infoShare ACADEMY

Testy – Cypress i testy e2e aplikacji

infoShare Academy

infoShareAcademy.com

HELLO

Maciej Mikulski

Senior Front End Developer
@JIT.Team @Dialecticanet.com

infoShare ACADEMY

01. Instalacja

Dokumentacja i instalacja

https://docs.cypress.io/guides/overview/why-cypress

Instalacja Cypress'a

```
npm install cypress –save-dev
```

Uruchomienie w oknie przeglądarki

```
npx cypress open
```

Uruchomienie w trybie "headless"

```
npx cypress run
```

Zapisywanie wyników w Cypress Dashboard

```
Npx cypress run --record
```

Skrypty npm

Uruchom Cypress

```
“scripts”: {
  “cypress:open”: “cypress open”
}
```

Wykonaj testy w trybie headless/CI

```
“scripts”: {
  “cypress:run”: “cypress run”
}
```

Struktura folderów Cypress

Pierwsze utuchomienie

Npx cypress open

Cypress rozpoznaje czy testy są już skonfigurwane czy nie, pozwala nam wybrać podstawowe element konfiguracji oraz tworzy początkową strukturę plików.
Następnie pozwala utworzyć podstawowy test lub testy przykładowe.

cypress.config.js – plik konfiguracyjny

./cypress

- fixtures – zestawy danych wykorzystywane w testach
- support
  - commands.js – nasze własne komendy Cypress
  - e2e.js – skrypty uruchamiane przed wykonaniem testów – np. przygotowanie środowiska itp.
- e2e – właściwe testy

Założenia testów e2e

Cel testów end to end (e2e)

Testowanie kompletnego system

Testy e2e są zazywczaj najwolniejsze i “najdroższe” w przygotowaniu i wykonaniu, więc skupiamy się na testowaniu tego czego nie da się przetestować w testach integracyjnych i jednostkowych.

Testy uruchamiane są na kompletnym środowisku:

- Testowym
- pre-produkcyjnym (release candidate)
- docelowym - produkcyjnym

Smoke testy – testy głównych “szczęśliwych” scenariuszy potwierdzające działanie systemu

Testowanie regresji – testy dotychczasowych scenariuszy

03. Pierwszy test

Struktura testów

```
describe('nazwa scenariusz', () => {
        beforeEach(() => {
                // Uruchom przed każdym testem
        })

        it('opis testu', () => {
                // Zalecana struktura AAA
                // Arrange – przygotuj dane i "świat" do testów
                // Act – wykonaj czynność
                // Assert – zweryfikuj efekty wykonanej czynności
        })
})
```

Pierwszy test

```
describe('Strona główna', () => {
        it('Wita odwiedzającego', () => {
                // Arrange – przygotuj dane i "świat" do testów
                const header = 'Witaj świecie';
                // Act – wykonaj czynność
                cy.visit('https://localhost:3000');
                // Assert – zweryfikuj efekty wykonanej czynności
                cy.contains(header);
        })
})
```

Podstawowe komendy

cy.visit – przejdź do adresu (https://docs.cypress.io/api/commands/visit)

cy.visit('http://localhost:3000/login')

Po skonfigurowaniu bazowego adresu, nie musimy go podawać w każdym teście

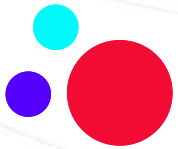
```
// cypress.config.js
const { defineConfig } = require('cypress')

module.exports = defineConfig({ e2e: { baseUrl: 'http://localhost:3000' } })

. // test.cy.js
cy.visit('/login')
```

cy.get – znajdź element na stronie

cy.get('.class') – używaj selektorów jak w jQuery



Podstawowe komendy – akcje

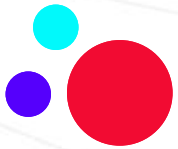
.type – wpisz tekst

cy.get('.login-input').type('login')

.click, .dblclick – kliknij element

cy.get('button').click()

infoShare ACADEMY



Relacje elementów

.first(), .last()

.parent(), .children()

.siblings()

.within()

Asercje

.contains() – znajduje i sprawdza zawartość pierwszego element
https://docs.cypress.io/api/commands/contains#Syntax

cy.contains('hello')

cy.get('.form').contains('username')

Oraz asercje znane z biblioteki Chai, Sinon I jQuery:

https://docs.cypress.io/guides/references/assertions#TDD-Assertions

.equal()

.isNull()

Które można łączyć wraz z powtózeniami i timeoutami:
.should('be.empty')
https://docs.cypress.io/api/commands/should#Syntax

Mockowanie API

Alternatywą do testowania pełnej aplikacji jest mockowanie API – czyli zastąpienie odpowiedzi serwera przygotowanymi w ramach testów przykładami.

