SauceDemo Automation Testing – Portfolio Document

Prepared by: Deepika S

Role: QA Automation Tester (Portfolio Project)

1. Project Overview

Project Name: SauceDemo E2E Automation Testing

Tool Used: Selenium with Python, PyTest

Framework Type: Page Object Model (POM)

Test Type: Functional & End-to-End UI Automation

Application Under Test (AUT): https://www.saucedemo.com/

2. Objective

To automate a complete purchase flow on SauceDemo, starting from login → adding product → checkout → verifying successful order completion.

3. Tools & Technologies

- Programming Language: Python
- Automation Tool: Selenium WebDriver
- Test Framework: Pytest
- Reporting: pytest-html
- Browser: Chrome
- Design Pattern: Page Object Model (POM)

4. Test Flow Diagram (Optional)

Login → Add to Cart → View Cart → Checkout Info → Order Confirmation

5. Project Structure

SauceDemo_Automation/

```
|
├── Pages/
| ├── base_page.py
| ├── login_page.py
| ├── inventory_page.py
| ├── cart_page.py
| └── checkout_page.py
|
├── Tests/
| └── test_saucedemo_e2e.py
|
├── Reports/
| ├── report.html
| └── checkout_success.png
```

6. Test Scenarios

Test Case ID	Scenario Description	Expected Result
TC001	Verify successful login	User navigates to inventory page
TC002	Add product to cart	Product added successfully
TC003	Proceed to checkout	Checkout page opens
TC004	Fill checkout info	Info accepted and continued
TC005	Complete order	“THANK YOU” message displayed

7. Simulated Bug Example

Simulated Bug Example – SauceDemo Application

Field	Details
Bug Title	Incorrect Cart Badge Count After Adding Items
Severity	Medium
Priority	High
Tested URL	https://www.saucedemo.com/

Steps to Reproduce	1.Loginwith `standard_user / secret_sauce` 2.Add **2 products** to cart 3.Click the cart icon
Expected Result	Cart badge should display “2” items.
Actual Result	Cart badge displays “1” instead of “2”.
Screenshot	*cart_bug_simulation.png*
Status	*Simulated Bug – Added for demonstration of real QA reporting.*

Description:

This simulated bug highlights the QA tester’s ability to catch UI-to-logic mismatches in e-commerce workflows. It emphasizes the verification of cart count accuracy, a common defect in web applications.

7. Script Implementation

Base Page

```
from selenium.webdriver.support.ui import WebDriverWait from

selenium.webdriver.support import expected_conditions as EC



class BasePage:

  def __init__(self, driver): self.driver =

    driver self.wait =

    WebDriverWait(driver, 30) def

    open(self, url):


    self.driver.get(url)
  def click(self, locator): element =
    self.wait.until(EC.element_to_be_clickable(locator))
    element.click()


  def type(self, locator, text):
    element =
    self.wait.until(EC.visibility_of_element_located(locator))
    element.clear() element.send_keys(text)


  def        get_text(self,        locator):        element        =
    self.wait.until(EC.visibility_of_element_located(locator))     return
    element.text
```

Login Page

```
from selenium.webdriver.common.by import By
from pages.base_page_saucedemo import BasePage


class LoginPage(BasePage):
  USERNAME = (By.ID, "user-name")
  PASSWORD = (By.ID, "password")
  LOGIN_BUTTON = (By.ID, "login-button")
  def open_login_page(self):
    self.open("https://www.saucedemo.com/")
  def login(self, username, password):
    self.type(self.USERNAME, username)
    self.type(self.PASSWORD, password)
    self.click(self.LOGIN_BUTTON)
```

Inventory Page

```
from selenium.webdriver.common.by import By from
selenium.webdriver.support.ui import WebDriverWait from
selenium.webdriver.support import expected_conditions as EC
from pages.base_page_saucedemo import BasePage



class InventoryPage(BasePage):
  ADD_TO_CART_BUTTON = (By.ID, "add-to-cart-sauce-labs-backpack")
  CART_ICON = (By.CLASS_NAME, "shopping_cart_link")
  INVENTORY_CONTAINER = (By.ID, "inventory_container")


  def wait_for_inventory_page(self):
    WebDriverWait(self.driver, 10).until(
      EC.visibility_of_element_located(self.INVENTORY_CONTAINER)
    )

  def add_to_cart(self):
    print("Trying to click Add to Cart button")
    WebDriverWait(self.driver, 10).until(
    EC.element_to_be_clickable(self.ADD_TO_CART_BUTTON)

    ).click()
  def go_to_cart(self):

    print("Trying to click Cart icon") cart_icon =

    WebDriverWait(self.driver, 10).until(

      EC.element_to_be_clickable(self.CART_ICON)

    )

    self.driver.execute_script("arguments[0].click();", cart_icon)

    print("Cart icon clicked via JavaScript")
```

Cart Page

```
from selenium.webdriver.common.by import By from

selenium.webdriver.support.ui import WebDriverWait from

selenium.webdriver.support import expected_conditions as EC



class CartPage:

  CHECKOUT_BUTTON = (By.ID, "checkout")


  def __init__(self, driver):

    self.driver = driver


    self.wait = WebDriverWait(driver, 30)


  def proceed_to_checkout(self):

    print("Clicking Checkout button safely...")

    # Wait until button is clickable


    checkout_btn = self.wait.until(EC.element_to_be_clickable(self.CHECKOUT_BUTTON))
    # JS click ensures any JS listener works
```

```
        self.driver.execute_script("arguments[0].click();", checkout_btn) #

        Wait until URL changes to checkout-step-one.html

        self.wait.until(lambda d: "checkout-step-one.html" in

        d.current_url) print("Checkout page navigation triggered")
```

