

Ash and Emily

Test Automation

11th September 24



Agenda

1. Why test automation – 5m
2. What do we want you to get out of the today's session – 1m
3. Explain the task – 10m
4. Have a go – 20m
5. Give us feedback – 4m



Why do we do test automation?

Join at menti.com | use code 7222 1437

Why do we do test automation?

transpiration

leader focus

creative

inspiration

bold

fast



Objectives

Objective: Show how easy it is to get started with automation

This session is designed as an introduction, so the goal is not to master this today but to start a conversation that we can continue in the future. We understand not everyone will be involved in writing tests but we hope this will give an understanding of how it is done.

We recognize that our group has a range of experience and coding backgrounds, so it's perfectly okay if not everything clicks right away.

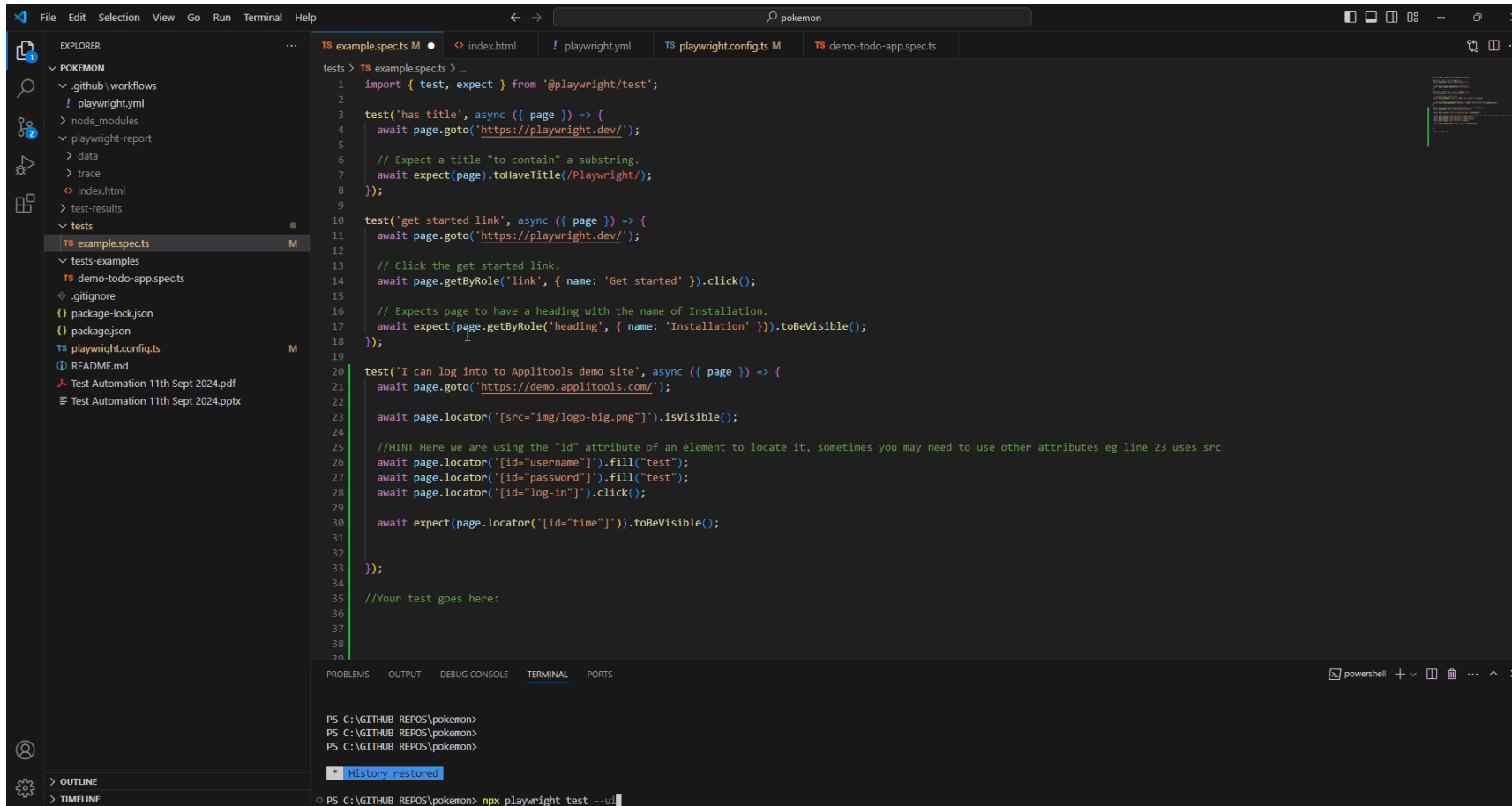
Tool selected: Playwright

- Microsoft tool
- Well documented
- Increased use in recent times.

