orange™

versus

Yann Helleboid

Orange Innovation 2024

Cypress, Playwright, what ?

Test automation tools

Cypress, Playwright, why ?

Open source / Free

Easy / Powerful

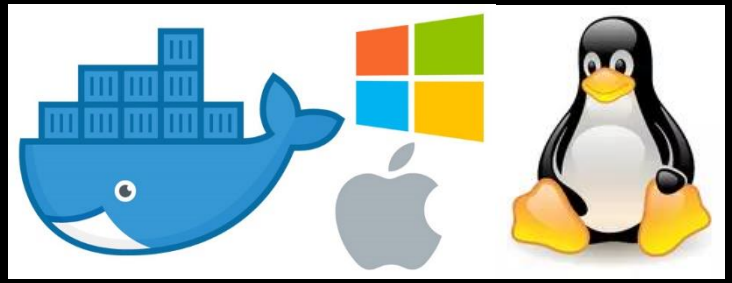
Cypress, Playwright, what for ?

HTTP oriented functional tests

GUI and API

Cypress, Playwright, where ?

Most platforms



Your machine or on a VM or on a cloud

Cypress, Playwright, what’s the difference ?

The tools are very similar

Mostly through their “dynamic waiting” feature

friendly ui
better integrated js/browser
standard libs included

step-by-step debug possible
js + python + others
browser tab management

Cypress, Playwright, how ?

Hands-on training

see by yourselves

 means it’s your turn to type

Let’s go !

exercise 1

goal : locally install the tools

install the local environment


exercise 1

install a proper editor or IDE

notepad++, eclipse, vscode, vim …

install node.js

Go to https://nodejs.org/en

Follow the install guide for your OS

launch : 2 modes

open in a GUI => design, debug and analyse
use headed browser

run as a CLI => run tests
use headless browser

Cypress, Playwright and browsers

Cypress is shipped with a electron browser

Cypress can use installed standard browsers

Cypress can control headed or headless browsers

Playwright is shipped with out any browser

You must install specific browsers for Playwright

Playwright can control headed or headless browsers

install


```
λ mkdir cypress_training
λ cd cypress_training
λ npm init -y
λ npm install cypress
```

```
λ mkdir playwright_training
λ cd playwright_training
λ npm init playwright
```

Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
√ Do you want to use TypeScript or JavaScript? · JavaScript
√ Where to put your end-to-end tests? · tests
√ Add a GitHub Actions workflow? (y/N) · false
√ Install Playwright browsers (can be done manually via 'npx playwright install')? (Y/n) · true

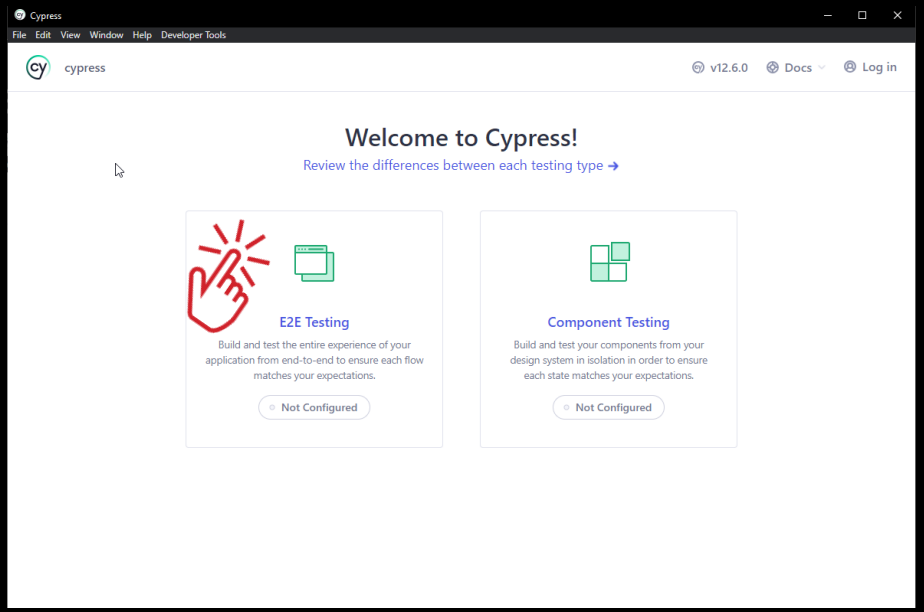
install


```
λ npx cypress open
```

```
λ npx playwright test --ui
```

```
λ npx playwright test --debug
```

