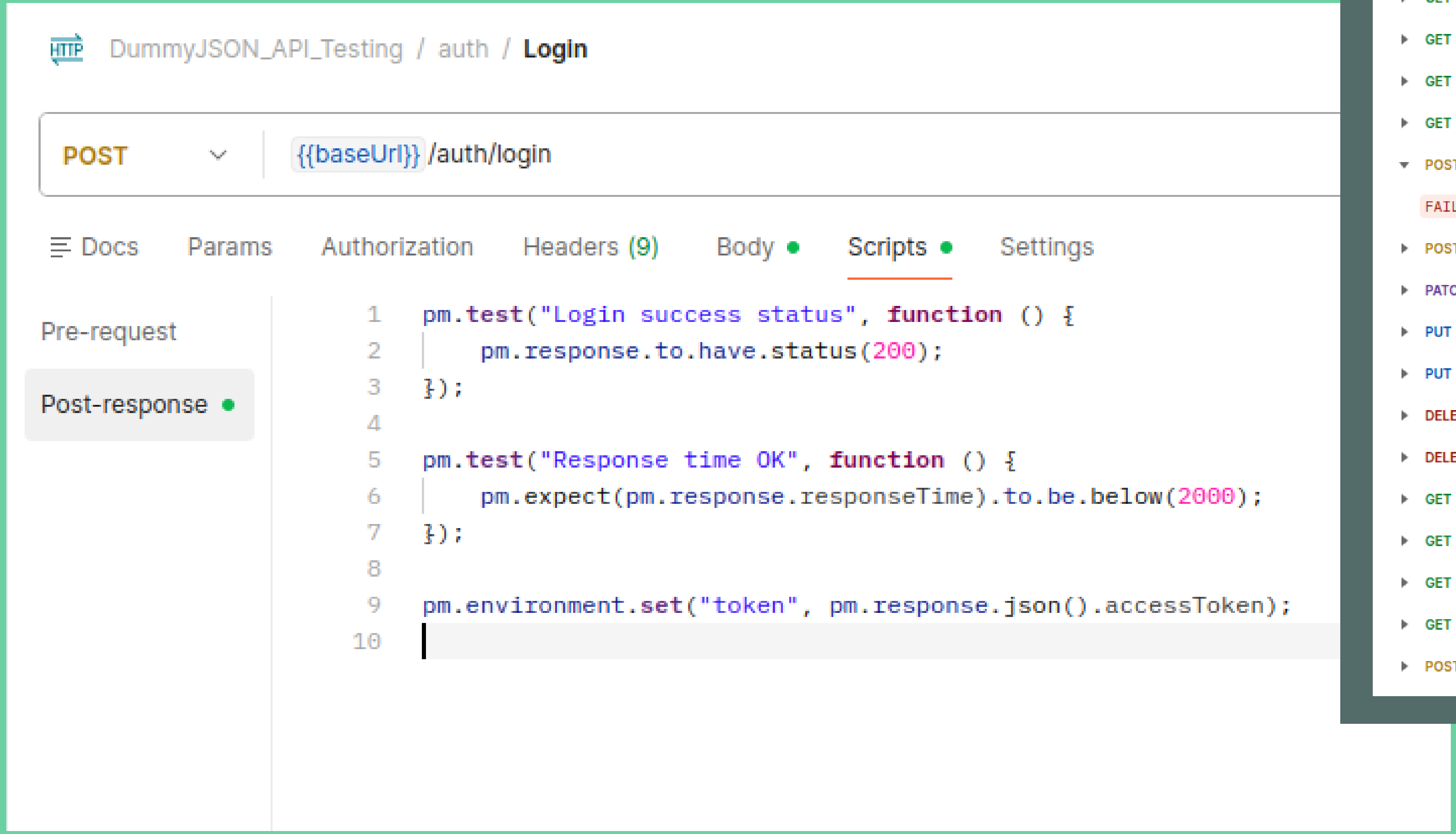
```java
package SauceDemo;

import java.time.Duration;
import java.util.List;
import java.util.Random;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;
import org.testng.Assert;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class myTestCase {

    WebDriver driver;
    String TheWebSite = "https://www.saucedemo.com/";
    Random random = new Random();
    String TheEmail = "standard_user";
    String ThePassword = "secret_sauce";


    @BeforeTest

    public void mySetup() throws InterruptedException {

        driver = new FirefoxDriver();
        driver.get(TheWebSite);
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
        driver.manage().window().maximize();

    }

    @Test(priority = 1)

    public void Login() {


        WebElement theUserNameInputField = driver.findElement(By.id("user-name"));
        theUserNameInputField.sendKeys(TheEmail);

        WebElement thePasswordInputfield = driver.findElement(By.id("password"));
        thePasswordInputfield.sendKeys(ThePassword);

        WebElement theLoginButton = driver.findElement(By.id("login-button"));
        theLoginButton.click();

        Assert.assertTrue(driver.getCurrentUrl().contains("inventory"));

    }
```