Taken away installation but is very simple and if interested can be found [here:](#)

Before we explain the task....

Demo on the repository you have and how to run Playwright In UI mode



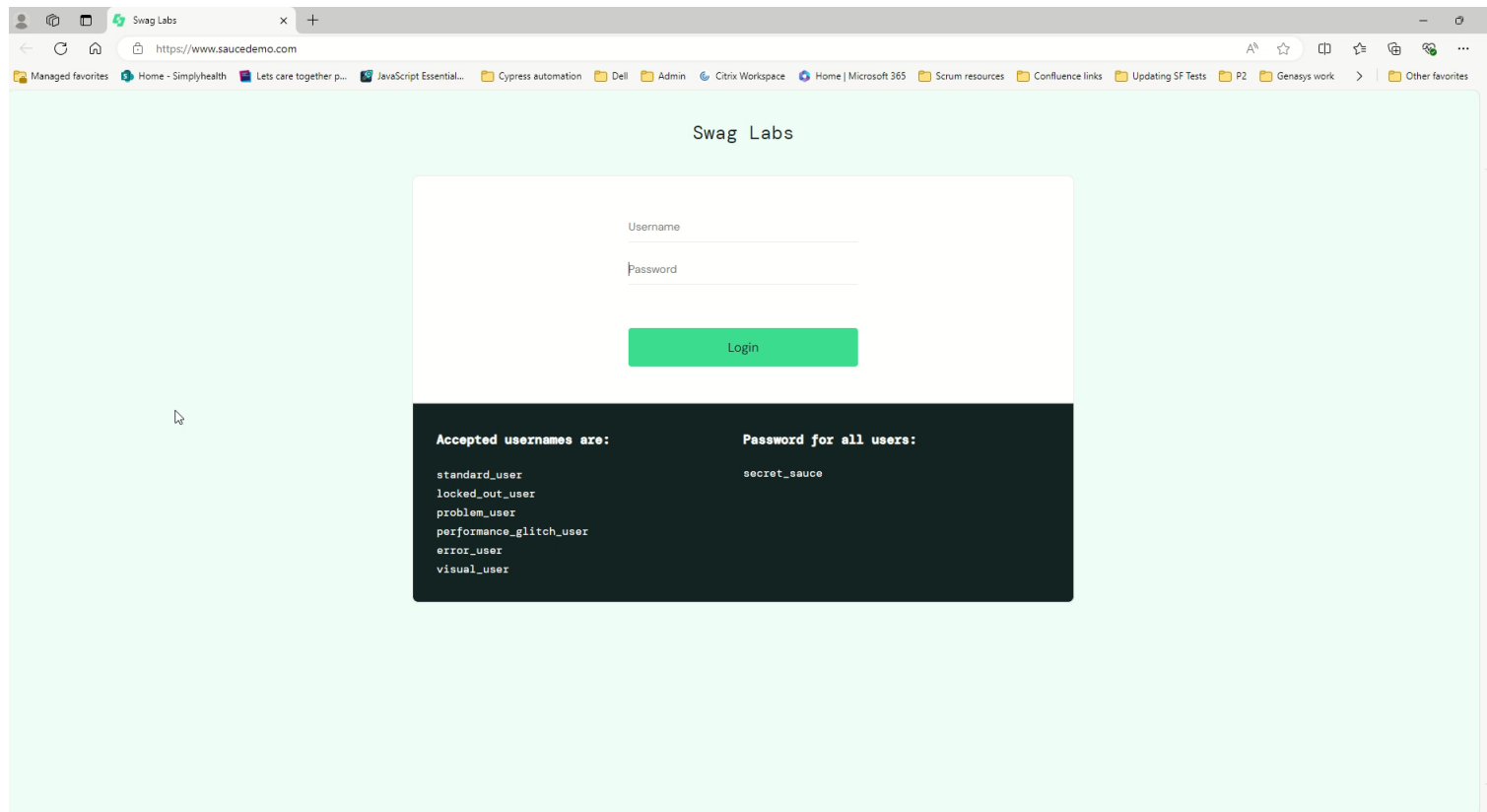
The screenshot shows a Visual Studio Code editor with a project named 'pokemon'. The Explorer sidebar on the left shows the file structure, including 'tests' and 'example.specs'. The main editor area displays the content of 'example.specs.ts', which contains three Playwright test cases. The first test checks the page title, the second tests a 'get started' link, and the third tests a login form. The bottom panel shows the Terminal with the command 'npx playwright test --ui' and its output.

```
tests > TS example.specs.ts > ...
1  import { test, expect } from '@playwright/test';
2
3  test('has title', async ({ page }) => {
4    await page.goto('https://playwright.dev/');
5
6    // Expect a title "to contain" a substring.
7    await expect(page).toHaveTitle(/Playwright/);
8  });
9
10 test('get started link', async ({ page }) => {
11   await page.goto('https://playwright.dev/');
12
13   // Click the get started link.
14   await page.getByRole('link', { name: 'Get started' }).click();
15
16   // Expects page to have a heading with the name of Installation.
17   await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();
18 });
19
20 test('I can log into to AppliTools demo site', async ({ page }) => {
21   await page.goto('https://demo.applitoools.com/');
22
23   await page.locator('[src="img/logo-big.png"]').isVisible();
24
25   //HINT Here we are using the "id" attribute of an element to locate it, sometimes you may need to use other attributes eg line 23 uses src
26   await page.locator('[id="username"]').fill("test");
27   await page.locator('[id="password"]').fill("test");
28   await page.locator('[id="log-in"]').click();
29
30   await expect(page.locator('[id="time"]')).toBeVisible();
31
32 });
33
34 //Your test goes here:
35
36
37
38
39
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\GITHUB REPOS\pokemon>
PS C:\GITHUB REPOS\pokemon>
PS C:\GITHUB REPOS\pokemon>
History restored
PS C:\GITHUB REPOS\pokemon> npx playwright test --ui
```