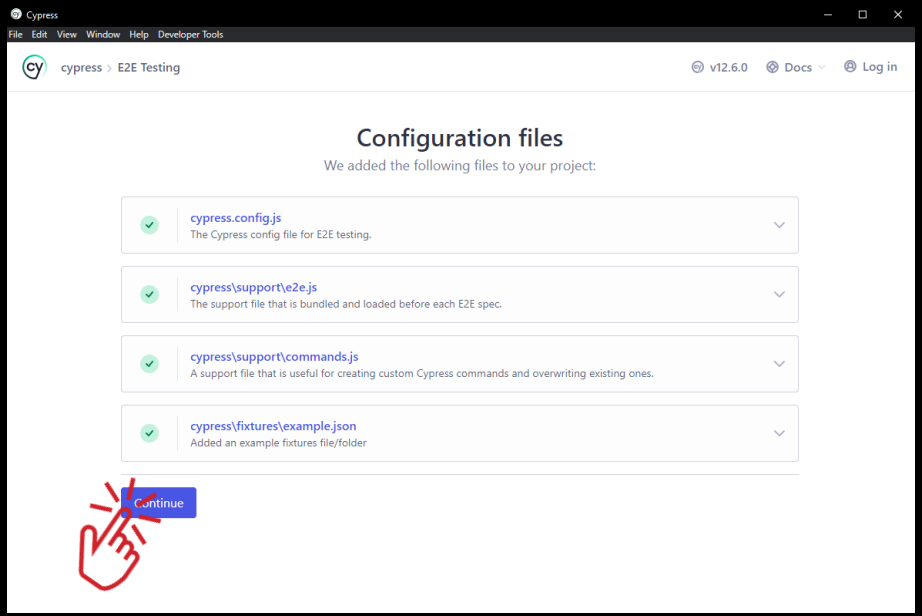
install



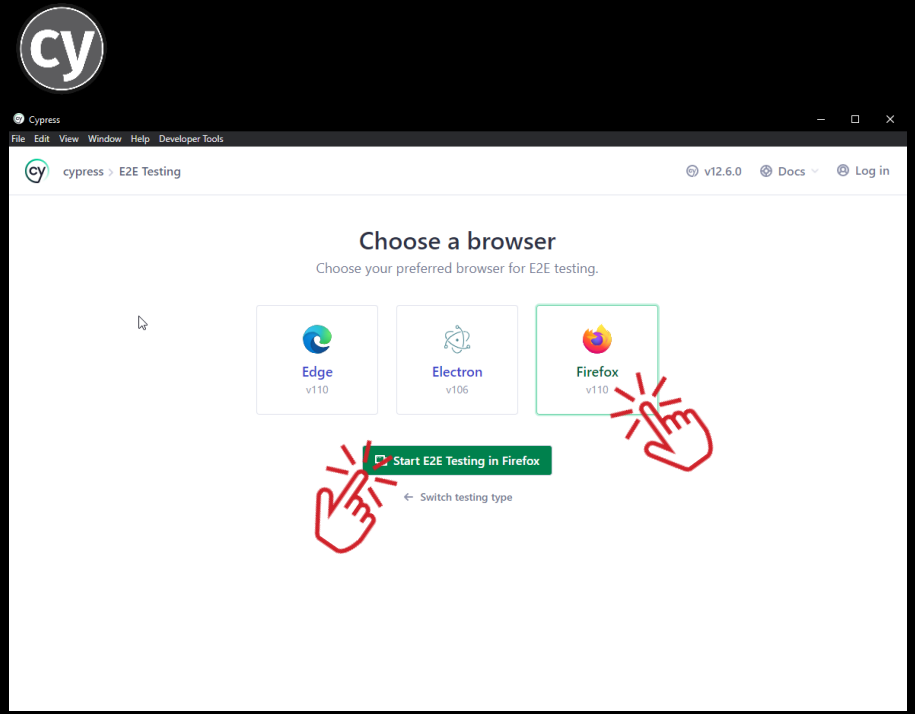
exercise 1

install

exercise 1



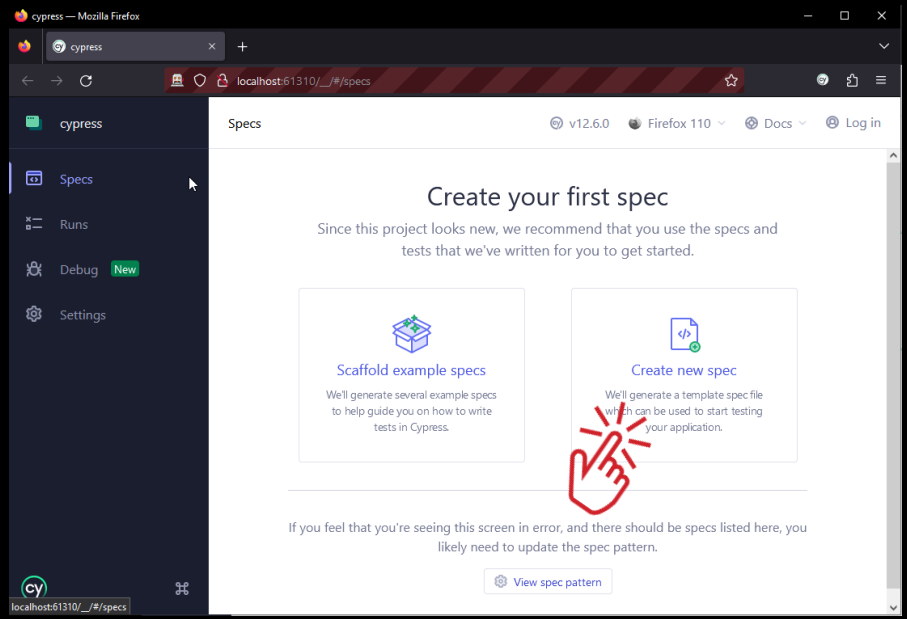
install



exercise 1

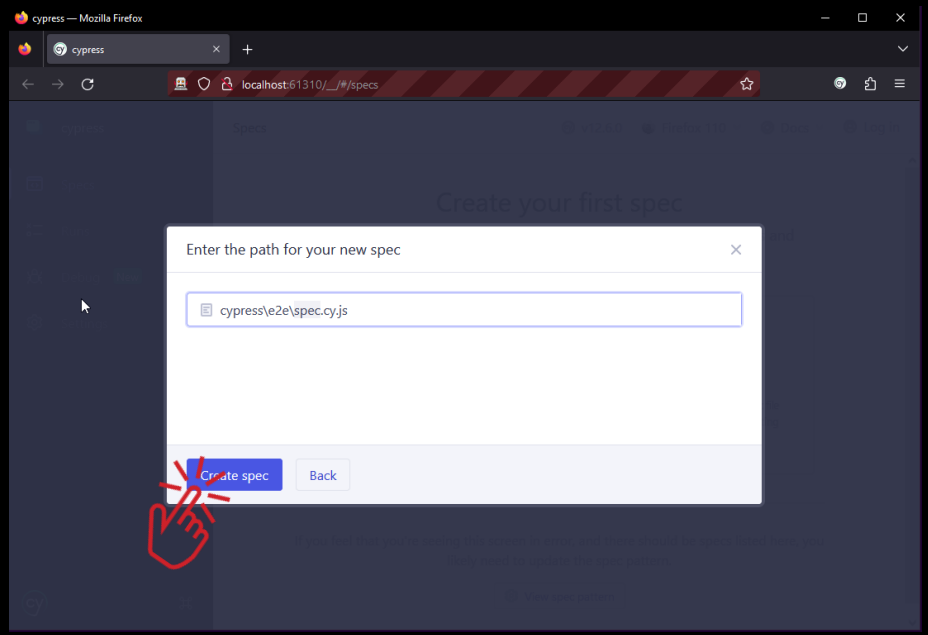
install

exercise 1



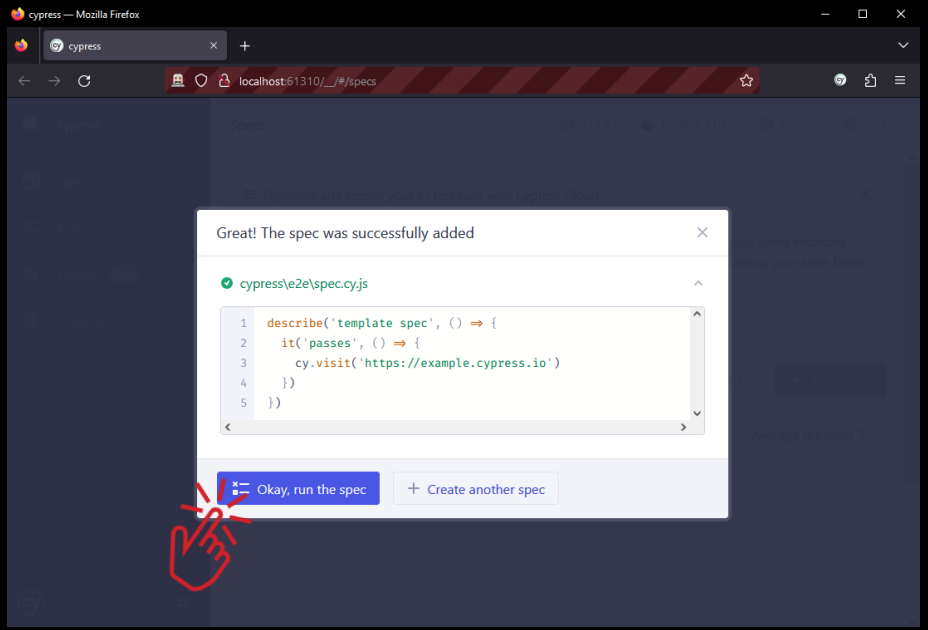
install

exercise 1



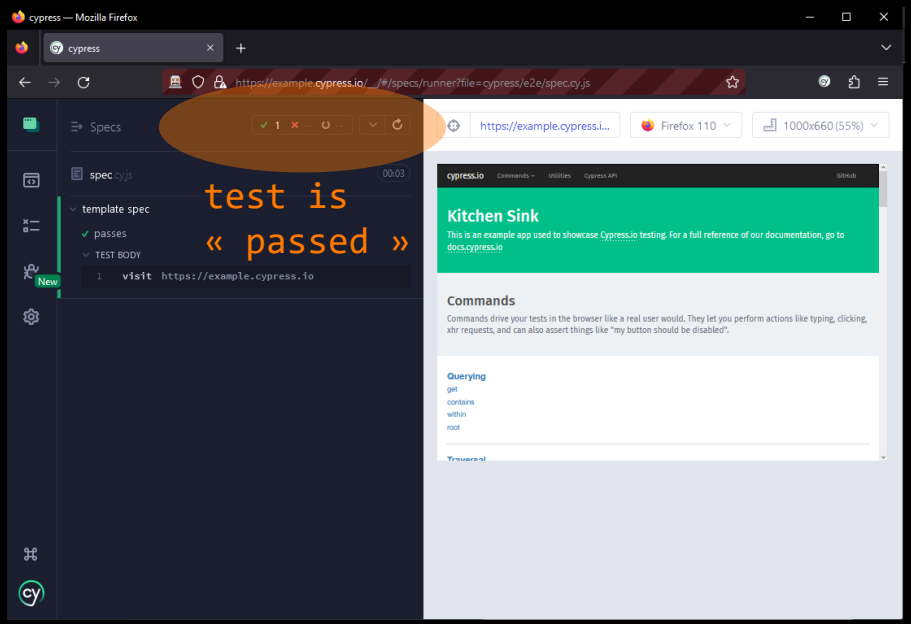
install

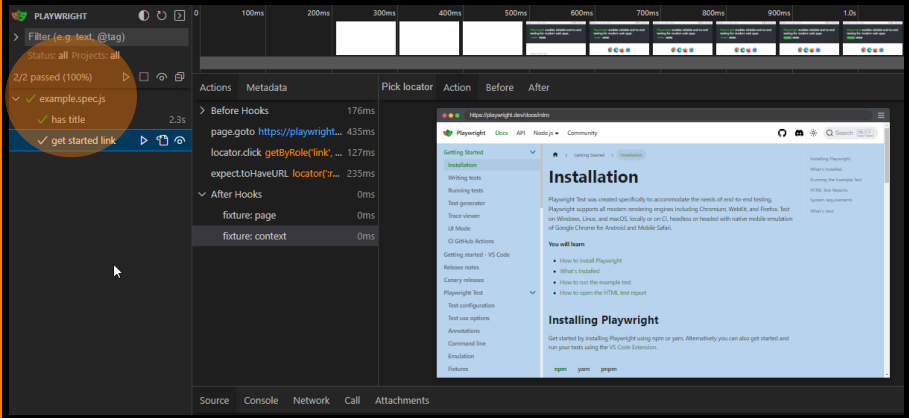
exercise 1



install

exercise 1





run headless

headless run will be used in a ci pipeline

exercise 2

goal : headless run

run headless


```
λ npx cypress run --e2e
--browser firefox
```

```
λ npx playwright test
--project=firefox
```

run headless

```
D:\dev\cypress
λ npx cypress run --e2e --browser firefox

(Run Starting)

  Cypress:        12.6.0
  Browser:        Firefox 110 (headless)
  Node Version:   v19.7.0 (C:\Program Files\nodejs\node.exe)
  Specs:          1 found (spec.cy.js)
  Searched:       cypress/e2e/**/*.cy.{js,jsx,ts,tsx}

Running:  spec.cy.js                                        (1 of 1)

template spec
  √ passes (1789ms)

1 passing (4s)

(Results)

  Tests:        1
  Passing:      1
  Failing:      0
  Pending:      0
  Skipped:      0
  Screenshots:  0
  Video:        false
  Duration:     3 seconds
  Spec Ran:     spec.cy.js

(Run Finished)

  Spec                                  Tests  Passing  Failing  Pending  Skipped
  √  spec.cy.js                 00:03       1        1        -        -        -
  √  All specs passed!          00:03       1        1        -        -        -
```

exercise 2

```
D:\dev\playwright  (playwright@1.0.0)
λ npx playwright test

Running 3 tests using 3 workers
  3 passed (9.1s)

To open last HTML report run:

  npx playwright show-report
```

load an url

load a url (web site or api route) with the browser

```
cy.visit('https://site.com');
    or
cy.visit('/');
```

```
await page.goto('https://site.com');
    or
await page.goto('/');
```

