HerokuApp Automation Testing – Portfolio Document

Prepared by: Deepika S

Role: QA Automation Tester (Trainee)

1. Project Overview

Project Name: HerokuApp Functional UI Automation
Tool Used: Selenium with Python, PyTest
Framework Type: Page Object Model (POM)
Test Type: Functional, Smoke, and Regression Testing
Application Under Test (AUT): https://the-internet.herokuapp.com/

2. Objective

To automate multiple web features such as Login, Dropdown, Alerts, and Dynamic Loading on the HerokuApp site using Selenium WebDriver with a structured Page Object Model framework.

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

Login → Dropdown → Alerts → Dynamic Loading → Result Validation → Screenshot Capture

5. Project Structure

```
HerokuApp_Automation/
│
├── pages/
│   ├── base_page.py
│   ├── home_page.py
│   ├── login_page.py
│   ├── dropdown_page.py
│   ├── alert_page.py
│   └── dynamic_loading_page.py
│
├── tests/
│   ├── test_login.py
│   └── test_app_feature.py
│
├── utils/
│   └── screenshot.py
│
├── conftest.py
└── requirements.txt
```

6. Test Scenarios

Test Case ID	Scenario Description	Expected Result
TC001	Verify valid login	Login successful message displayed
TC002	Verify invalid login	Error message displayed
TC003	Select options in dropdown	Correct option displayed
TC004	Handle all JavaScript alerts	Alerts handled correctly
TC005	Verify dynamic content load	'Hello World!' displayed

7. Simulated Bug Example

Simulated Bug Example – HerokuApp Application

Field	Details
Bug Title	Incorrect Error Message Displayed for Invalid Login
Severity	High
Priority	Medium
Tested URL	https://the-internet.herokuapp.com/login
Steps to Reproduce	1.Go to login page 2.Enter username: `tomsmith` 3.Enter password: `wrongpass` 4.Enter **Login**
Expected Result	Error message should display: *"Your password is invalid!"*
Actual Result	Error message displayed: *"Your username is invalid!"*
Screenshot	*login_invalid_tomsmith.png*
Status	*Simulated Bug – Created to demonstrate real-world defect documentation.*

Description:

This simulated bug demonstrates a negative login validation mismatch, showcasing critical thinking in identifying text-based UI errors. The automation script `test_login.py` validates both positive and negative cases.

7. Script Implementation

Base Page

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

from selenium.webdriver.support import expected_conditions as EC

from selenium.webdriver.common.by import By

import os



class BasePage:

  def __init__(self, driver):

    self.driver = driver

    self.wait = WebDriverWait(driver, 20)
```

```
    def click(self, locator):
        self.wait.until(EC.element_to_be_clickable(locator)).click()

    def send_keys(self, locator, text):
        self.wait.until(EC.element_to_be_clickable(locator)).send_keys(text)

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

Home Page

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

class HomePage(BasePage):

    DROPDOWN_LINK = (By.LINK_TEXT, "Dropdown")
    ALERTS_LINK = (By.LINK_TEXT, "JavaScript Alerts")
    DYNAMIC_LOADING_LINK = (By.LINK_TEXT, "Dynamic Loading")

    def open(self):
        self.driver.get("https://the-internet.herokuapp.com/")

    def go_to_dropdown(self):
        self.click(self.DROPDOWN_LINK)
```

```
    def go_to_alerts(self):
        self.click(self.ALERTS_LINK)

    def go_to_dynamic_loading(self):
        self.click(self.DYNAMIC_LOADING_LINK)
```

Login 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 LoginPage:
    URL = "https://the-internet.herokuapp.com/login"  # page-specific URL

    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(driver, 10)

    # Open page
    def open(self):
        self.driver.get(self.URL)

    # Locators
    username_input = (By.ID, "username")
```

```
    password_input = (By.ID, "password")
    login_button = (By.CSS_SELECTOR, "button[type='submit']")
    success_message = (By.ID, "flash")

    # Actions
    def enter_username(self, username):
self.wait.until(EC.visibility_of_element_located(self.username_input)).send_keys(username)

    def enter_password(self, password):
self.wait.until(EC.visibility_of_element_located(self.password_input)).send_keys(password)

    def click_login(self):
        self.wait.until(EC.element_to_be_clickable(self.login_button)).click()

    def get_success_message(self):
        return self.wait.until(EC.visibility_of_element_located(self.success_message)).text
```