API functional testing was executed using Postman.
Test requests were created and executed to cover:

- Authentication using valid and invalid credentials
- Retrieving product and user data
- Cart-related operations

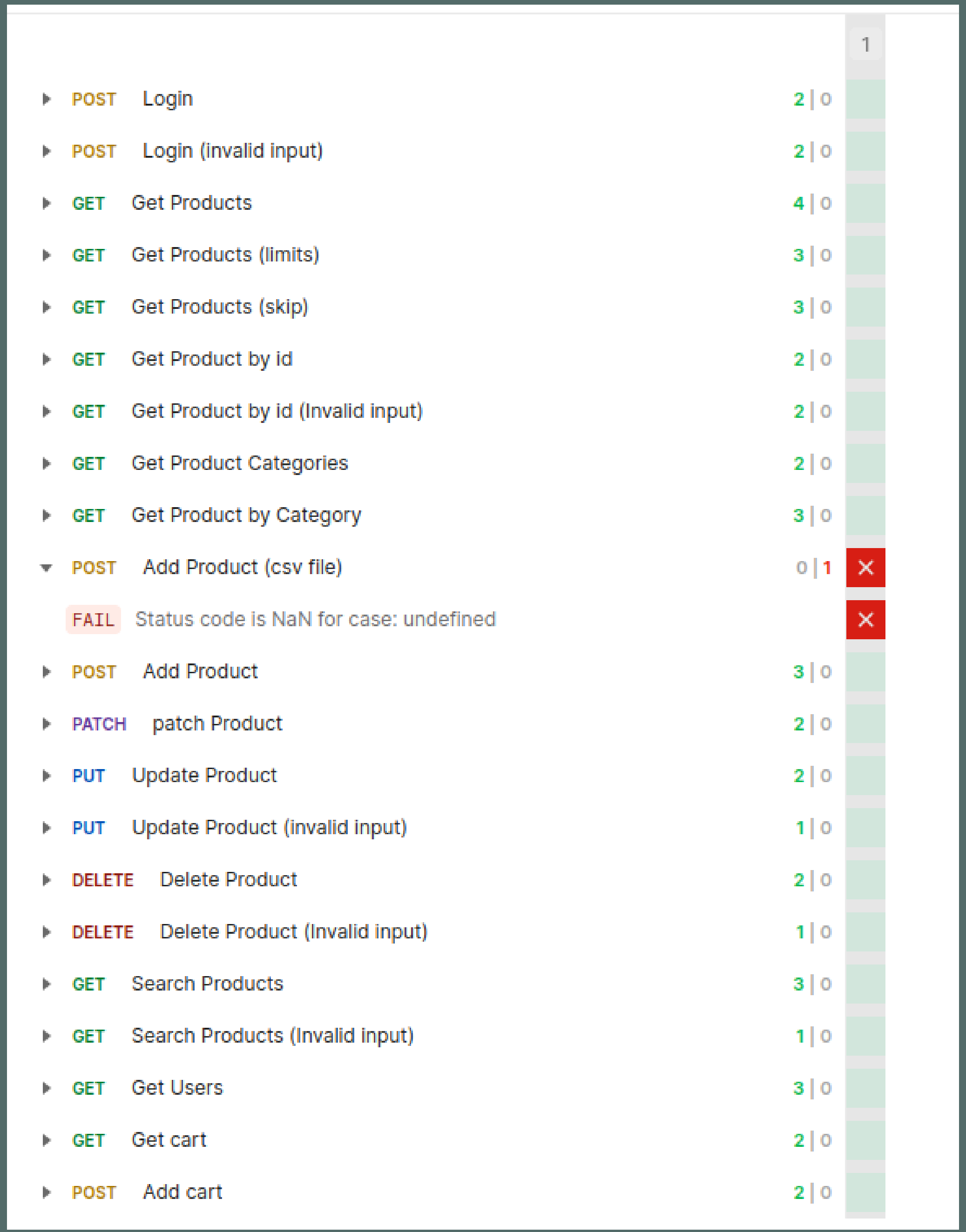
Environment variables and authentication tokens were used to support dynamic execution.
Automated assertions were added to validate responses, ensuring correct API behavior for both positive and negative scenarios.