use default «baseUrl» defined in config

exercise 3

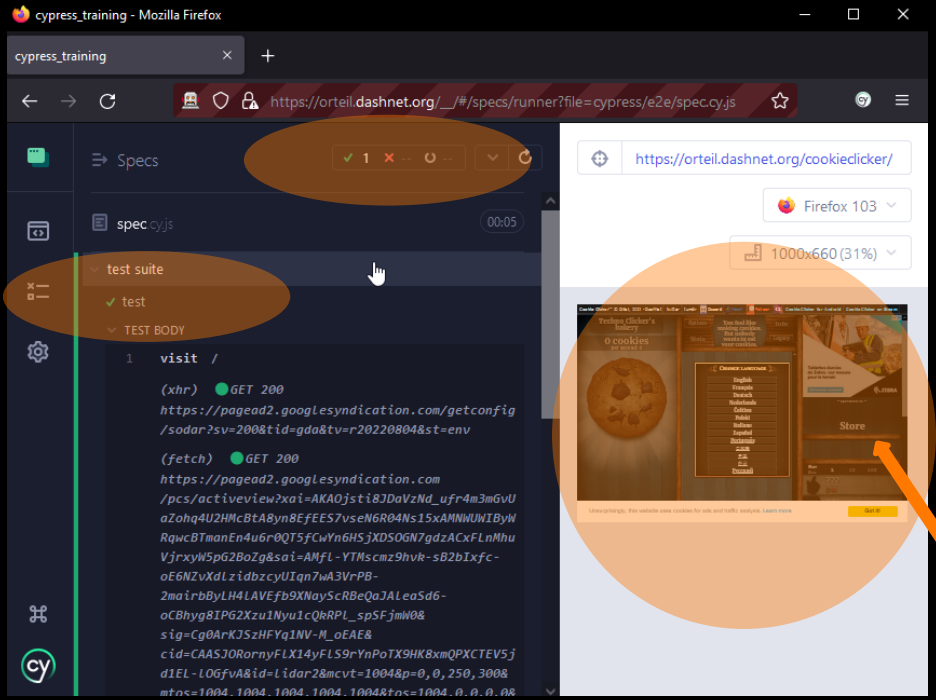
goal : load an url

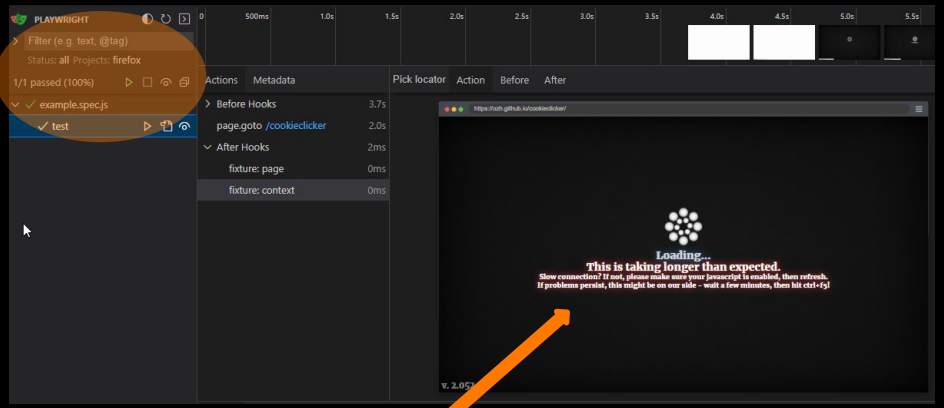
load an url


```
λ cat cypress/e2e/spec.cy.js
const url =
'https://ozh.github.io/cookieclicker';
describe('test suite', () =>
{
  it('test', () =>
  {
    cy.visit(url);
  });
});
```

```
λ cat tests/example.spec.js
const url =
'https://ozh.github.io/cookieclicker';
const { test, expect } =
require('@playwright/test');
test.describe('test suite', () =>
{
  test('test', async ({page}) =>
  {
    await page.goto(url);
  });
});
```

load an url





note the difference
Cypress ends the test when everything is loaded
Playwright ends the test when the « loaded » event is fired

exercise 3

get, click and auto retry

get a DOM element using a selector
retrying (no wait needed)

```
cy.get(selector);
```

```
await page.locator(selector);
```

click on an actionable element
retrying (no wait needed)

```
cy.get(selector).click();
```

```
await page.locator(selector).click(
```

exercise 4

goal : select a DOM element and click on it

get, click and auto retry

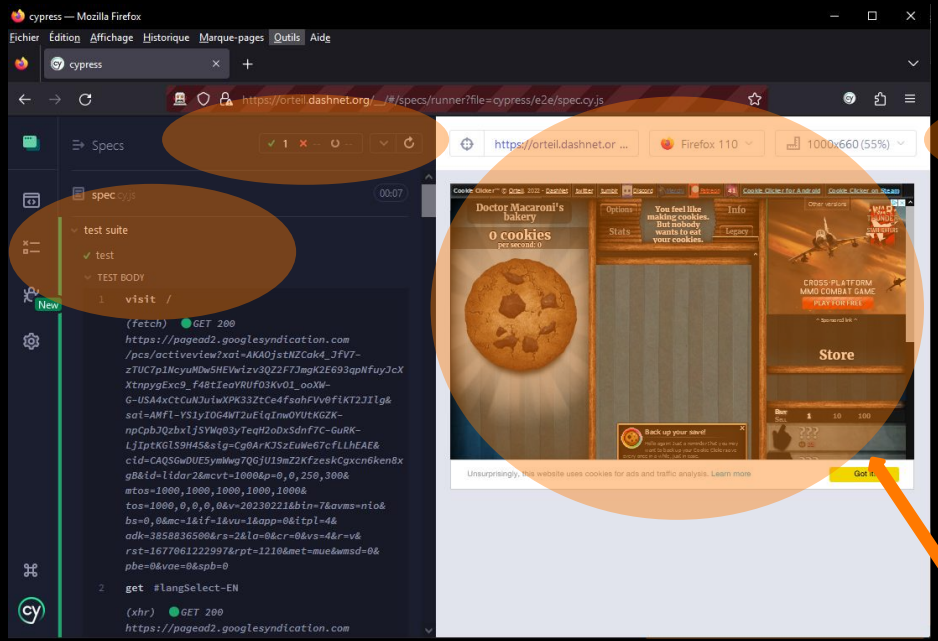
use specific test ids whenever it is possible
and encourage developers to add them or add them yourself