Demo on how to find locators



Do we have any developers in the room who would be happy to support each group?

The task – Option 1 – simplest option

1. UI test to log in to an application
 1. Navigate to URL [Swag Labs \(saucedemo.com\)](https://saucedemo.com)
 2. Enter username *standard_user*
 3. Enter password *secret_sauce*
 4. Click login
 5. Confirm user is logged in

The task – Option 2 – more complex

1. API test to call an API endpoint
 1. Call a “*get*” endpoint e.g. <http://pokeapi.co/api/v2/pokemon/charmeleon>
 1. headers: {'Content-Type': 'application/json'},
 2. Confirm success response (200)
 3. View the response
 1. What abilities does your pokemon have?

API documentation <https://pokeapi.co/>

Do we have any developers in the room who would be happy to support each group?

Example solutions are available on the next slides

Example solutions – UI test

```
test('I can log into to SwagLabs', async ({ page }) => {  
  await page.goto('https://www.saucedemo.com/');  
  
  // Expect a title "to contain" a substring.  
  await expect(page).toHaveTitle(/Swag Labs/);  
  
  await page.locator('[data-test="username"]').fill("standard_user");  
  await page.locator('[data-test="password"]').fill("secret_sauce");  
  await page.locator('[data-test="login-button"]').click();  
  await expect(page.locator('[data-test="shopping-cart-link"]')).toBeVisible();  
  
});
```

Example solutions – API test

```
test('I can find information about Charmander', async ({ request }) => {  
  const response = await request.post(`http://pokeapi.co/api/v2/pokemon/ditto`, {  
    headers: {  
      'Content-Type': 'application/json',  
    })  
  
  console.log(await response.json())  
  expect(response.status()).toBe(200)  
});
```

Tips

<code>npx playwright test --ui</code>	Type into terminal to open Playwright UI
API testing Playwright	Info writing API test
Writing tests Playwright	UI test writing basics
LocatorAssertions Playwright	Assertions for elements
https://playwright.dev/docs/intro#running-the-example-test-in-ui-mode	Info running in UI mode

Future

Page Object model, BDD and other best practice - [Best practice / Principles of test automation - Test Automation - Confluence \(atlassian.net\)](#)

**What have you
learnt today?**



Basic setup of laptops for info:

- Laptop with Visual studio code and internet access
- Github repo cloned and a branch created with playwright installed
- Node and git installed