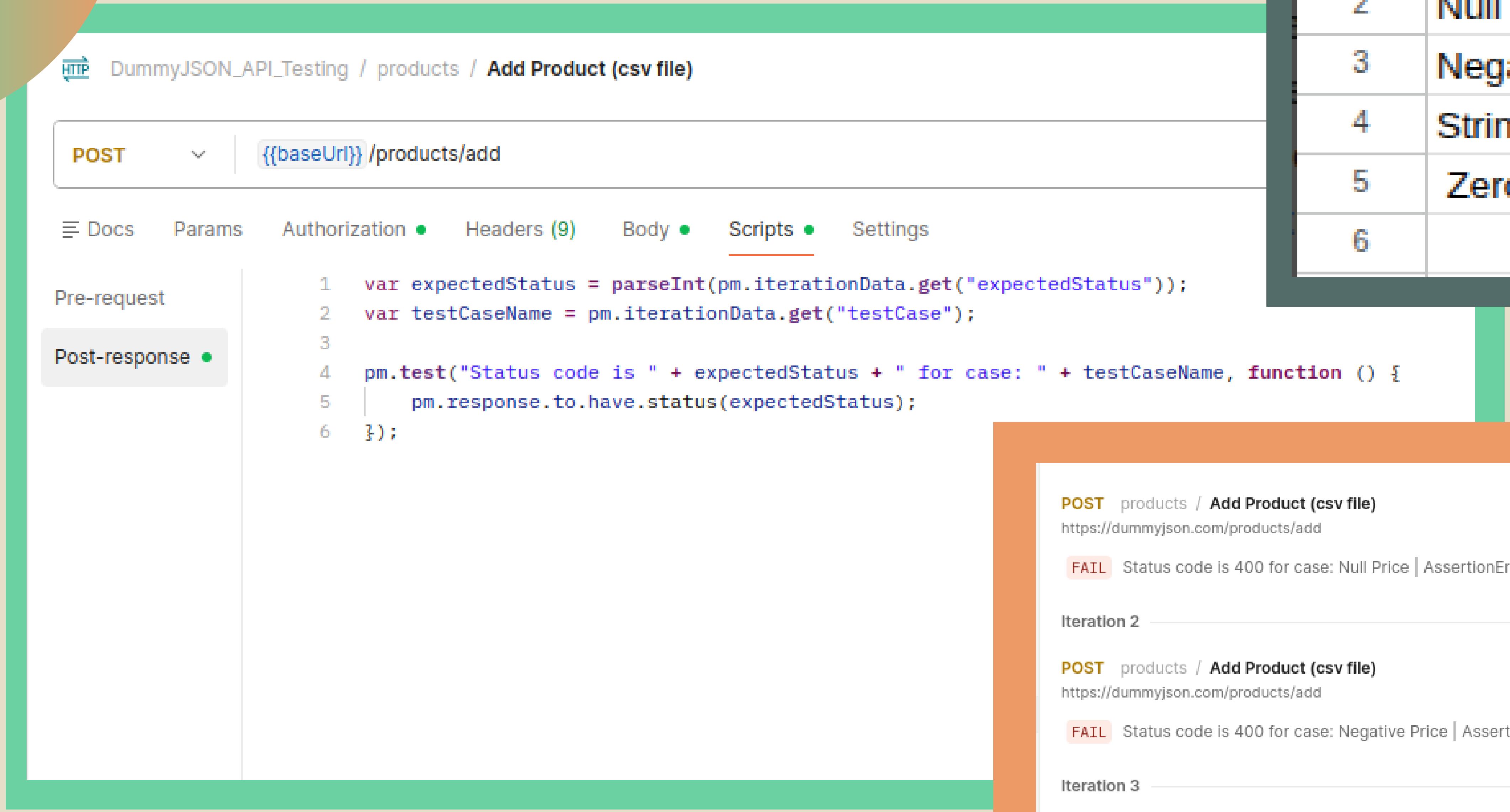
API Testing


DummyJSON_API_Testing / auth / **Login**

POST {{baseUrl}} /auth/login

```
pm.test("Login success status", function () {
    pm.response.to.have.status(200);
});

pm.test("Response time OK", function () {
    pm.expect(pm.response.responseTime).to.be.below(2000);
});

pm.environment.set("token", pm.response.json().accessToken);
```

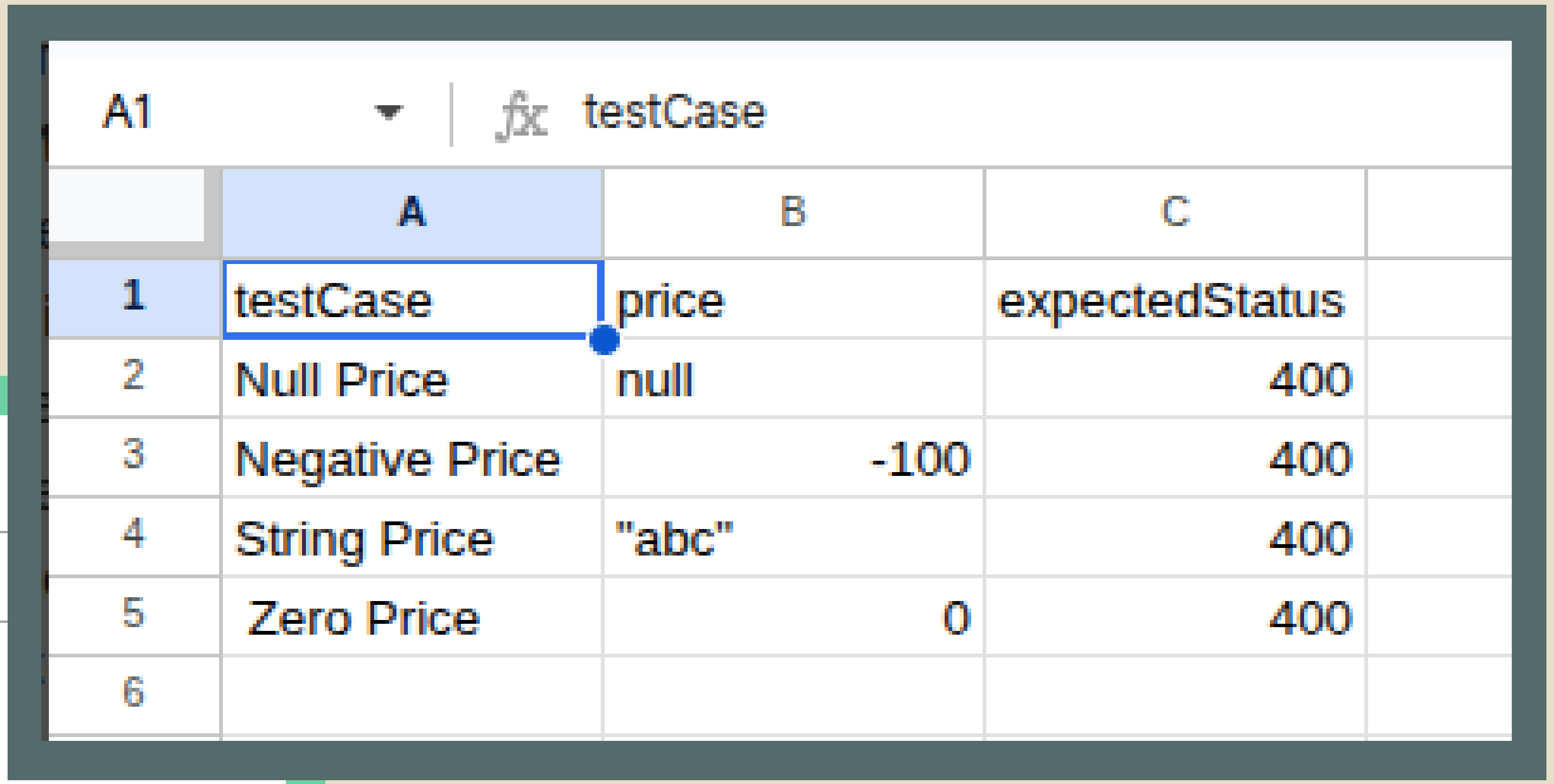

```
1
POST    Login                               2 | 0
POST    Login (invalid input)               2 | 0
GET     Get Products                        4 | 0
GET     Get Products (limits)               3 | 0
GET     Get Products (skip)                 3 | 0
GET     Get Product by id                   2 | 0
GET     Get Product by id (Invalid input)   2 | 0
GET     Get Product Categories              2 | 0
GET     Get Product by Category             3 | 0
POST    Add Product (csv file)              0 | 1
FAIL    Status code is NaN for case: undefined
POST    Add Product                         3 | 0
PATCH   patch Product                       2 | 0
PUT     Update Product                      2 | 0
PUT     Update Product (invalid input)      1 | 0
DELETE  Delete Product                      2 | 0
DELETE  Delete Product (Invalid input)      1 | 0
GET     Search Products                     3 | 0
GET     Search Products (Invalid input)     1 | 0
GET     Get Users                           3 | 0
GET     Get cart                            2 | 0
POST    Add cart                            2 | 0
```