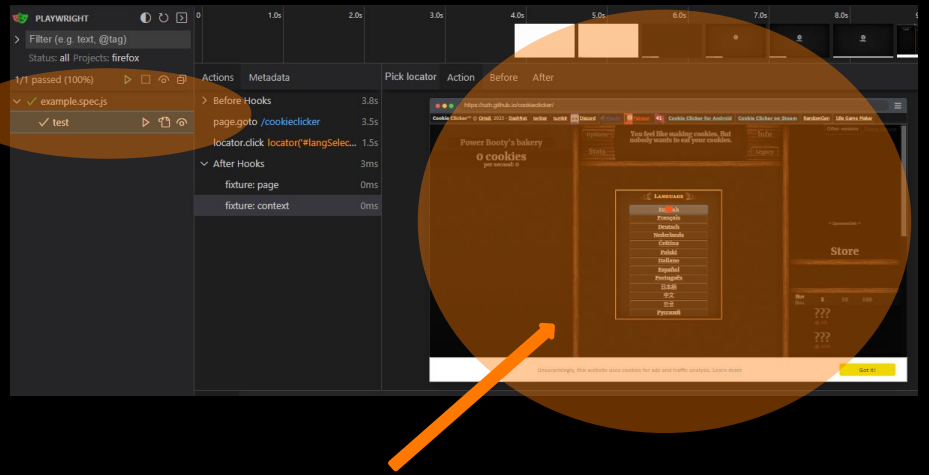
```
λ cat cypress/e2e/spec.cy.js
…
const lang = '#langSelect-EN';
const care = '.cc_btn_accept_all’;
…
describe('test suite', () =>
{
  …
  cy.get(lang).click();
  cy.get(care).click();
});
```

```
λ cat tests/example.spec.js
…
const lang = '#langSelect-EN';
const care = '.cc_btn_accept_all’;
…
test.describe('test suite', () =>
{
  …
  await page.locator(lang).click();
  await page.locator(care).click();
});
```

get, click and auto retry

exercise 4





again note the difference
Cypress ends the test when everything is loaded
Playwright ends the test immediately

asserting

asserting is a major feature of an automated testing tool

Cypress is providing the major standard assertion libs out of the box :

- chai (BDD and TDD)
- chai-jquery
- sinon-chai
- common assertions (should)

Playwright is providing only 1 build-in assertion system. Any standard other lib must be be downloaded and specifically imported.

exercise 5

goal : use an assertion

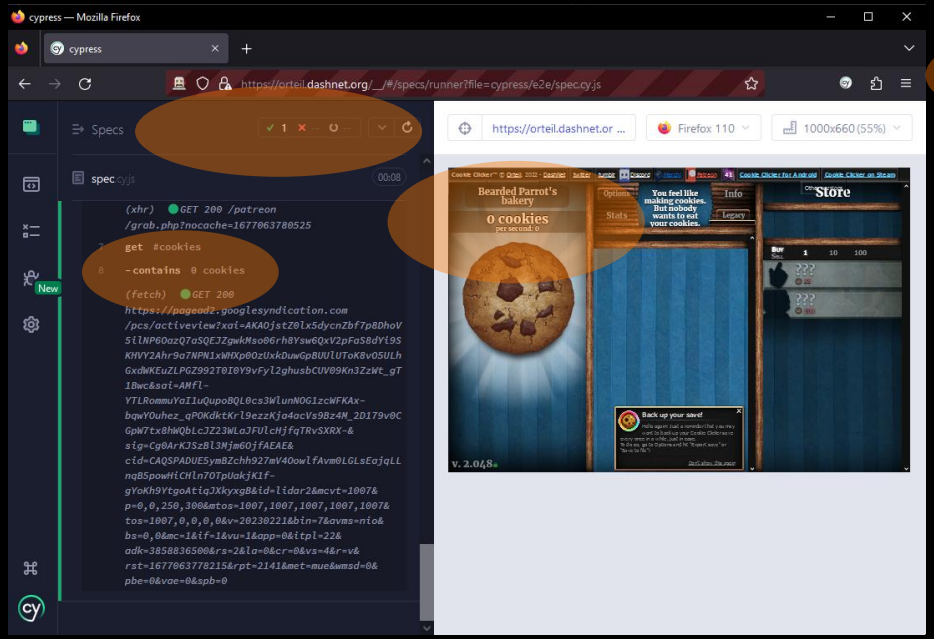
asserting

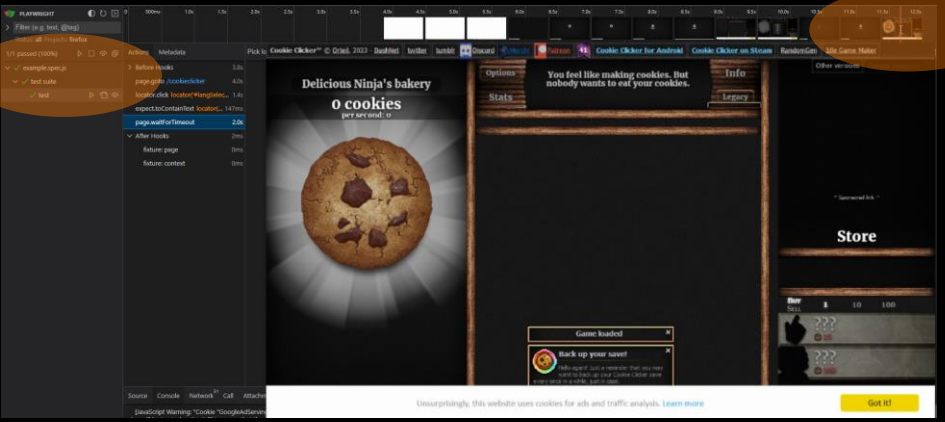
```
λ cat cypress/e2e/spec.cy.js
…
const nbcookies = '#cookies';
const zerocookies = '0 cookie’;
…
describe('test suite', () =>
{
  …
  cy.get(nbcookies).contains(zerocookie);
});
```

```
λ cat tests/example.spec.js
…
const nbcookies = '#cookies';
const zerocookies = '0 cookie’;
…
test.describe('test suite', () =>
{
  …
  await expect(page.locator(nbcookies))
  .toContainText(zerocookie);
});
```

asserting

exercise 5





test organization and context management

Cypress and Playwright are clearing the context by default

- the DOM
- the cookies
- the local storage

between each test !

test organization and context management

Cypress proposes 2 mechanisms for context management

session

tests are isolate by default, context is saved using a cy.session call and restored using another cy.session call

restore is taking a bit of time and the entire test must be in an it scope

test isolation

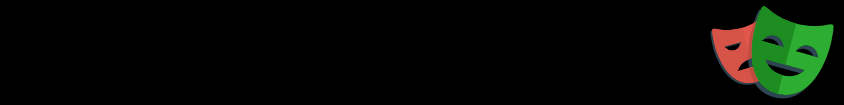
preferred solution

the user controls if Cypress clears the context or not for every test suite aka describe scope

test content can be splitted over several it calls

it runs faster

test organization and context management



Playwright proposes to use the beforeAll and afterAll functions to open and close the page.