Checkout Page

```
from selenium.webdriver.common.by import By from

selenium.webdriver.support.ui import WebDriverWait from

selenium.webdriver.support import expected_conditions as EC

import time



class CheckoutPage:

    FIRST_NAME = (By.ID, "first-name")

    LAST_NAME = (By.ID, "last-name")

    POSTAL_CODE = (By.ID, "postal-code")

    CONTINUE_BUTTON = (By.ID, "continue")

    FINISH_BUTTON = (By.ID, "finish")

    SUCCESS_TEXT = (By.CLASS_NAME, "complete-header")


    def __init__(self, driver): self.driver =

        driver self.wait =

        WebDriverWait(driver, 10)



    def _react_type(self, locator, value):

        """React-safe typing method that updates Virtual DOM state"""

        element = self.wait.until(EC.element_to_be_clickable(locator))
```

```
self.driver.execute_script("arguments[0].scrollIntoView(true);",
element) time.sleep(0.2)


# Focus field
self.driver.execute_script("arguments[0].focus();", element)
time.sleep(0.2)


# Clear any existing text
element.clear()
time.sleep(0.2)


# Set value and trigger React synthetic event
self.driver.execute_script("""
  const [el, val] = arguments;
  const lastVal = el.value;
  el.value = val;
  const event = new Event('input', { bubbles: true });
  event.simulated = true;


  // React 17+ internal event tracker
  const tracker = el._valueTracker; if
  (tracker) tracker.setValue(lastVal);


  el.dispatchEvent(event);
```

```
    """, element, value)
    time.sleep(0.4)

    self.driver.execute_script("arguments[0].blur();", element)

    time.sleep(0.3)



def fill_checkout_info(self, first_name, last_name, postal_code):

    print("Filling checkout info (React-safe typing)...")

    self._react_type(self.FIRST_NAME, first_name)

    self._react_type(self.LAST_NAME, last_name)

    self._react_type(self.POSTAL_CODE, postal_code)



def click_continue(self):

    print(" Clicking Continue button...")

    btn = self.wait.until(EC.element_to_be_clickable(self.CONTINUE_BUTTON))

    self.driver.execute_script("arguments[0].click();", btn) time.sleep(1)



def click_finish(self):

    print("Clicking Finish button...")

    btn = self.wait.until(EC.element_to_be_clickable(self.FINISH_BUTTON))

    self.driver.execute_script("arguments[0].click();", btn) time.sleep(1)



def verify_success(self):

    print(" Verifying success message...")

    success = self.wait.until(EC.visibility_of_element_located(self.SUCCESS_TEXT))

    assert "THANK YOU" in success.text.upper(), "Checkout failed"
```

```
        print(" Checkout successful!")


        self.driver.save_screenshot("checkout_success.png")
        print("Screenshot saved as checkout_success.png")
```

Test Script (Pytest)

```
import    time    import
pytest

from selenium.webdriver.common.by import By from
selenium.common.exceptions import NoSuchElementException
from pages.login_page_saucedemo import LoginPage from
pages.inventory_page_saucedemo import InventoryPage from
pages.checkout_page_saucedemo import CheckoutPage from
pages.cart_page_saucedemo import CartPage

from utils.screenshot import take_screenshot # Reuse your screenshot utility



# POSITIVE SCENARIO – Valid Login and End-to-End Flow
@pytest.mark.positive
@pytest.mark.smoke
@pytest.mark.regression def
test_saucedemo_e2e(chrome_driver):
driver = chrome_driver


  login_page = LoginPage(driver)
  inventory_page = InventoryPage(driver)
  cart_page = CartPage(driver) checkout_page =
  CheckoutPage(driver)


  # ---- Login ---login_page.open_login_page()
  login_page.login("standard_user", "secret_sauce")
  take_screenshot(driver, "valid_login_success")


  # ---- Inventory ---
  inventory_page.wait_for_inventory_page()
  inventory_page.add_to_cart()
  inventory_page.go_to_cart()
  take_screenshot(driver, "item_added_to_cart")


  # ---- Cart ---cart_page.proceed_to_checkout()
  take_screenshot(driver, "proceed_to_checkout")


  # ---- Checkout ----
  checkout_page.fill_checkout_info("Deepika", "S", "600100")
  checkout_page.click_continue()
  checkout_page.click_finish()
  checkout_page.verify_success() take_screenshot(driver,
  "order_completed")
# NEGATIVE SCENARIO – Missing Username Validation
```

```
@pytest.mark.negative def
test_missing_username_validation(chrome_driver):
driver = chrome_driver


 # Step 1: Open login page login_page =
 LoginPage(driver)
 login_page.open_login_page()
 take_screenshot(driver, "login_page_opened")


 # Step 2: Enter only password (no username)
 password_box = driver.find_element(By.ID, "password")
 password_box.send_keys("secret_sauce")
 take_screenshot(driver, "entered_password_only")


 # Step 3: Click Login button
 login_button = driver.find_element(By.ID, "login-button")
 login_button.click() time.sleep(2)

 take_screenshot(driver, "clicked_login_without_username")


 # Step 4: Capture and verify the error message
 try:
   error_message = driver.find_element(By.CSS_SELECTOR, "h3[data-test='error']").text
   print("Displayed Error:", error_message) take_screenshot(driver,
   "missing_username_error")
   assert "Epic sadface: Username is required" in error_message, \
     "BUG FOUND: Expected validation message not displayed!" print("Validation
   working correctly — Username is required message shown.")

 except NoSuchElementException:
   take_screenshot(driver, "error_message_not_found")
   print(" BUG FOUND: No validation message displayed.")
   raise
```