DummyJSON_API_Testing / products / **Add Product (csv file)**

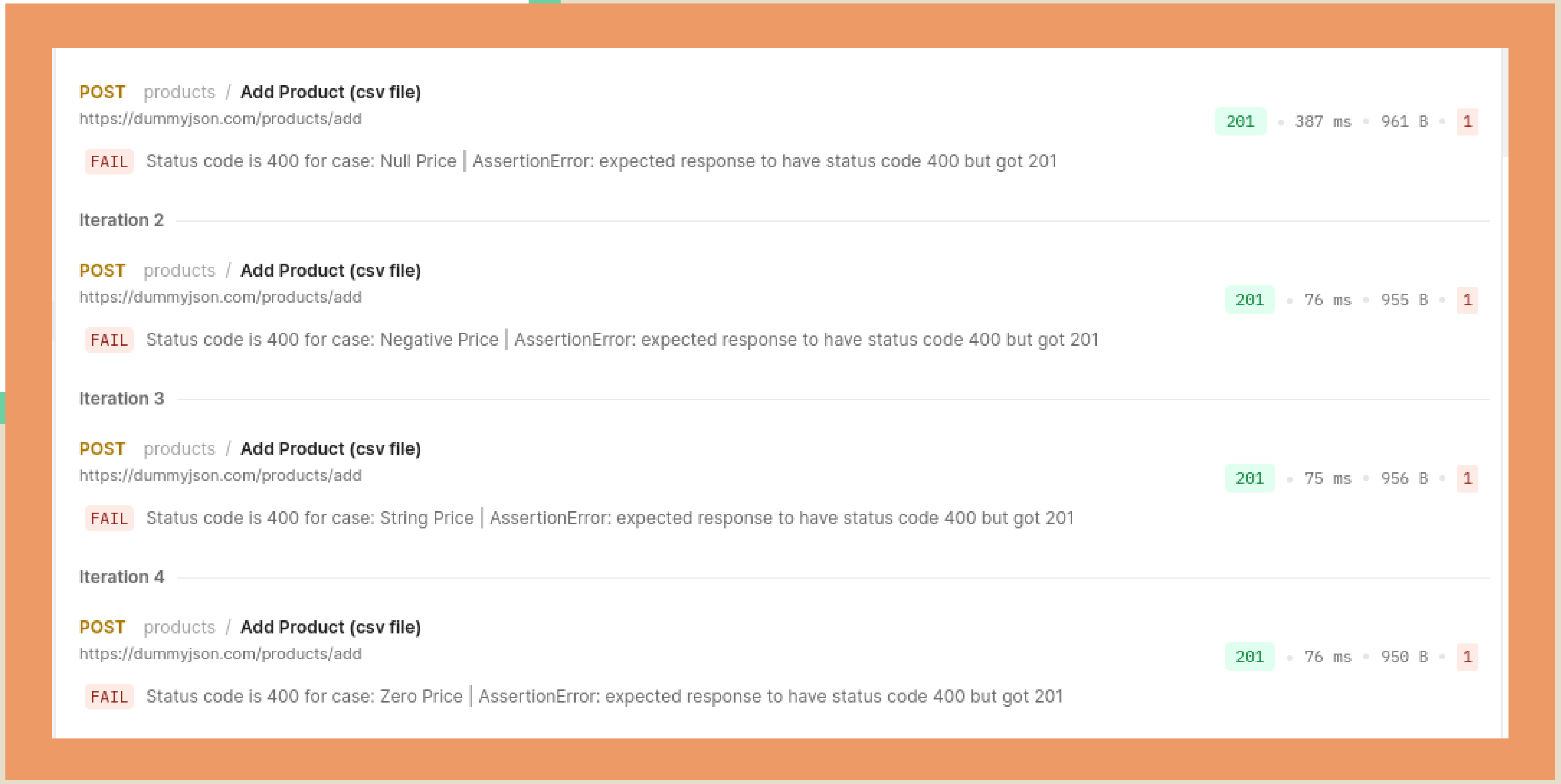
POST {{baseUrl}} /products/add

```
var expectedStatus = parseInt(pm.iterationData.get("expectedStatus"));
var testCaseName = pm.iterationData.get("testCase");

pm.test("Status code is " + expectedStatus + " for case: " + testCaseName, function () {
    pm.response.to.have.status(expectedStatus);
});
```