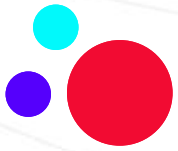
```
cy.intercept(
    {
      method: 'POST',
      url: 'http://localhost:3001/login',
    },
    {token: 'abc'
  ).as('login')
```

Własne komendy

Cypress pozwala na definiowanie własnych komend, które pozwalają nam łączyć i reużywać powtarzalne sekwencje, np. logowanie.

Dodaj w pliku:
cypress/support/commands.js

```
Cypress.Commands.add('login', (email, pw) => {})
Cypress.Commands.addAll({
  login(email, pw) {},
  visit(orig, url, options) {},
})
```



Operowanie środowiskiem

cy.exec() – uruchamia komendy systemowe

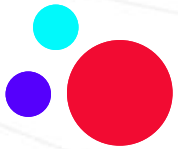
cy.task() – uruchamia komendy node.js

cy.request() – wykonuje requesty do API

infoShare ACADEMY

Cypress Testing Library

infoShareAcademy.com



React Testing Library w Cypress.js

https://testing-library.com/docs/cypress-testing-library/intro/

Instalacja

```
npm install --save-dev @testing-library/cypress
```

W cypress/support/command.js dodaj:

```
import '@testing-library/cypress/add-commands’
```

Przykłady użycia:

```
cy.findByRole('button', {name: /Jackie Chan/i}).click()
cy.get('form')
  .findByRole('button', {name: /Button Text/i})
  .should('exist')
```

infoShare ACADEMY

Testy “head-less”

CI/CD

Continuous Integration/Continuous Deployment

We współczesnym procesie wytwarzania oprogramowania, wartością jest skrócenie czasu pomiędzy implementacją wybranej funkcjonalności, a wdrożeniem jej do działającej aplikacji. Wymaga to automatyzacji procesu integracji zmian (CI) oraz ich wdrożenia (CD).

Aby zagwarantować bezpieczeństwo i jakość wprowadzanych zmian, proces musi zawierać w sobie mechanizmy gwarancji jakości (Quality Assurance) m. in. autmatyczne testy.

Testy automatyczne powinny być uruchamiane najlepiej:

- Przed zintegrowaniem zmian – czyli na branchu zawierającym Merge Request
- Przed wdrożeniem zmian – czyli na branchu głównym, przed wdrożeniem (deployment)
- Po wdrożeniu – na środowisku docelowym, z możliwością automatycznego wycofania zmian

Uruchamianie Cypress'a w CI/CD

Cypress może być uruchamiany w trybie "head-less", czyli bez podglądu interfejsu w przeglądarce.

npx cypress run

Możemy uruchamiać lokalnie – jeśli chcemy uruchomić wszytskie testy w celu weryfikacji regresji.

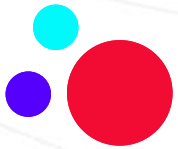
Testy uruchamiane automatycznie w procesie CI/CD za pomocą narzędzi takich jak Jenkins/ GitLab i inne.

Często wykorzystwane są kontenery Docker'a do dostarczenia aplikacji I gotowego środowiska uruchomienia testów.

Testy mogą raportować wyniki bezpośrednio lub do Cypress Dashboard.

Testowanie API

infoShareAcademy.com



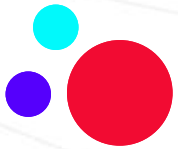
Testy API za pomocą Cypress

Poza testami UI Cypress umożliwia testowanie API poprzez bezpośrednią komunikację z API aplikacji.

```
cy.request('POST', 'http://localhost:8888/login', { username: wiki’, password: ‘tajneHasło’ }).then(
  (response) => {
    // response.body is automatically serialized into JSON
    expect(response.body).to.have.property('name', ‘Wiktoria') // true
  }
)
```

Testowanie komponentów

infoShareAcademy.com



Testy komponentów React za pomocą Cypress

https://docs.cypress.io/guides/component-testing/quickstart-react#Configuring-Component-Testing