Don't forgot to run the test suite in "serial mode" to avoid Playwright running the tests in parallel.

test organization and context management

```
describe('clear env', { testIsolation:true }, () =>
{
  it('clear', () => { cy.log(‘context cleared'); });
});

describe('testing Cookie Clicker', { testIsolation:false }, () =>
{
 it('test1’, () => { // do something });
 it('test2’, () => { // do something else in the same context });
});
```

clears context

keeps context

test organization and context management

```
test.describe('test suite', () =>
{
  test.describe.configure({ mode: 'serial' });
  let page;
  test.beforeAll(async ({ browser }) => { page = await browser.newPage(); });
  test.afterAll(async () => { await page.close(); });

  test('test1', async () => { // do something with page });
  test('test2', async () => { // do something else in the same context });
});
```

exercise 6

goal : organize tests and control context

test organization and context management

exercise 6

```
…
const bigcookie = '#bigcookie';
const onecookie = '1 cookie';
…
describe('clear env', { testIsolation:true }, () => {
  it('clear', () => { cy.log('env cleared'); }); });
describe('test suite', { testIsolation:false }, () => {
  it('init', () =>  {
    cy.visit(url);
    cy.get(lang).click();
    cy.get(care).click();
    cy.get(nbcookies).contains(zerocookie); });
  it('bake', { scrollBehavior: false }, () =>  {
    cy.get(bigcookie).click();
    cy.get(nbcookies).contains(onecookie); });
});
```

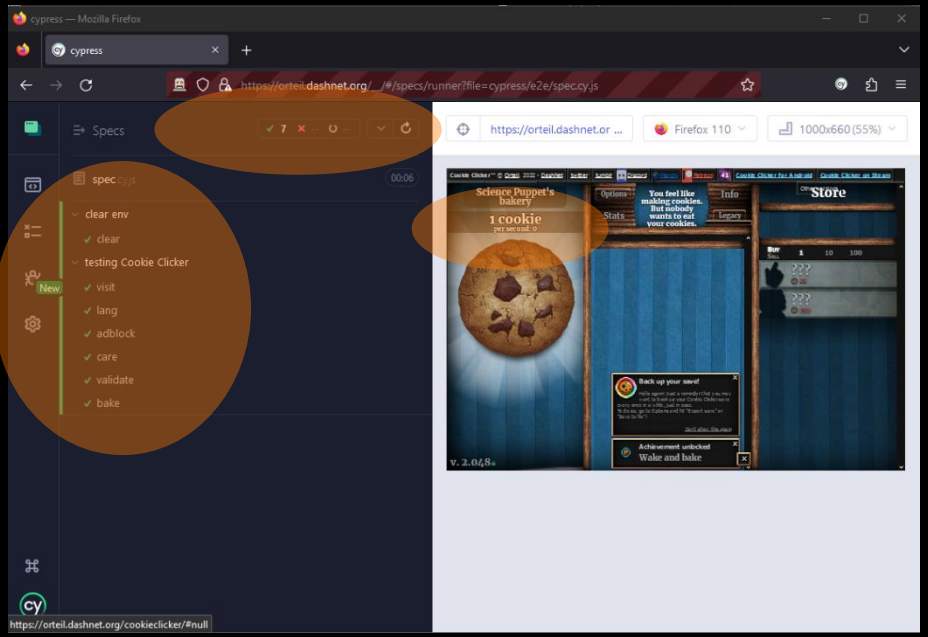
test organization and context management

exercise 6

```
…
const bigcookie = '#bigcookie';
const onecookie = '1 cookie';
…
test.describe('test suite', () => {
  test.describe.configure({ mode: 'serial' });
  let page;
  test.beforeAll(async ({ browser }) => { page = await browser.newPage(); });
  test.afterAll(async () => { await page.close(); });
  test('init', async () => {
    await page.goto(url);
    await page.locator(lang).click();
    await page.locator(care).click();
    await expect(page.locator(nbcookies)).toContainText(zerocookie); });
  test('bake', async () => {
    await page.locator(bigcookie).click();
    await expect(page.locator(nbcookies)).toContainText(onecookie); });
});
```

test organization and context management

exercise 6



keyboard interaction

whether it is for filling a form or any other keyboard usage

```
cy.get(selector).type('{ctrl}s');
```

```
await page.keyboard.press('Control+s');
```

press and type are different in Playwright !

exercise 7

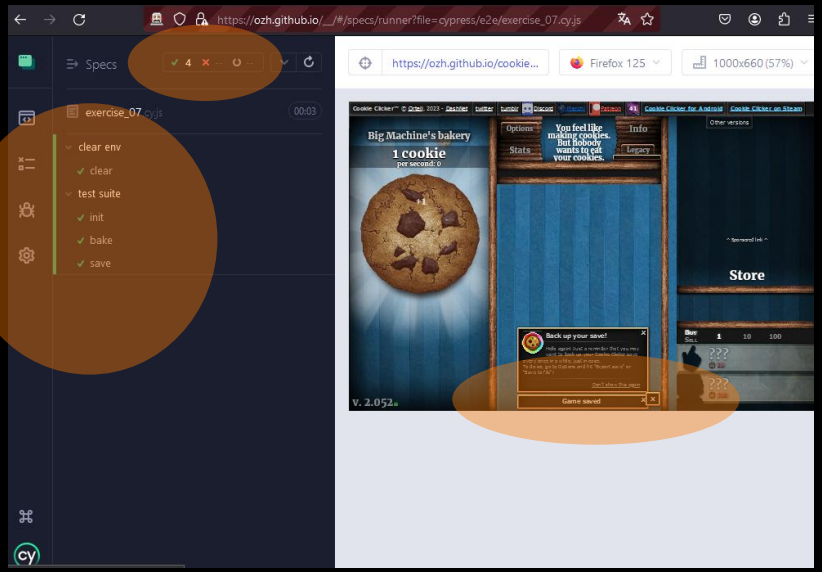
goal : simulate keyboard action

keyboard interaction

```
λ cat cypress/e2e/spec.cy.js
describe('test suite', () =>
{
  …
  it('save', () =>
  {
    cy.get('body').type('{ctrl}s');
  });
});
```

```
λ cat tests/example.spec.js
test.describe('test suite', () =>
{
  …
  test('save', async () =>
  {
    await
page.keyboard.press('Control+s');
  });
});
```