Dropdown Page

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import Select
from pages.base_page import BasePage

class DropdownPage(BasePage):
    DROPDOWN = (By.ID, "dropdown")
```

```
    def select_by_text(self, text):
        select_element = self.wait.until(lambda d: d.find_element(*self.DROPDOWN))
        select = Select(select_element)
        select.select_by_visible_text(text)
        return select.first_selected_option.text

    def select_by_value(self, value):
        select_element = self.wait.until(lambda d: d.find_element(*self.DROPDOWN))
        select = Select(select_element)
        select.select_by_value(value)
        return select.first_selected_option.text
```

Alert Page

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

class AlertPage(BasePage):

    JS_ALERT_BTN = (By.XPATH, "//button[text()='Click for JS Alert']")
    JS_CONFIRM_BTN = (By.XPATH, "//button[text()='Click for JS Confirm']")
    JS_PROMPT_BTN = (By.XPATH, "//button[text()='Click for JS Prompt']")
    RESULT_TEXT = (By.ID, "result")

    def handle_js_alert(self):
        self.click(self.JS_ALERT_BTN)
        alert = self.driver.switch_to.alert
```

```
        alert.accept()
        return "JS Alert handled"

    def handle_js_confirm(self):
        self.click(self.JS_CONFIRM_BTN)
        alert = self.driver.switch_to.alert
        alert.dismiss()
        return "JS Confirm handled"

    def handle_js_prompt(self, message):
        self.click(self.JS_PROMPT_BTN)
        alert = self.driver.switch_to.alert
        alert.send_keys(message)
        alert.accept()
        return self.get_text(self.RESULT_TEXT)
```

Dynamic Loading Page

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


class DynamicLoadingPage(BasePage):

    START_BUTTON = (By.XPATH, "//div[@id='start']/button")
    FINISH_TEXT = (By.ID, "finish")
```

```
    EXAMPLE2_LINK = (By.LINK_TEXT, "Example 2: Element rendered after the fact")

    def open_example2(self):
        self.click(self.EXAMPLE2_LINK)

    def start_loading(self):
        self.click(self.START_BUTTON)
        return self.get_text(self.FINISH_TEXT)
```

Conftest

```
# conftest.py
import os
import pytest
from selenium import webdriver
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.chrome.options import Options as ChromeOptions
from webdriver_manager.chrome import ChromeDriverManager


def create_chrome_driver():
    """Create Chrome driver with all automation warnings and popups disabled."""
    chrome_options = ChromeOptions()

    # Stable launch arguments
    chrome_options.add_argument("--start-maximized")
```

```
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--disable-dev-shm-usage")
chrome_options.add_argument("--disable-gpu")
chrome_options.add_argument("--disable-notifications")
chrome_options.add_argument("--disable-infobars")
chrome_options.add_argument("--disable-popup-blocking")

#  Disable Chrome password manager + popup
chrome_options.add_experimental_option("prefs", {
   "credentials_enable_service": False,
   "profile.password_manager_enabled": False
})
chrome_options.add_argument("--disable-save-password-bubble")
chrome_options.add_argument("--disable-password-manager-reauthentication")

#  Suppress “Chrome is being controlled by automated test software”
chrome_options.add_experimental_option("excludeSwitches", ["enable-automation"])
chrome_options.add_experimental_option("useAutomationExtension", False)
chrome_options.add_argument("--disable-blink-features=AutomationControlled")

# Create Chrome driver
driver = webdriver.Chrome(
   service=ChromeService(ChromeDriverManager().install()),
   options=chrome_options
)
return driver
```

```
@pytest.fixture
def chrome_driver():
    driver = create_chrome_driver()
    yield driver
    driver.quit()
```

Test_login

```
# tests/test_login.py
import pytest
from pages.login_page import LoginPage
from utils.screenshot import take_screenshot