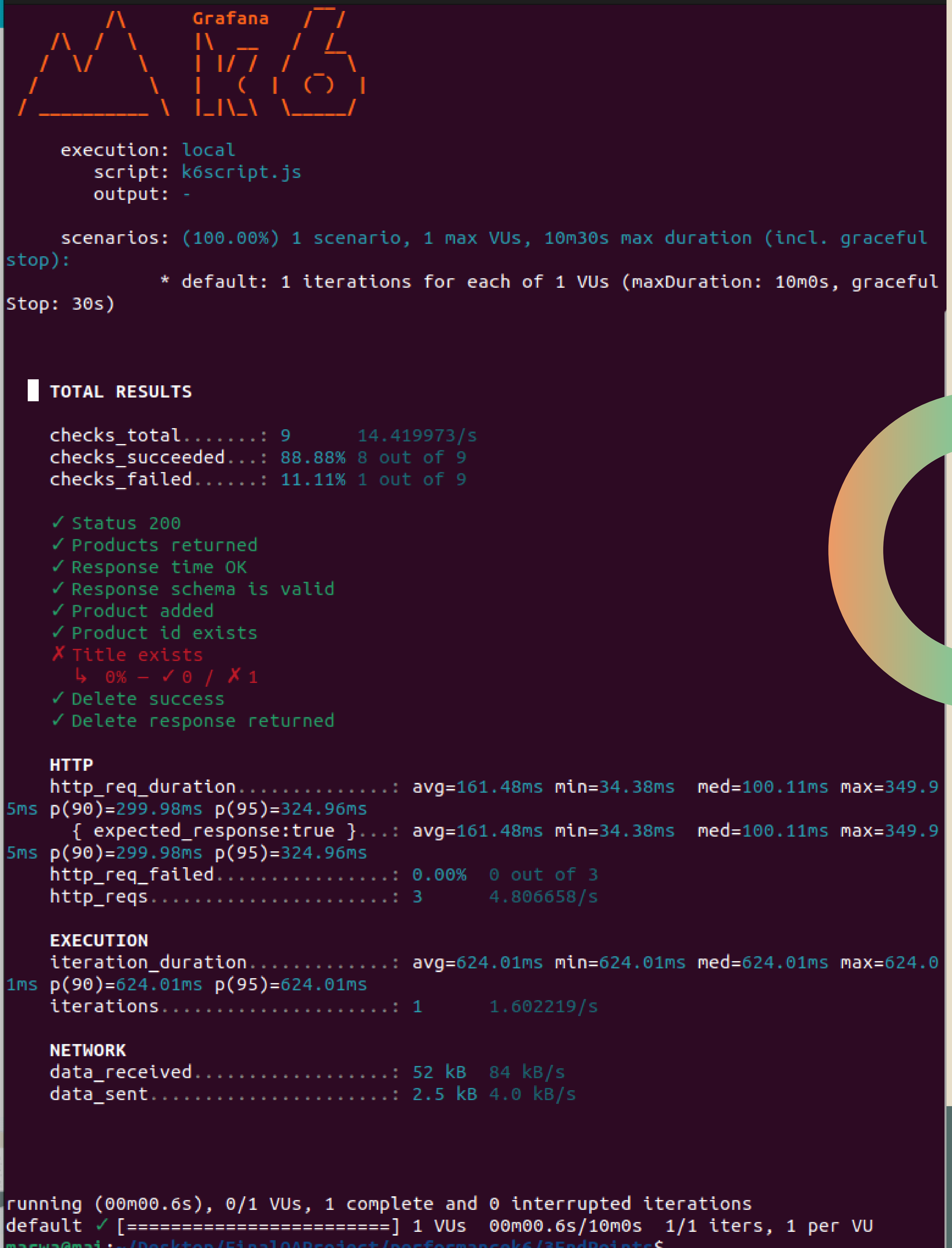


A1 fx testCase

	A	B	C
1	testCase	price	expectedStatus
2	Null Price	null	400
3	Negative Price	-100	400
4	String Price	"abc"	400
5	Zero Price	0	400
6			



```
POST  products / Add Product (csv file)
https://dummyjson.com/products/add                                   201 • 387 ms • 961 B • 1
FAIL  Status code is 400 for case: Null Price | AssertionError: expected response to have status code 400 but got 201

Iteration 2
POST  products / Add Product (csv file)
https://dummyjson.com/products/add                                   201 • 76 ms • 955 B • 1
FAIL  Status code is 400 for case: Negative Price | AssertionError: expected response to have status code 400 but got 201

Iteration 3
POST  products / Add Product (csv file)
https://dummyjson.com/products/add                                   201 • 75 ms • 956 B • 1
FAIL  Status code is 400 for case: String Price | AssertionError: expected response to have status code 400 but got 201

Iteration 4
POST  products / Add Product (csv file)
https://dummyjson.com/products/add                                   201 • 76 ms • 950 B • 1
FAIL  Status code is 400 for case: Zero Price | AssertionError: expected response to have status code 400 but got 201
```



```
     execution: local
        script: k6script.js
        output: -

     scenarios: (100.00%) 1 scenario, 1 max VUs, 10m30s max duration (incl. graceful
stop):
              * default: 1 iterations for each of 1 VUs (maxDuration: 10m0s, graceful
Stop: 30s)



  █ TOTAL RESULTS

    checks_total.......: 9       14.419973/s