8. Test Execution

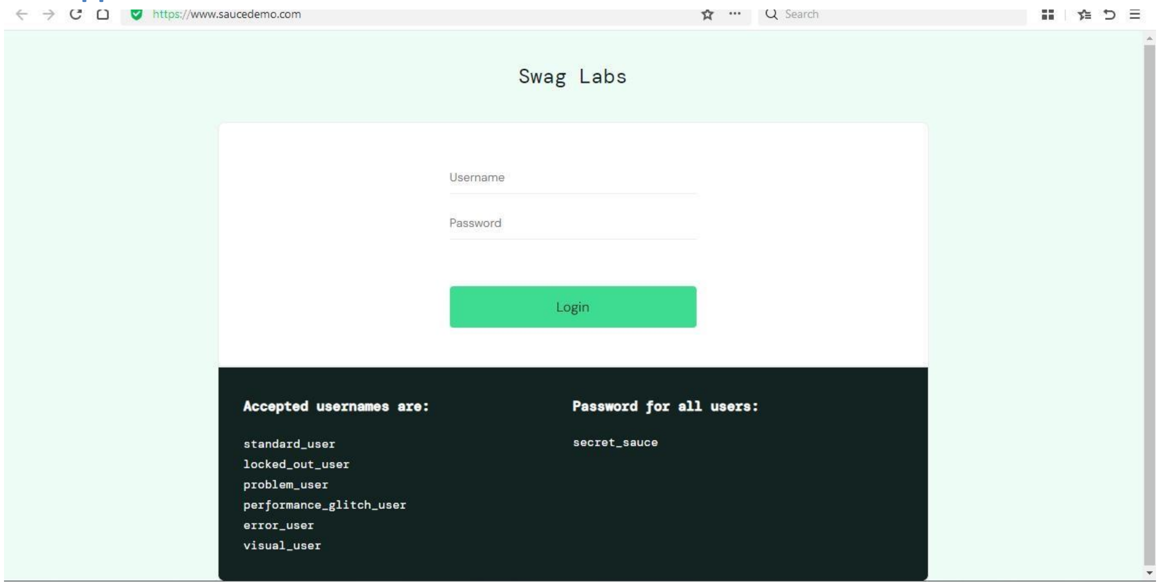
Command Used:

```
pytest -s Tests/test_saucedemo_e2e.py --html=Reports/report.html --self-
containedhtml
```

Execution Result: Passed

9. Screenshots

9.1. Application Screenshots



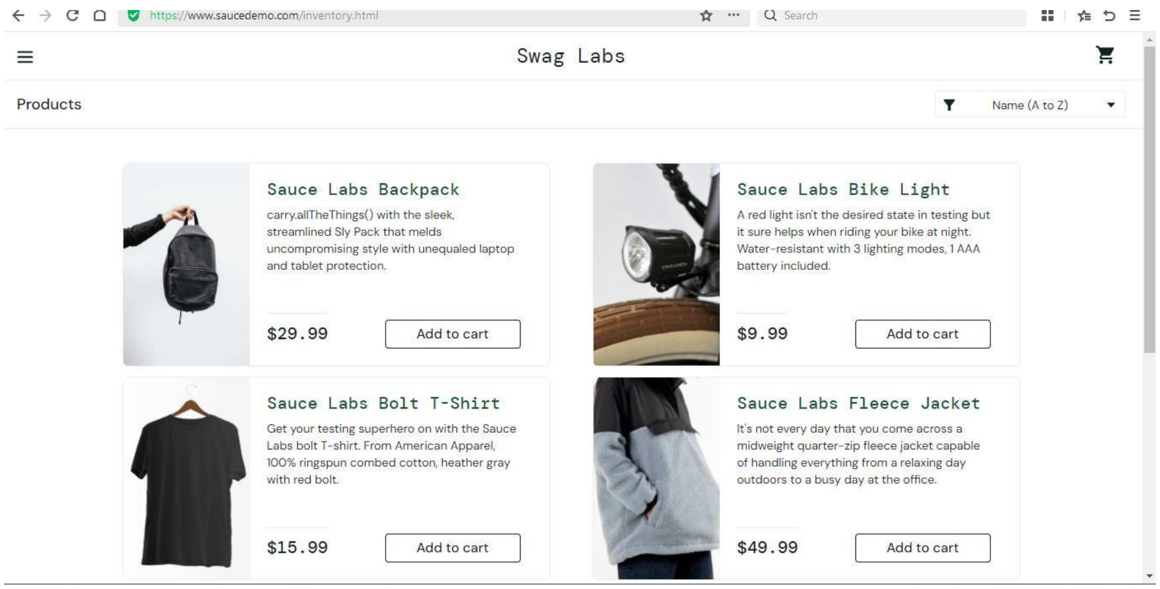
Accepted usernames are:

standard_user
locked_out_user
problem_user
performance_glitch_user
error_user
visual_user

Password for all users:

secret_sauce

Figure 1: *Login Page – SauceDemo Application*



Sauce Labs Backpack

carry.allTheThings() with the sleek, streamlined Sly Pack that melds uncompromising style with unequaled laptop and tablet protection.