@pytest.mark.parametrize(
    "username,password,expected",
    [
        ("tomsmith", "SuperSecretPassword!", True),
        ("tomsmith", "wrongpass", False),
        ("wronguser", "SuperSecretPassword!", False)
    ]
)
def test_login(chrome_driver, username, password, expected):
    driver = chrome_driver
    login_page = LoginPage(driver)
    login_page.open()
```

```
    take_screenshot(driver, "login_page_opened")
    login_page.enter_username(username)
    login_page.enter_password(password)
    login_page.click_login()
    take_screenshot(driver, f"login_after_click_{username}")

    flash_text = login_page.get_success_message()
    try:
        if expected:
            assert "You logged into a secure area!" in flash_text
            take_screenshot(driver, f"login_success_{username}")
        else:
            assert "Your username is invalid!" in flash_text or "Your password is invalid!" in
flash_text
            take_screenshot(driver, f"login_invalid_{username}")
    except AssertionError:
        take_screenshot(driver, f"login_fail_{username}")
        raise
```

Test_app_feature

```
import pytest
from pages.home_page import HomePage
from pages.dropdown_page import DropdownPage
from pages.alert_page import AlertPage
```

```
from pages.dynamic_loading_page import DynamicLoadingPage
from utils.screenshot import take_screenshot


class TestHerokuApp:

    def test_herokuapp_all(self, chrome_driver):
        driver = chrome_driver
        home = HomePage(driver)
        home.open()
        take_screenshot(driver,"home_page_opened")

        # ---------- Dropdown ----------
        home.go_to_dropdown()
        dropdown = DropdownPage(driver)
        assert dropdown.select_by_text("Option 1") == "Option 1"
        assert dropdown.select_by_value("2") == "Option 2"
        take_screenshot(driver,"dropdown_selected_options")
        driver.back()

        # ---------- Alerts ----------
        home.go_to_alerts()
        alert_page = AlertPage(driver)
        assert alert_page.handle_js_alert() == "JS Alert handled"
        assert alert_page.handle_js_confirm() == "JS Confirm handled"
        assert "Deepika - QA Tester" in alert_page.handle_js_prompt("Deepika - QA Tester")
```

```
take_screenshot(driver,"alerts_all_handled")
driver.back()