    checks_succeeded...: 88.88% 8 out of 9
    checks_failed......: 11.11% 1 out of 9

    ✓ Status 200
    ✓ Products returned
    ✓ Response time OK
    ✓ Response schema is valid
    ✓ Product added
    ✓ Product id exists
    ✗ Title exists
       ↳  0% — ✓ 0 / ✗ 1
    ✓ Delete success
    ✓ Delete response returned

    HTTP
    http_req_duration..............: avg=161.48ms min=34.38ms  med=100.11ms max=349.9
5ms p(90)=299.98ms p(95)=324.96ms
      { expected_response:true }...: avg=161.48ms min=34.38ms  med=100.11ms max=349.9
5ms p(90)=299.98ms p(95)=324.96ms
    http_req_failed................: 0.00%  0 out of 3
    http_reqs......................: 3      4.806658/s

    EXECUTION
    iteration_duration.............: avg=624.01ms min=624.01ms med=624.01ms max=624.0
1ms p(90)=624.01ms p(95)=624.01ms
    iterations.....................: 1      1.602219/s

    NETWORK
    data_received..................: 52 kB  84 kB/s
    data_sent......................: 2.5 kB 4.0 kB/s




running (00m00.6s), 0/1 VUs, 1 complete and 0 interrupted iterations
default ✓ [======================] 1 VUs  00m00.6s/10m0s  1/1 iters, 1 per VU
marwa@mai:~/Desktop/FinalQAProject/performancek6/3EndPoints$
```

Performance Testing

API performance testing was executed using k6.
Two test profiles were implemented:

- Smoke testing to verify basic system availability
- Load testing to simulate concurrent users

Ramp-up and think-time strategies were applied to simulate realistic user behavior.
Execution results were analyzed to identify performance behavior and potential bottlenecks.

Thank You