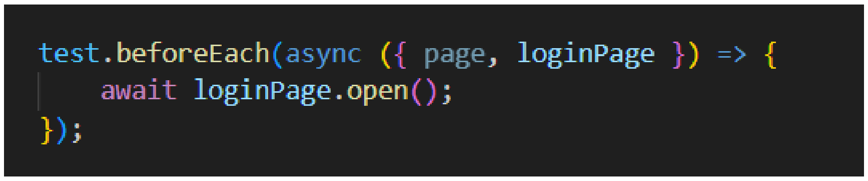
Hi

I'd like to share some good ideas/features of Playwright framework in context of e2e tests.

I've created simple solution in repository https://github.com/ChrisFrost2/playwright-examples based on *sausedemo* page.

Selected points worth to emphasize in Playwright solution:

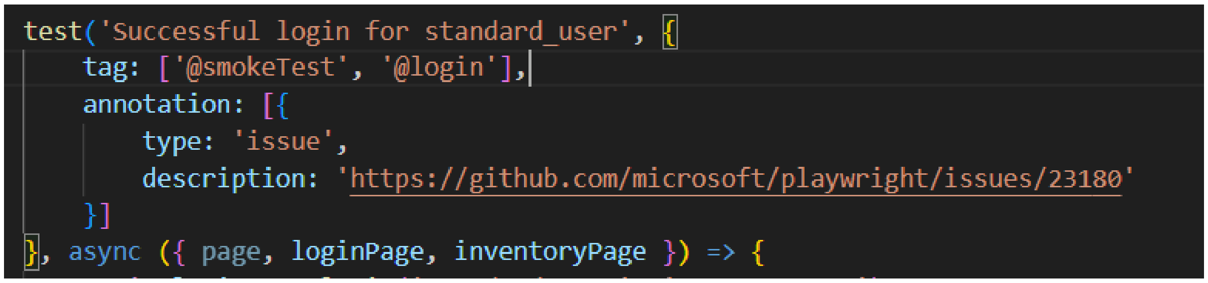
1. Fixtures mechanism (helpers), possible to extend test in easy and fast way. Here used for extending by one of POMs



```
test.beforeEach(async ({ page, loginPage }) => {
    await loginPage.open();
});
```

We reuse part of code and encapsulate logic, e.g. POMs work faster as fixture compared to the traditional approach where we need to instantiate the class each time.

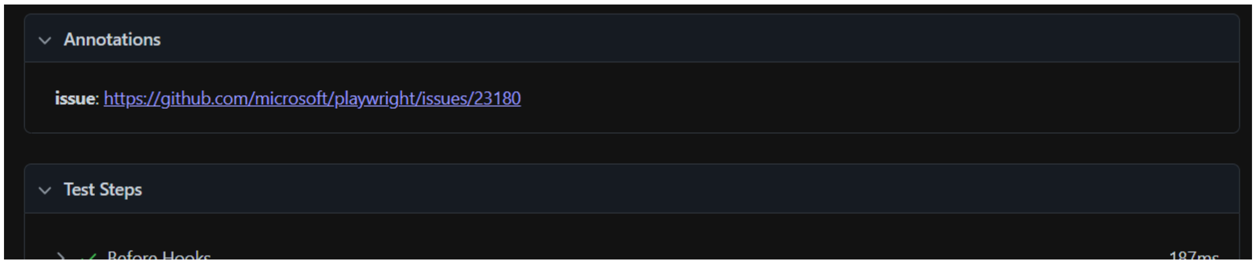
2. Tags and annotations



```
test('Successful login for standard_user', {
    tag: ['@smokeTest', '@login'],
    annotation: [{
        type: 'issue',
        description: 'https://github.com/microsoft/playwright/issues/23180'
    }]
}, async ({ page, loginPage, inventoryPage }) => {
```

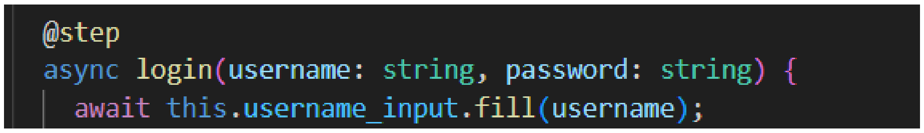
We can describe tests by tags and run specific groups e.g. *npx playwright test -g "@only"*.