Cypress : keyboard interaction



mobile web

to simulate a mobile web device :

set the viewport size

```
cy.viewport(screen);
```

set the user agent

```
cy.intercept('/', (req) =>
req.headers['user-agent'] = useragent);
```

set the viewport size

```
 test.use({ viewport:
 { width: 123, height: 456 }});
```

set the user agent

```
test.use({ userAgent: useragent });
```

exercise 8

goal : simulate a mobile browser

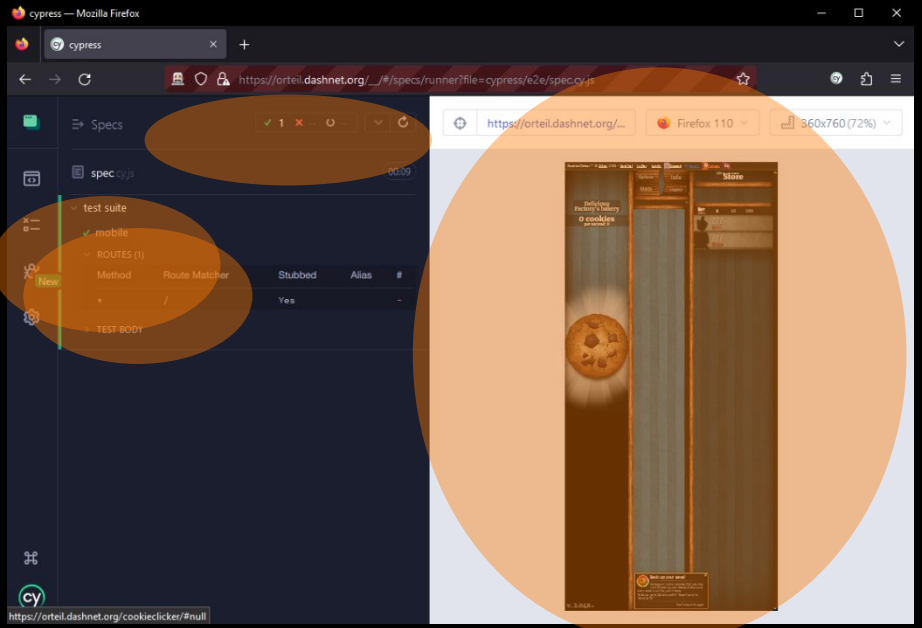
mobile web

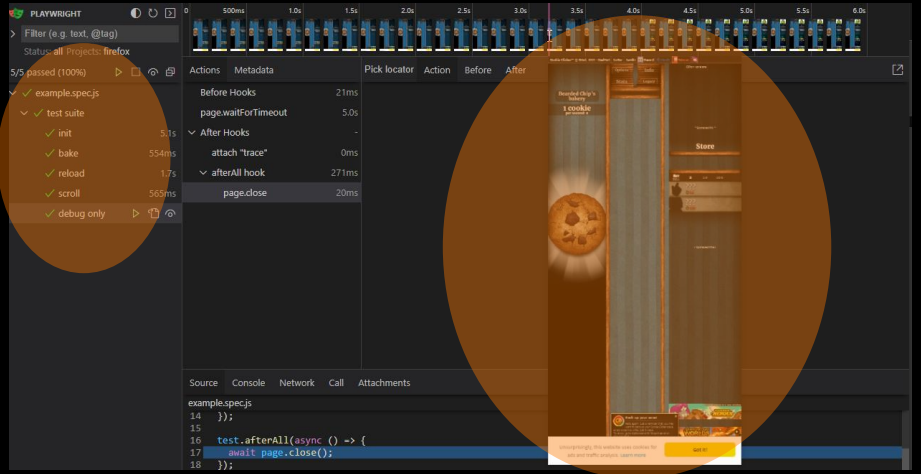

```
λ cat cypress/e2e/spec.cy.js

…

const config = {screen:'samsung-s10',
                useragent:"Mozilla/5.0 (Android 9;
Mobile; rv:97.0) Gecko/20100101 Firefox/97.0"};

…

describe('test suite', () => {
  it('mobile', () => {
    cy.viewport(config.screen);
    cy.intercept('/', (req) => req.headers['user-
agent'] = config.useragent);
    cy.visit(url);
    cy.get(lang).click();
    cy.get(care).click();
    cy.get(nbcookies).contains(zerocookie);
  });
});
```

```
λ cat tests/example.spec.js

…

const config = {screen:{width:360, height:760},
                useragent:"Mozilla/5.0 (Android 9;
Mobile; rv:97.0) Gecko/20100101 Firefox/97.0"};

…

test.describe('test suite', () => {
  test.use({ userAgent: config.useragent });
  test.use({ viewport: config.screen });
  test('test', async ({page}) =>  {
    await page.goto(url);
    await page.locator(lang).click();
    await page.locator(care).click();
    await expect(page.locator(nbcookies))
          .toContainText(zerocookie);
  });
});
```

mobile web

exercise 8





api testing

to test an api :

```
cy.request('GET', 'api.url/route')
.then((response) =>
{
  expect(response.status).to.eq(200);
}
```

```
const users=await request.get(url);
expect(users.ok()).toBeTruthy();
```