# ---------- Dynamic Loading ----------
home.go_to_dynamic_loading()
dyn_page = DynamicLoadingPage(driver)
dyn_page.open_example2()
take_screenshot(driver,"dynamic_loading_page_opened")
assert dyn_page.start_loading() == "Hello World!"
take_screenshot(driver,"dynamic_loading_completed")
```

8. Screenshots

8.1 Login Page Screenshot

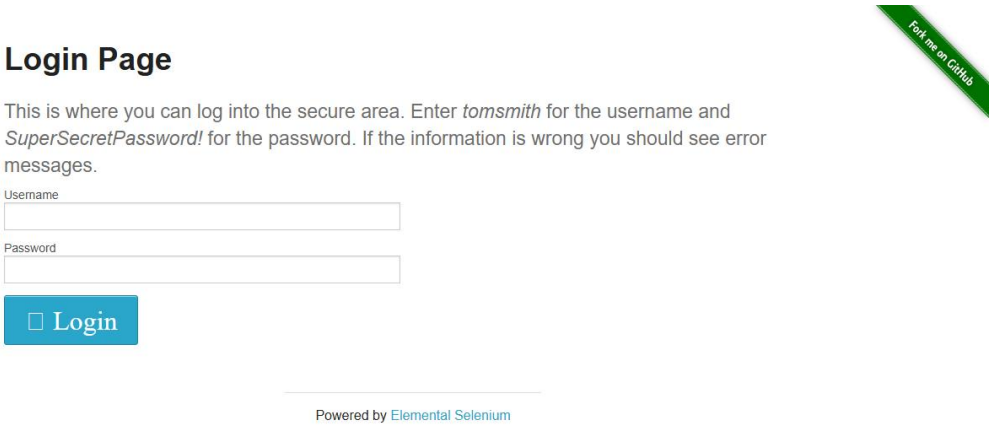

Figure 1: *Login Page – HerokuApp Login Screen Opened*

8.2 Successful Login Screenshot

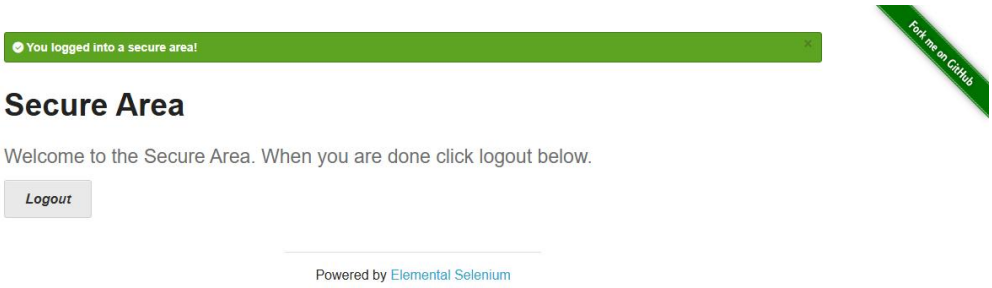

Figure 2*: Successful Login – Secure Area Message Displayed*

8.3 Invalid Login Attempt Screenshot

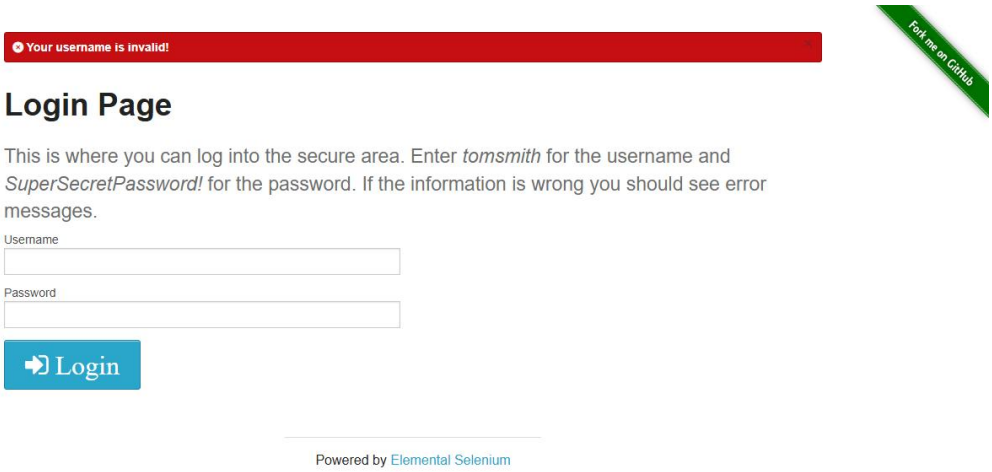