$29.99

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

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

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

Figure 2: Inventory Page – Product Added to Cart

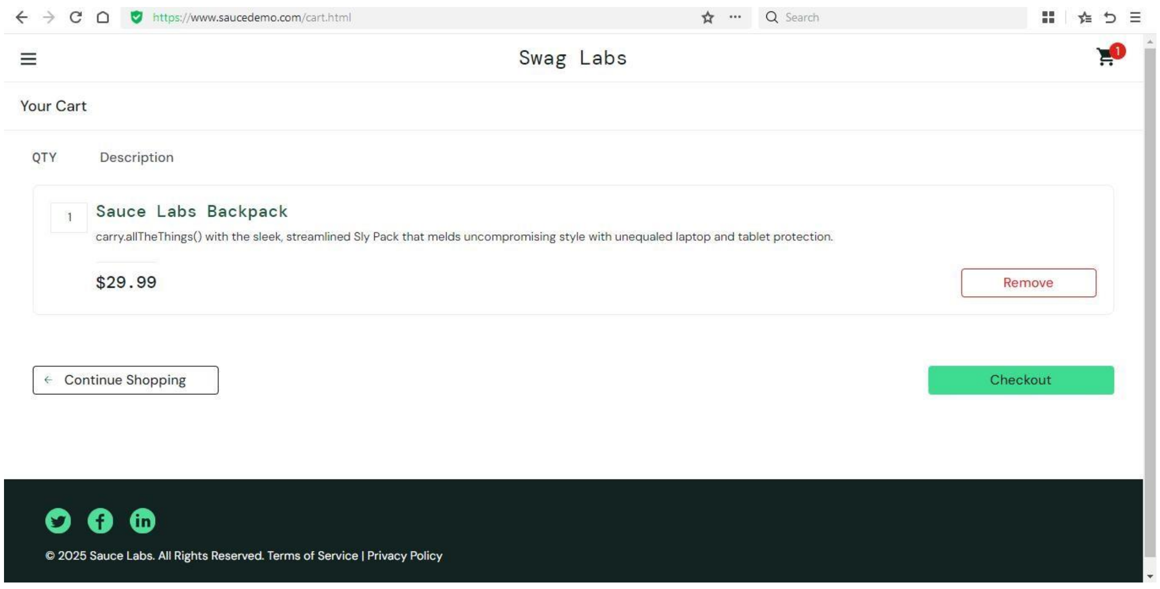

https://www.saucedemo.com/cart.html
Search
Swag Labs
1
Your Cart
QTY
Description
1
Sauce Labs Backpack
carry.allTheThings() with the sleek, streamlined Sly Pack that melds uncompromising style with unequaled laptop and tablet protection.
$29.99
Remove
Continue Shopping
Checkout
© 2025 Sauce Labs. All Rights Reserved. Terms of Service | Privacy Policy

Figure 3: Cart Page – Before Checkout

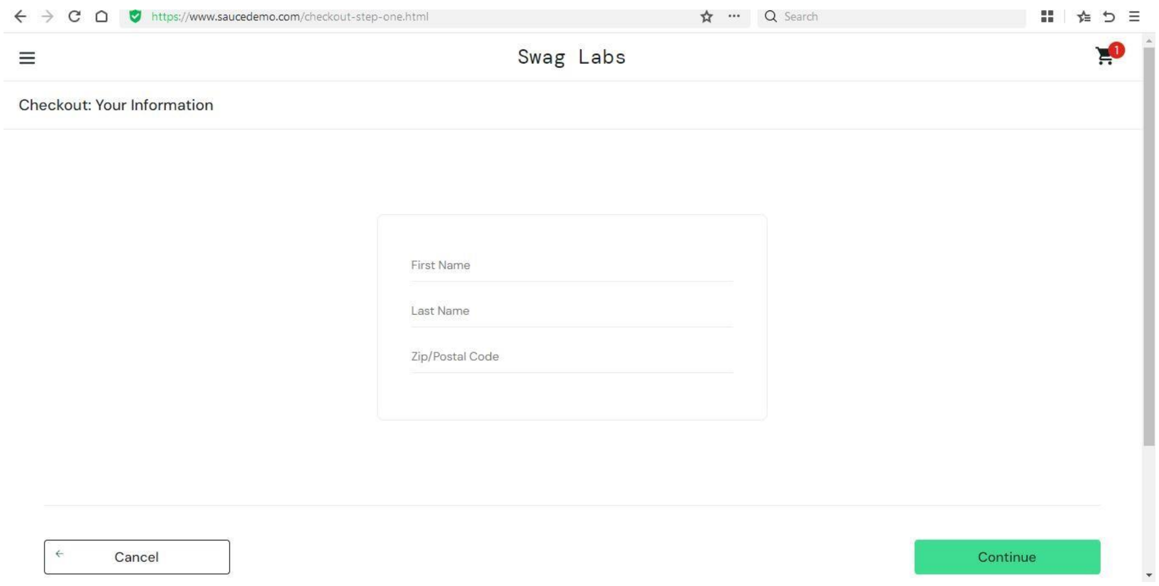
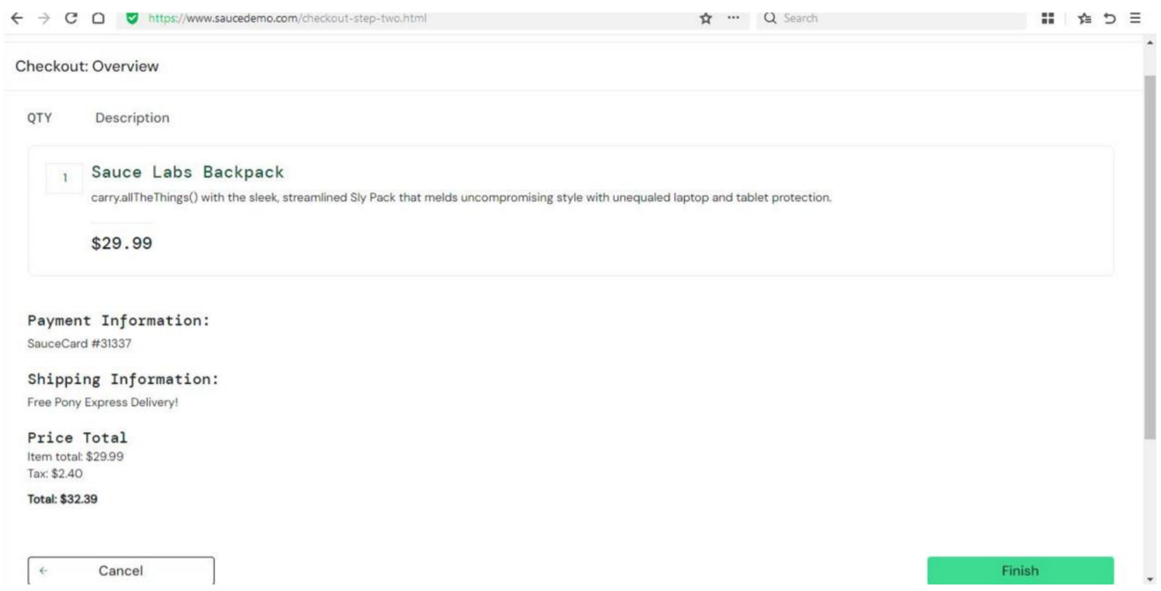