Possible to connect test implementation in code with test definition in e.g. jira by annotation with link. Will be presented in the report



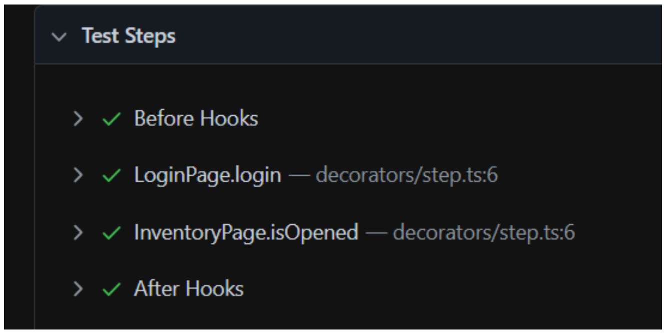
issue: https://github.com/microsoft/playwright/issues/23180

3. Test.step added as decorator



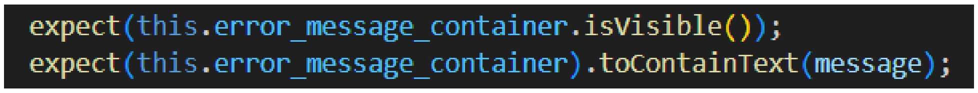
```
@step
async login(username: string, password: string) {
  await this.username_input.fill(username);
```

Thanks to the code in 'tests/decorators/step.ts' all methods with @step decorator will be wrapped by 'test.step' automatically and report will be more readable (easier to read, debug and reproduce) e.g.



```
Test Steps
  ✓ Before Hooks
  ✓ LoginPage.login — decorators/step.ts:6
  ✓ InventoryPage.isOpened — decorators/step.ts:6
  ✓ After Hooks
```

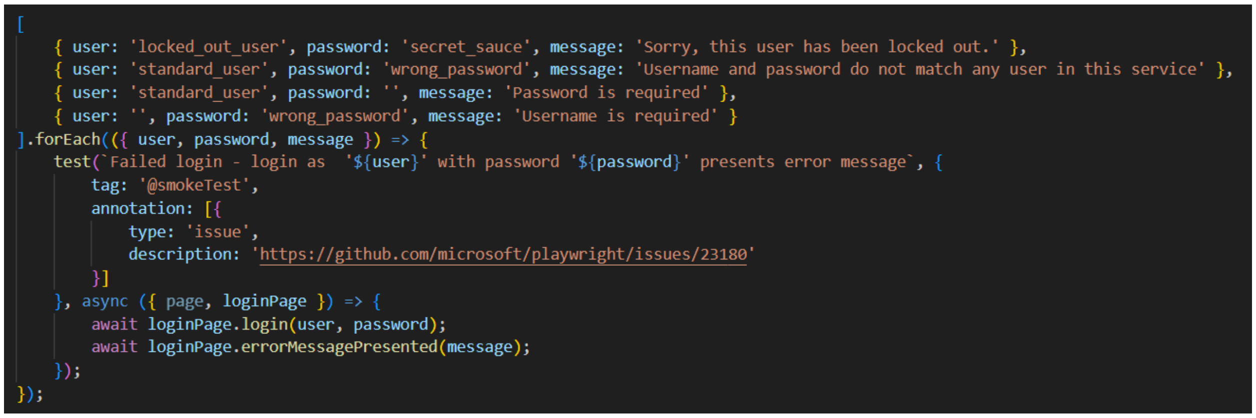
4. Use locators and auto-retrying assertions



```
expect(this.error_message_container.isVisible());
expect(this.error_message_container).toContainText(message);
```

Potentially less flaky, more flexible.

5. Data-driven testing by parametrization of test, like in *login.spec.ts* e.g.



```
[
    { user: 'locked_out_user', password: 'secret_sauce', message: 'Sorry, this user has been locked out.' },
    { user: 'standard_user', password: 'wrong_password', message: 'Username and password do not match any user in this service' },
    { user: 'standard_user', password: '', message: 'Password is required' },
    { user: '', password: 'wrong_password', message: 'Username is required' }
].forEach(({ user, password, message }) => {
    test(`Failed login - login as  '${user}' with password '${password}' presents error message`, {
        tag: '@smokeTest',
        annotation: [{
            type: 'issue',
            description: 'https://github.com/microsoft/playwright/issues/23180'
        }]
    }, async ({ page, loginPage }) => {
        await loginPage.login(user, password);
        await loginPage.errorMessagePresented(message);
    });
});
```

6. Possible to use UI actions, API calls in one engine and mock endpoints out of the box. It gives testes way to build mixed, flexible test scenarios.

Playwright creators collected many good solutions from other framworks in their product. I just wanted to share it, because it looks usefull.