Figure 3: *Invalid Login Attempt – Error Message Displayed*

8.4 Invalid Login Attempt Screenshot

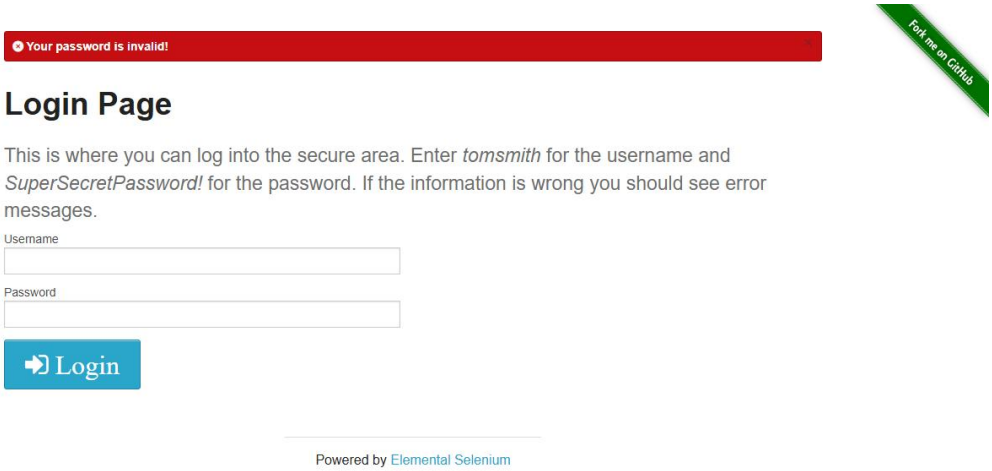

Figure 4: *Invalid Login Attempt – Error Message Displayed*

8.5 Home Page Screenshot

Figure 5: *HerokuApp Home Page – Feature Navigation Displayed*

8.6 Dropdown Selection Screenshot

Figure 6: *Dropdown Feature – Options Selected Successfully*

8.7 Alert Handling Screenshot

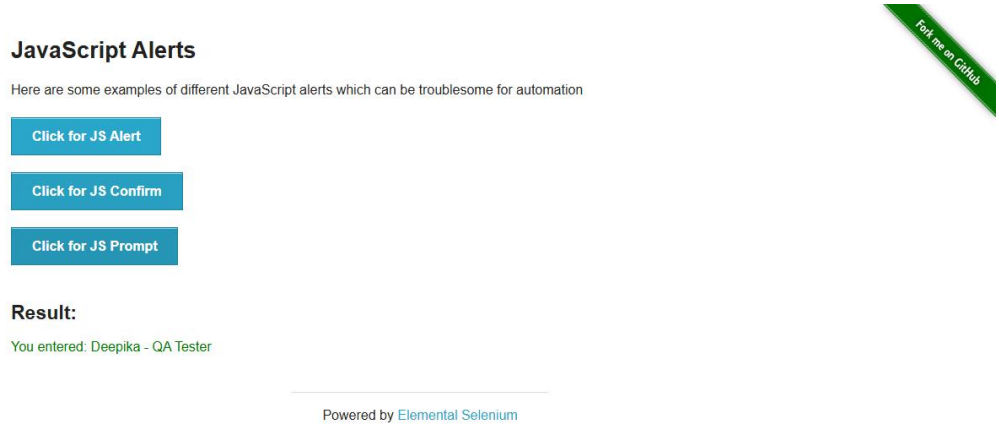

Figure 7: *JavaScript Alerts – Handled Successfully*

8.8 Dynamic Loading Page Screenshot

Figure 8: *Dynamic Loading Page – Example 2 Opened*