https://www.saucedemo.com/checkout-step-one.html
Search
Swag Labs
1
Checkout: Your Information
First Name
Last Name
Zip/Postal Code
Cancel
Continue

Figure 4: Checkout Information Page – User Details Entry



https://www.saucedemo.com/checkout-step-two.html

Checkout: Overview

QTY Description

1 Sauce Labs Backpack

carry.allTheThings() with the sleek, streamlined Sly Pack that melds uncompromising style with unequaled laptop and tablet protection.

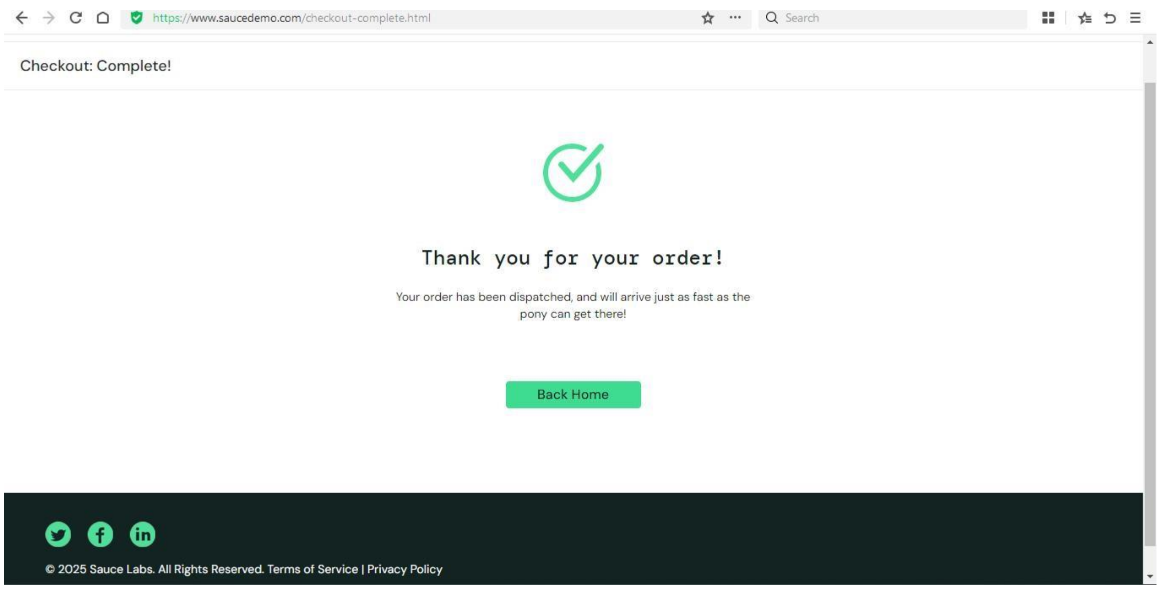
$29.99

Payment Information:
SauceCard #31337

Shipping Information:
Free Pony Express Delivery!

