Test Automation with Playwright

1. Create a new project

1. Open the terminal
2. Create a new directory: `mkdir playwright-training`
3. Navigate to the directory: `cd playwright-training`
4. Initialize a new Playwright project with NPM: `npm init playwright@latest`
5. Select `TypeScript` and install the browsers.

Congratulations, you have successfully set up a new Playwright project.

Implement our first test

1. Open Visual Studio Code (if not already done)
2. Open the project we just created.
    - File
    - Open Folder
    - Navigate to ‘playwright-training’
    - Click on [Open]

3. Modify the default spec-file.
    - The file resides in `tests/example.spec.ts`
    - Change the contents so that `practicesoftwaretesting.com` will be opened and verify the title.

4. Execute the modified spec-file
    - Click on Terminal (top menu)
    - Click on New Terminal
    - Run the following command: `npx playwright test --ui
    - OR
    - Click on the play-button next to the test.

> Tip: The `baseURL` can also be set in `playwright.config.ts`

Record a test script

Playwright comes with a test recorder (Codegen)

1. Run the following command:
   `npx playwright codegen https://practicesoftwaretesting.com`

> Note: By default it will try to search for the `data-testid` attribute, other attributes can be set in the `testIdAttribute` property in `playwright.config.ts`

Implement more tests

Implement automated tests for the following scenarios:

Submit contact form

1. Click on Contact
2. Complete the form
3. Verify the text “Thanks for your message! We will contact you shortly.” is displayed.

login

1. Click on Sign in
2. Enter customer@practicesoftwaretesting.com as email address
3. Enter welcome01 as password
4. Click in Login
5. Verify that the text My account is visible

> Check if you can repeatedly run the test.

Navigate to the product detail page

1. On the homepage click on a product
2. Verify that the product name is displayed

Navigate to the product category page

1. Click on Categories
2. Click on Power tools
3. Verify the amount of products

Navigate to the product category page (unhappy)

1. Click on Categories
2. Click on Special tools
3. Verify that the text “There are no products available yet” is displayed

Randomize data using Faker

Create a new testscript that registers a new account. This time we will use Faker to generate the data needed for this testscript.

1. In Visual Studio Code run the following command in the terminal: `npm install @faker-js/faker –save-dev`
2. Create a new spec-file, name it: register.spec.cy.ts
3. The following import statement makes faker available in this file: `import { faker } from '@faker-js/faker'` Implement the register test script and use faker for the data generation, like: `faker.internet.email()`

Data-driven testing

With Playwright we can also implement data-driven testing.

The data-array can be stored in the same file as the test. We have the following account: `customer@practicesoftwaretesting.com` , `customer2@practicesoftwaretesting.com` , `admin@practicesoftwaretesting.com` all account have the same password: `welcome01`

```
const accounts = [

  // Data items

];
```

Use this file in a new data-driven test.

```
import { test, expect } from '@playwright/test';

for ( const account of accounts ) {
  test( `testing with ${account.email}`, async ( { page } ) => {

  // Test implementation

  } );
}
```

Implement Page Objects (Login)

Create Page Objects for the 'home.page.ts', 'login.page.ts', 'account.page.ts'

- Group pages together in a subfolder, called 'pages'
- Verifications should be part of the test and not part of the page objects

Implement a new login test that uses the page object model.

API Testing (GET)

We use: https://api-v4.practicesoftwaretesting.com/api/documentation

1. Try to retrieve a single brand, category or product

2. Implement a validation

API Testing (POST)

We use: https://api-v4.practicesoftwaretesting.com/api/documentation

1. Try to add a brand, category or product

2. Implement a validation

API Testing (PUT)

We use: https://api-v4.practicesoftwaretesting.com/api/documentation

1. Try to update a brand, category or product

2. Implement a validation

API Testing (Protect API)

We use: https://api-v4.practicesoftwaretesting.com/api/documentation

1. Try to retrieve the invoices
2. Log in as an admin or customer (via API) and extract the token

email: `admin@practicesoftwaretesting.com` / password: `welcome01`

1. Pass the token to the subsequent invoices request.
2. Implement a validation

Reporting (junit)

If you run the tests from a pipeline, you might want to generate some output that the pipeline can easily understand. Like a junit report. In this exercise we will generate junit xml reports

1. Define the reporter in `playwright.config.ts`

```
reporter: [['junit', { outputFile: 'results.xml' }]],
```

1. Run your tests from commandline and specify the reporter, like:
   `npx playwright run`

Reporting (allure)

In this Lab exercise, we add a more colorful report: Allure

1. Install the allure dependency to your project, with: `npm i -D allure-playwright`
2. Adjust the setupNodeEvents in `playwright.config.ts`, like:

   File: `playwright.config.ts`

   ```
   import { testPlanFilter } from "allure-playwright/dist/testplan";

   export default defineConfig({
     .....
     grep: testPlanFilter(),
     reporter: [["html"], ["line"], ["allure-playwright"]],

   });
   ```

3. Run your tests with:

`npx playwright test`

Next is to generate the report. There are multiple ways to get the allure command line tool on your system. For JavaScript projects, this is the easiest one:

Install: `npm install -D allure-commandline`

After the installation, you can run the following terminal command: `npx allure serve`