8.9 Dynamic Loading Completed Screenshot

Figure 9: *Dynamic Loading Completed – "Hello World!" Message Displayed*

8.10 Pytest HTML Report Screenshot

Click here to view the full HTML test report ***FINAL AUTOMATION PROJECT1\report.html***

Figure 10: *Pytest HTML Report – Test Execution Summary*

8.11 Folder Structure Screenshot

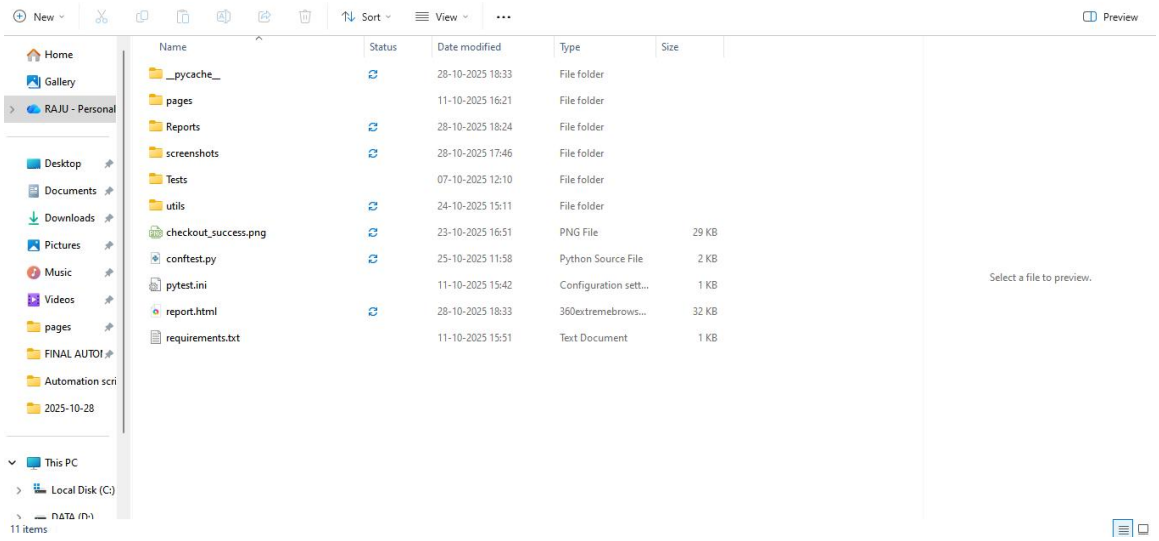

Figure 11: *Folder Structure – Organized Project Files (Pages, Tests, Utils)*

8.12 Allure Report Screenshots

Purpose: To show advanced reporting integration in your QA automation project.

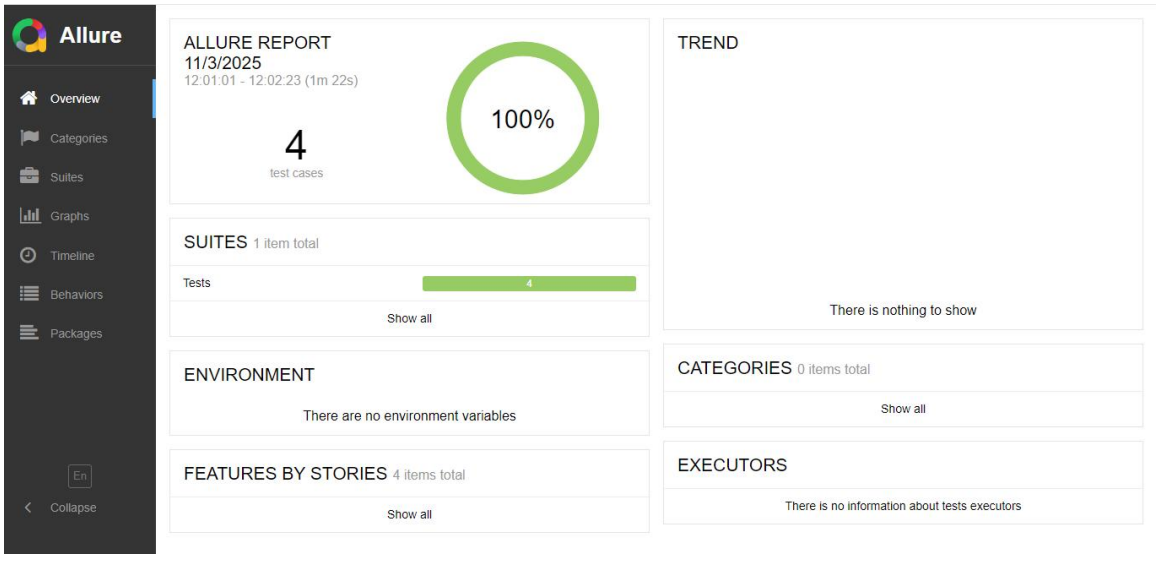