Price Total
Item total: $29.99
Tax: $2.40
Total: $32.39

Figure 5: *Checkout Overview Page – Order Summary*



https://www.saucedemo.com/checkout-complete.html

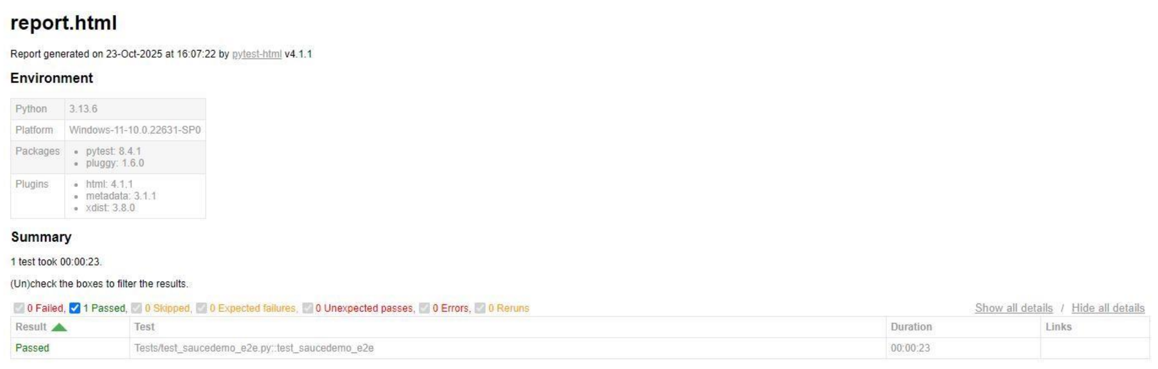
Checkout: Complete!

Thank you for your order!

Your order has been dispatched, and will arrive just as fast as the pony can get there!

© 2025 Sauce Labs. All Rights Reserved. Terms of Service | Privacy Policy

Figure 6: *Checkout Success Page – Order Completion Message*



report.html

Report generated on 23-Oct-2025 at 16:07:22 by pytest-html v4.1.1

Environment

Python	3.13.6
Platform	Windows-11-10.0.22631-SP0
Packages	• pytest: 8.4.1 • pluggy: 1.6.0
Plugins	• html: 4.1.1 • metadata: 3.1.1 • xdist: 3.8.0

Summary

1 test took 00:00:23.

(Un)check the boxes to filter the results.

☐ 0 Failed, ☑ 1 Passed, ☐ 0 Skipped, ☐ 0 Expected failures, ☐ 0 Unexpected passes, ☐ 0 Errors, ☐ 0 Reruns   Show all details / Hide all details

Result	Test	Duration	Links
Passed	Tests/test_saucedemo_e2e.py::test_saucedemo_e2e	00:00:23	

Figure 7: Pytest HTML Report – Test Execution Summary
Click here to view the full HTML test report -FINAL AUTOMATION PROJECT1\report.html

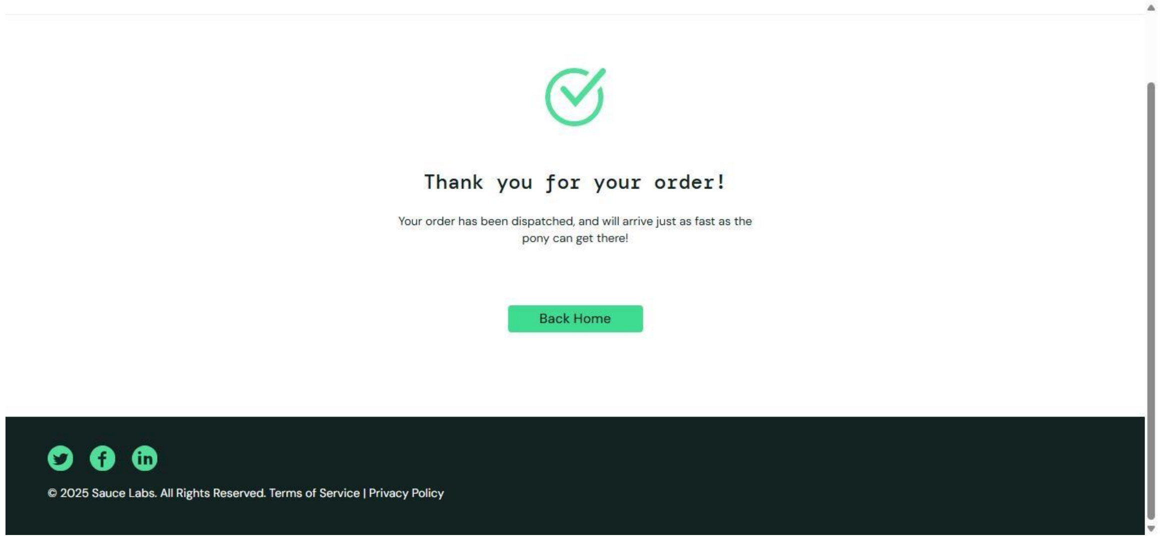

Thank you for your order!
Your order has been dispatched, and will arrive just as fast as the pony can get there!
Back Home
© 2025 Sauce Labs. All Rights Reserved. Terms of Service | Privacy Policy

Figure 8: *Automation Captured Screenshot – checkout_success.png*

Click here to view the success screenshot - FINAL AUTOMATION PROJECT1\checkout_success.png

9. Result Summary

Metric	**Value**
Total Test Cases	5
Passed	5
Failed	0
Browser	Chrome

Execution Time	~1 minutes

10. Allure Reporting (Advanced Feature)

10.1 Purpose

To enhance visualization of test execution results using the Allure Reporting Framework **— a modern dashboard that displays detailed insights such as passed/failed cases, timelines, and execution duration.**