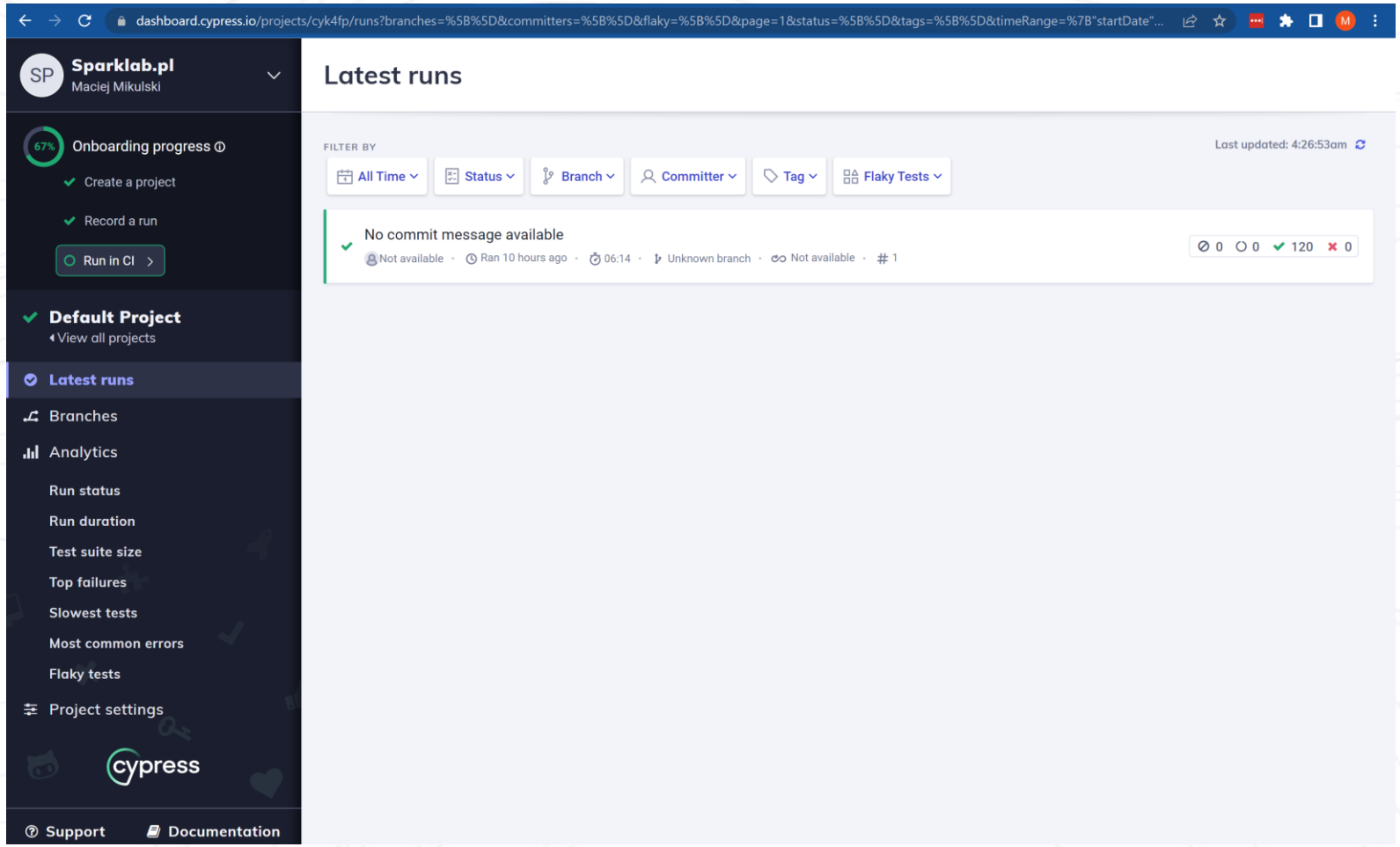
https://www.cypress.io/blog/2021/04/06/cypress-component-testing-react/

infoShare ACADEMY

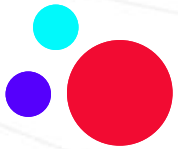
Cypress Dashboard

infoShareAcademy.com

Cypress Dashboard



infoShare ACADEMY



Cypress Dashboard

- Raportowanie i śledzenie wykonań testów
- Wyłapywanie kłopotliwych i wolnych testów

- Dostępne w wersji płatnej oraz darmowej

- Alternatywna wersja – ekonomiczna:
- https://sorry-cypress.dev/

Dziękuję za uwagę!

infoShareAcademy.com