exercise 9

goal : api testing

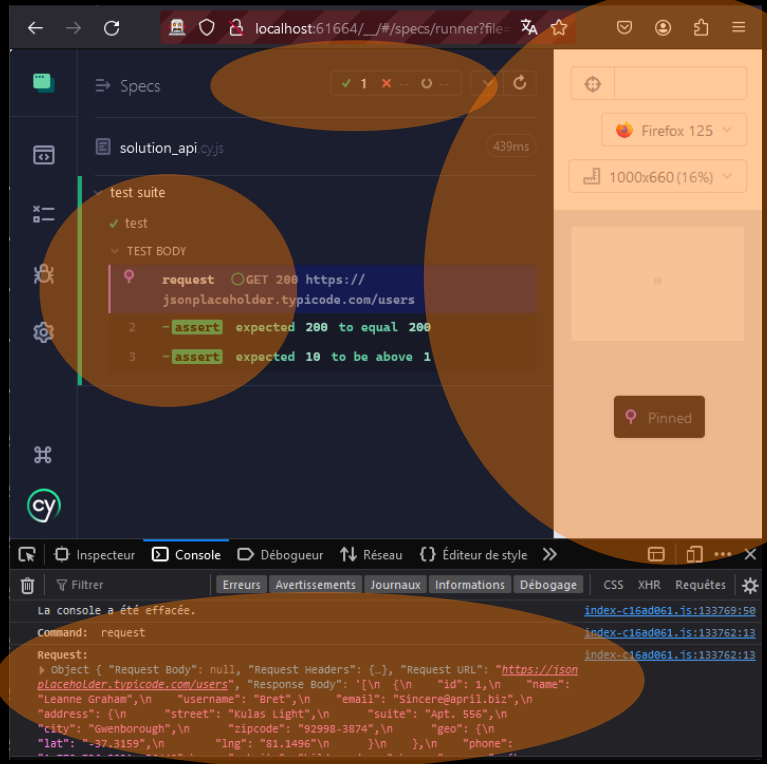
api testing

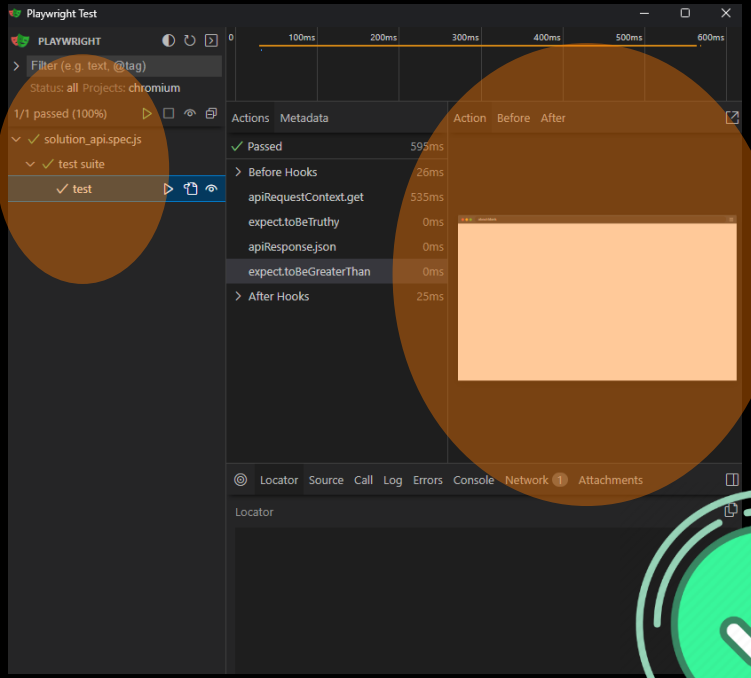

```
λ cat cypress/e2e/spec.cy.js
const
url='https://jsonplaceholder.typicode.com/users';

describe('test suite', () =>
{
  it('test', () =>
  {
    cy.request("GET", url).then((response) =>
    {
      expect(response.status).to.eq(200);
      expect(response.body.length)
                              .to.be.greaterThan(1);
    });
  });
});
```

```
λ cat tests/example.spec.js
const
url='https://jsonplaceholder.typicode.com/users’;
const {test, expect} = require('@playwright/test’);

test.describe('test suite', () =>
{
  test('test', async ({ request }) =>
  {
    const users = await request.get(url);
    expect(users.ok()).toBeTruthy();
    const users_json = await users.json();
    expect(users_json.length).toBeGreaterThan(1);
  });
});
```

api testing

exercise 9





containers, cloud and pipelines

Public docker images are
not up to date nor complete

Cypress docker public docker image is not up to date, old nodejs version, old cypress version and old browsers version

Playwright test is not available as a Docker image, only the Playwright library and browsers are

So let’s build them ourselves!

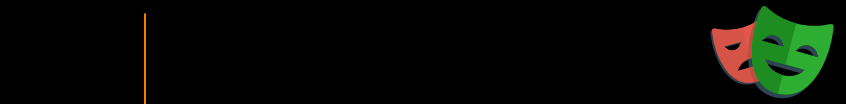
Build cypress and playwright images

Example with gitlab-ci (same for cypress and playwright)

```
λ cat .gitlab-ci.yml
include:
  - component: your.gitlab.url/to-be-continuous/docker/gitlab-ci-docker@5.10.1
    inputs:
      prod-publish-strategy: "auto"

stages:
      - build
      - package-build
      - package-test
      - publish
```

Build cypress and playwright images

```
λ cat Dockerfile
ARG NODE_VERSION='20.12.2'
ARG CYPRESS_VERSION='13.8.0'
ARG CHROME_VERSION='124.0.6367.60-1'
ARG EDGE_VERSION='124.0.2478.51-1'
ARG FIREFOX_VERSION='125.0.1'

FROM cypress/factory:3.5.4

COPY . /opt/app
WORKDIR /opt/app
RUN npm install --save-dev cypress@13.8.0
```



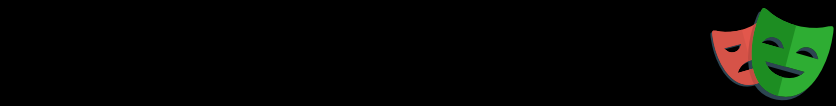
```
λ cat Dockerfile
FROM mcr.microsoft.com/playwright:v1.43.1-jammy
WORKDIR /app
RUN apt-get update && \
  curl –sL https://deb.nodesource.com/setup_20.x | bash - && \
  apt-get install -y nodejs && \
  npm cache clean --force && \
  npx create-playwright --quiet --lang=js --install-deps
```

Use those images to run tests

```
λ cat .gitlab-ci.yml
image: your.registry/cypress_build_docker
stages:
  - test
test:
   stage : test
   allow_failure: true
   parallel :
     matrix:
       - BROWSER:
         - chrome
         - firefox
         - edge
```

```
artifacts:
     when: always
     paths:
       - screenshots/
     name: "$CI_JOB_STARTED_AT"
   script:
     - cp -rf $PWD/cypress /opt/app
     - cp -f $PWD/cypress.config.js /opt/app
     - cd /opt/app
     - npx cypress run --headless --e2e --browser
$BROWSER
   after_script:
     - mv /opt/app/cypress/screenshots .
```

Use those images to run tests



```
λ cat .gitlab-ci.yml
image: your.registry/playwright_build_docker:main

stages:
  - test

test:
   stage : test
   allow_failure: true
   artifacts:
     when: always
     paths:
       - playwright-report/
       - test-results/
     name: "$CI_JOB_STARTED_AT"
   script:
     - cp -rf $PWD/tests /app
     - cp -f $PWD/playwright.config.js /app
     - cd /app
     - npx playwright test
   after_script:
     - mv /app/playwright-report .
     - mv /app/test-results .
```

containers, cloud and pipelines

demo

wrap-up

to wrap-up, I propose an exercice to simulate really playing Cookie Clicker ☺

exercise 10

goal : play Cookie Clicker

play Cookie Clicker

exercise 10

1- start with code for the exercice 6

2- click on the big cookie until you have enough cookies to buy a cursor, then buy the cursor

3- click on the big cookie until you have enough cookies to buy a grandma or an upgrade, then buy the latest

4- carry on until you have plenty of cookies ☺

Thanks