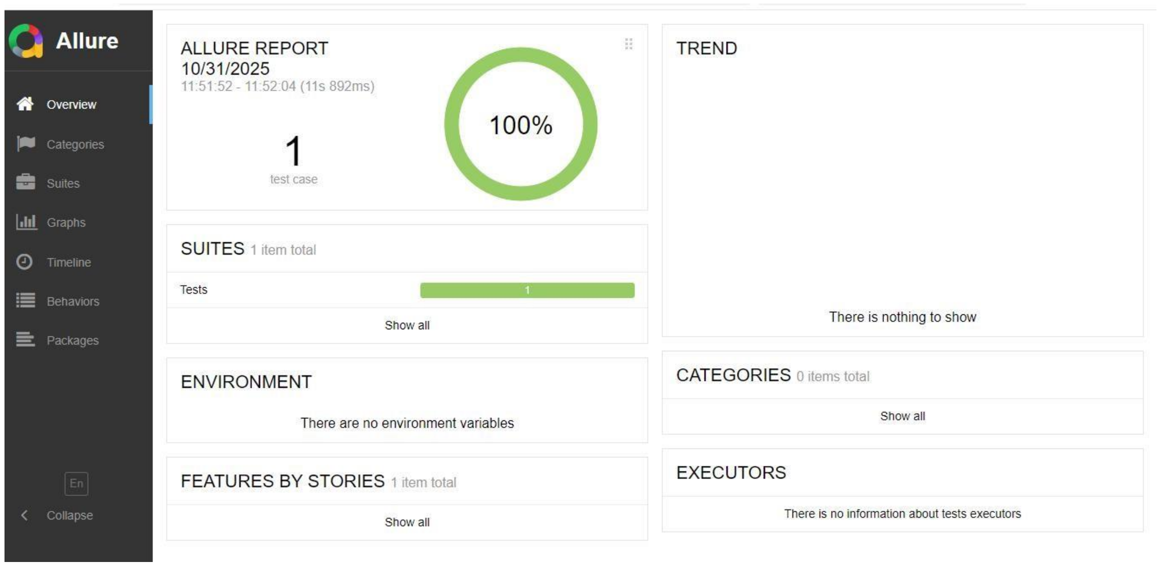
10.2 Commands Used

```
pytest -s Tests/test_saucedemo_e2e.py --alluredir=Reports/allure-results

allure generate Reports/allure-results -o Reports/allure-report --clean

allure open Reports/allure-report
```