Figure 12: *Allure Dashboard Overview – Displays total passed/failed test summary and trend graph.*

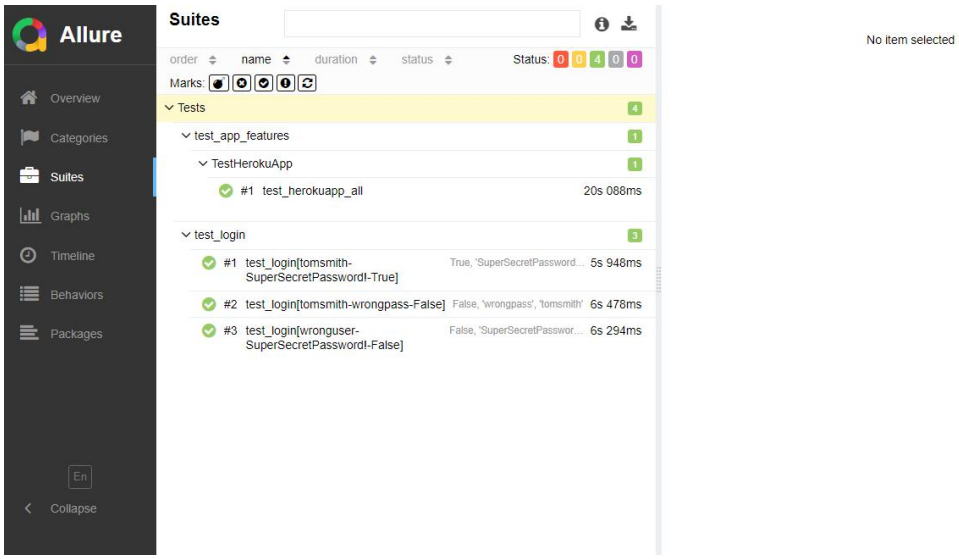

Figure 13: *Allure Suites View – Shows organized test suites and their execution status.*

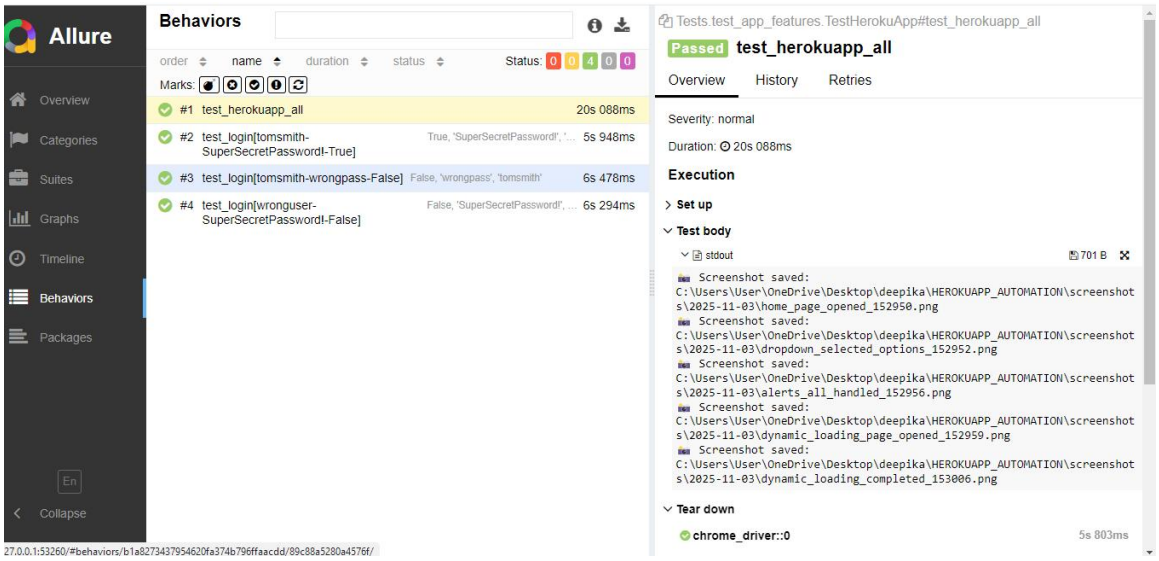

Figure 14: *Allure Behaviors View – Represents test coverage based on features and scenarios.*

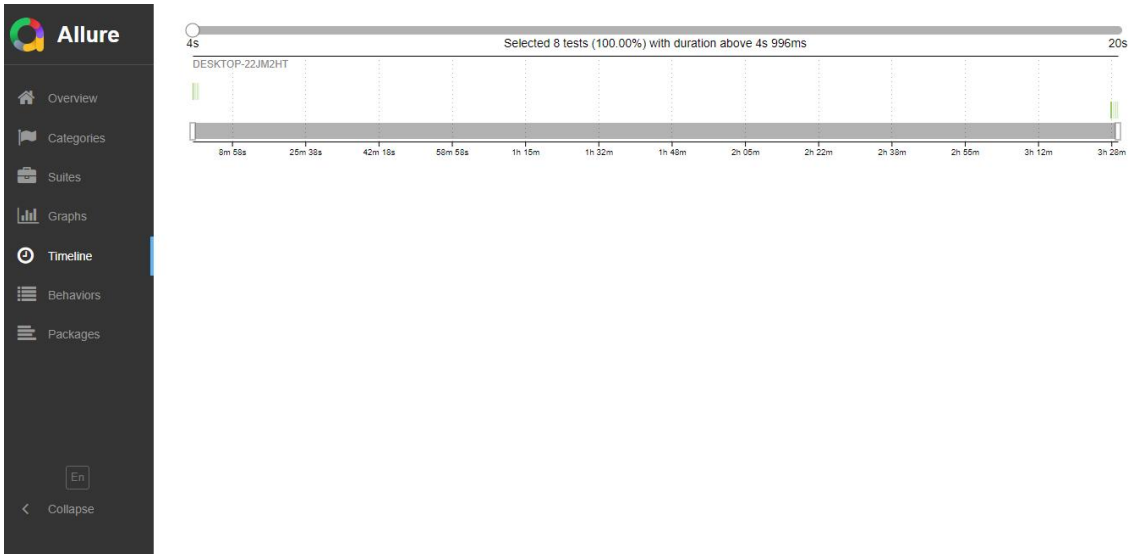

Figure 15: *Allure Timeline View – Displays duration and parallel execution of test cases.*

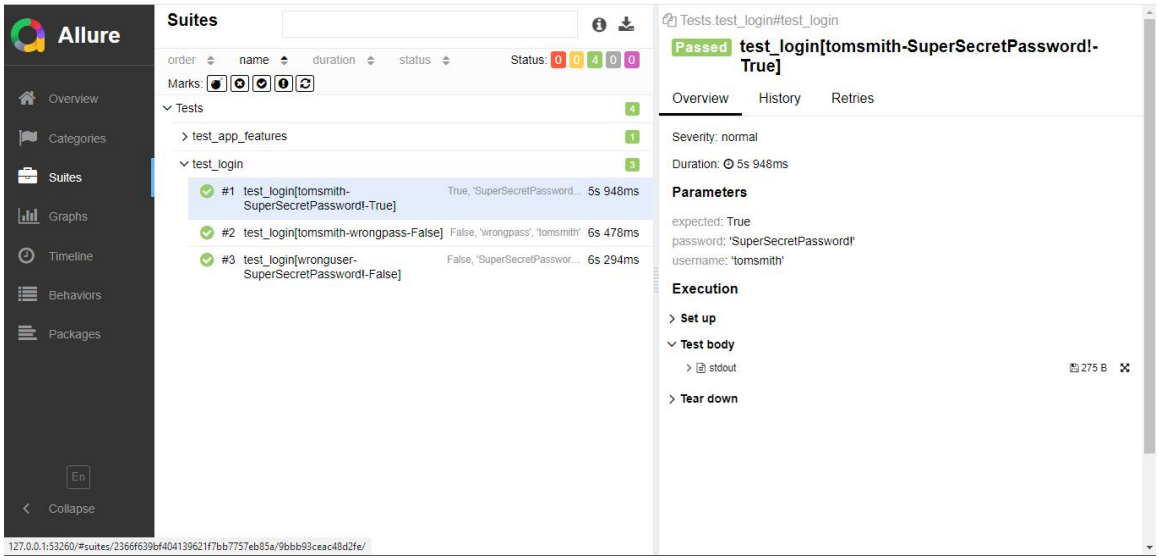

Figure 16: Allure Test Case Detail – Shows step-by-step logs of a specific test case.

9. Result Summary

Total Test Cases	5
Passed	5
Failed	0
Browser	Chrome
Execution Time	~1.5 minutes

10. Conclusion

This project demonstrates automation of multiple HerokuApp functionalities using Python, Selenium, and Pytest
following the Page Object Model (POM) design pattern. Screenshots are captured at every test step for better traceability,
and test results confirm smooth functional execution across all modules.

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 visibility
Ready for **CI/CD integration (GitHub Actions)** for continuous testing

GitHub Repository Link:
https://github.com/deepika-sekar-qa/DeepikaS_QA_Automation_Portfolio