10.3 Allure Report Screenshots



ALLURE REPORT
10/31/2025
11:51:52 - 11:52:04 (11s 892ms)
1
test case
100%

SUITES 1 item total
Tests 1

TREND
There is nothing to show

ENVIRONMENT
There are no environment variables

CATEGORIES 0 items total

FEATURES BY STORIES 1 item total

EXECUTORS
There is no information about tests executors

***Figure 9**: Allure Dashboard – Overall Test Summary*

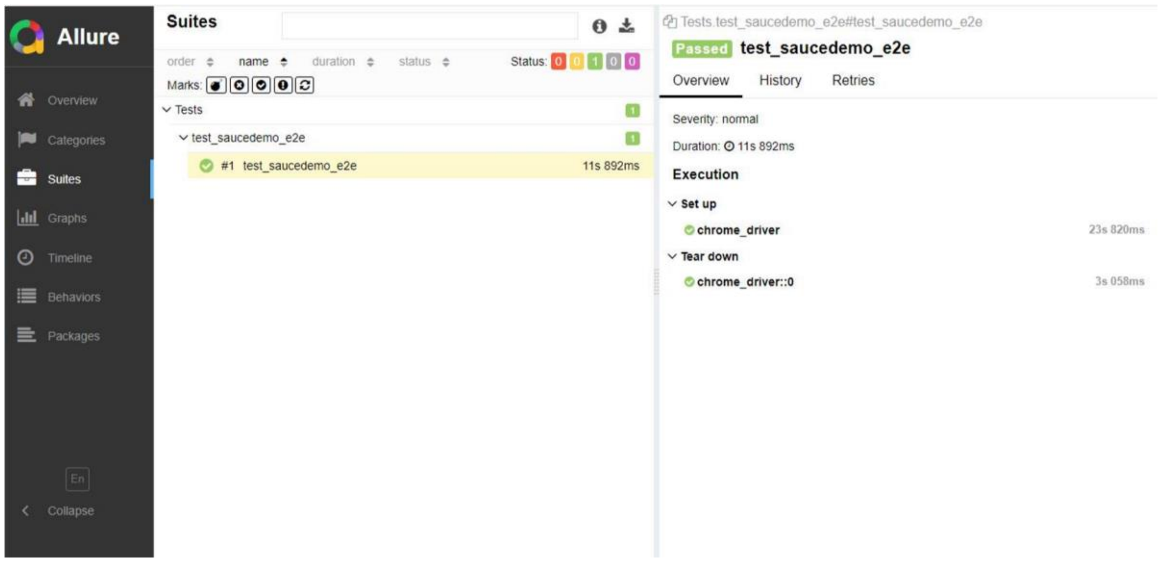


Figure 10: *Allure Suites – Test Steps and Assertions*

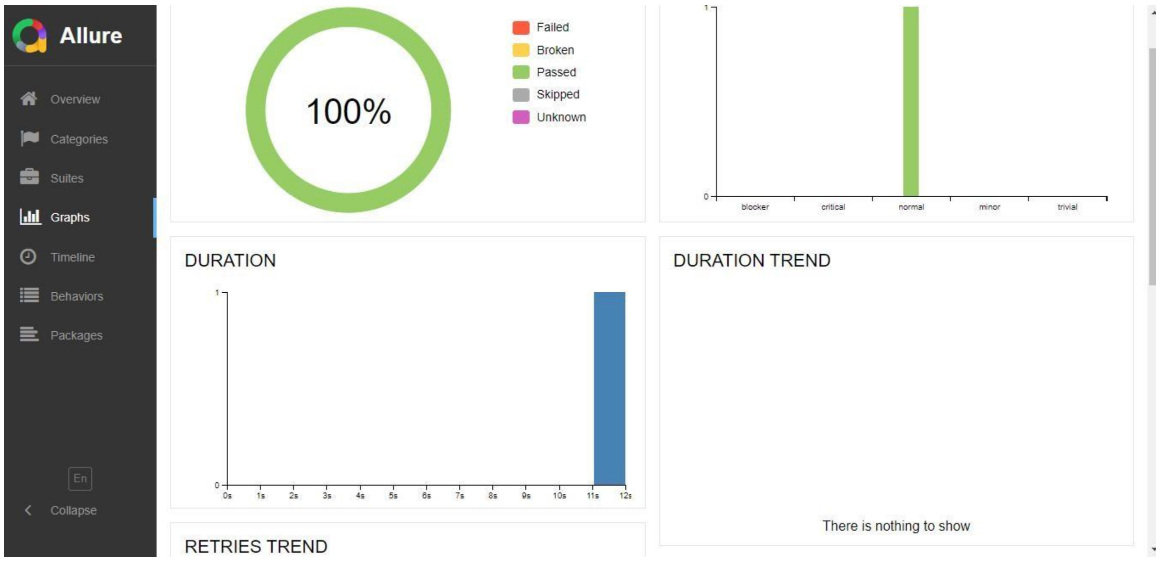


Figure 11: *Allure Graph View – Result Analytics*

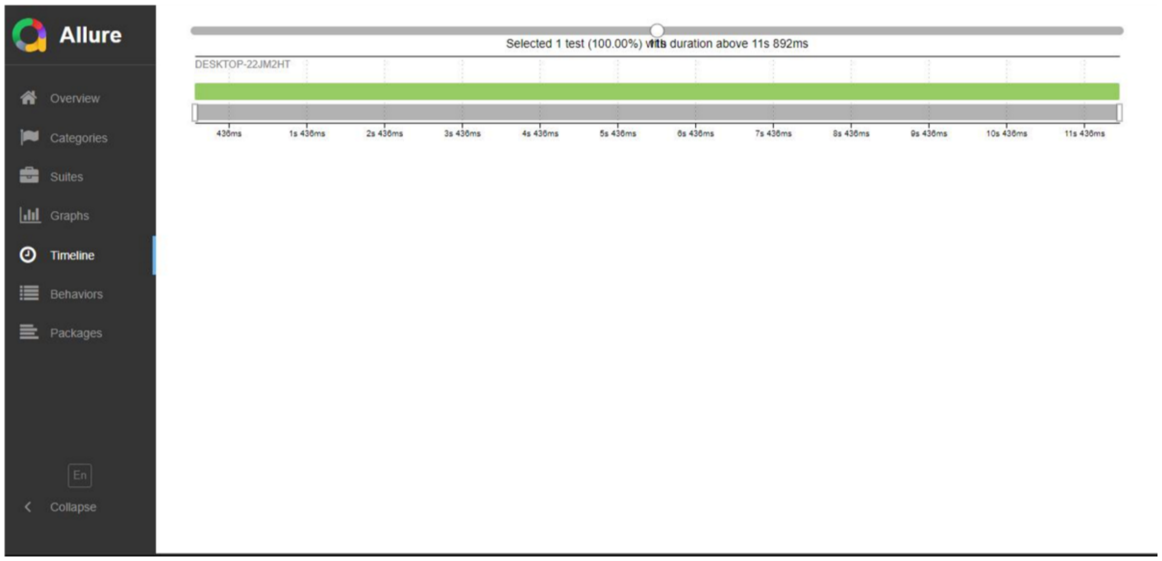


Figure 12: *Allure Timeline – Test Execution Duration*

10.7 Summary

The Allure Report successfully visualizes all automation test executions with 100% pass rate.Its interactive dashboards and detailed test breakdowns enhance the project's readability and demonstrate professional-level QA reporting skills.
These reports are now part of **Deepika S's QA Automation Portfolio** to showcase advanced reporting expertise to recruiters.

11. Conclusion

Successfully automated the entire purchase flow on SauceDemo using Selenium and Python with POM framework design.

All test cases passed and verified with screenshots and HTML report.

11. Future Integrations & Version Control Setup

✅ Integrated **Allure Reporting** for test visualization
✅ Added **requirements.txt** for one-click environment setup
✅ Captured and documented **Simulated Bug Example**
✅ Uploaded project to **GitHub (Public Repository)** for global visibility
✅ Ready for **CI/CD integration (GitHub Actions)** for continuous testing and reporting

GitHub Repository Link:

https://github.com/deepika-sekar-qa/DeepikaS_QA_Automation_Portfolio