CYPRESS.IO & E2E TESTS

https://github.com/przemyslawjanpietrzak

Rembrandt.re

```
open Rembrandt.Elements;

let update = (model: model, action: action):
(model, Command.command('action)) =>
  switch (action) {
  | Add => (model + 1, Command.null)
  | Sub => (model - 1, Command.null)
  | Twice => (model + 1, Command.action(Add))
  };

Rembrandt.run(
  ~model=42,
  ~update,
  ~view=
    (model, dispatch) =>
      <div>

        <div id="count"> {string_of_int(model) |>
```

Table of Contents

History

* Static web apps
* Selenium (2004)
* jQuery (2006)
* Backbone (2009)
* Cypress.io

*** Open source (runner)**

*** Written in CoffeeScript**

*** Based on Electron and Chromium**

End-to-end testing is a methodology used to test whether the flow of an application is performing as designed from start to finish. The purpose of carrying out end-to-end tests is to identify system dependencies and to ensure that the right information is passed between various system components and systems.

Chapter I
Overview

Getting started

```
npm install cypress --save-dev
npx cypress init
```

Files

```
├── fixtures
│   └── recruitment.json
├── integration
│   ├── recruitment.spec.js
├── plugins
│   └── index.js
├── screenshots
├── support
│   ├── commands.js
│   ├── index.js
│   ├── recruitment.js
└── utils.js
```

Test

```
it('user should be able to create new
recruitment', () =>
  cy
    .get('#new-recruitment-button').click()
    .get('#new-recruitment-
section').should('exist')
    .get('#name').type(name)
    .get('#description').type(description)
    .get('#supervisors').select('first
supervisor')

    .get('#submit-button').click()


    .get('.error').should('not.exist')
```

Command (page object)

```
Cypress.Commands.add('login', (username, password)
=> {
  cy
    .get('#login').should('exist')
    .get('#username').type(username)
    .get('#password').type(password)
    .get('#submit-button').click()

    .get('#login').should('not.exist')
    .get('#dashboard').should('exist'));
});
```

```
.login('admin', 'admin')
```

Fixture

```
{
  "data": {
    "id": "42",
    "name": "John Doe"
    "status": true,
    "isAdmin": false
  },
  "status": 200
}
```

```
response: 'fixture:new-chapter',
```

Demo #1

Chapter II
Test what/how?

What?

* Business logic
* Positive paths
* DB modifications

Why?

*** E2E tests are costly**

*** Required much time to run**

*** Have to be supported well**

How?

Data aria

```
cy.get('.button-blue').click()
```

```
cy.get('[data-aria="submit-button"]').click()
```

Assertion messages

Expect 42 to equal 41 ???

```
it('create new candidate', () => {
  cy
    .createCandidate(someData)
    .goToCandidatesList()
    .getCandidatesNumber().equal(candidates + 1,
'new candidate was NOT added to list')
  ;
});
```

No imperative wait

```
cy
  .doSomething()
  .wait(30000)
  .doSomething()
```

```
cypress run --config defaultCommandTimeout=10000
```

Random data

```
it('Update candidate data', () => {
  const name = generateRandomString();
  const description = generateRandomString(40);
  cy
    .goToFirstCandidate()
    .editCandidate()

    .get('#name').type(name)
    .get('#description').type(description)
    .click('#saveRecruitmentBottom')

    .refresh()
    .get('#name').should('equal', name)
    .get('#description').should('equal',
```

Separate it

*** Each test scenario must have his own data**

*** Prepare mocked database and reset it before test run**

***? Mock all endpoints**

Chapter III
HTTP spy

Prepare route

```
beforeEach(() => {
  cy.server({ delay: 1000 });
  cy.route('GET', 'candidates',
'fixture:candidates.json')
});
```

Demo #2

Prepare route

```
beforeEach(() => {
  cy.server({ delay: 1000 });
  cy.route({
    method: 'POST',
    url: '/api/candidates/42',
    response: 'fixture:candidate.json',
    onRequest: ({ request }) => {
      Object.assign(requests, { createCandidate:
request });
    },
  });
});
```

Assertion

```
it('create new candidate should send proper
request', () => {
  cy
    .createCandidate(someData)

    .assertEqual(requests, 'createCandidate', {
      username: 'John Doe',
      email: 'john@doe.com',
      idAdmin: false
    })
});
```

Chapter IV

CI

Docker

Docker

```
docker pull cypress
docker run
  --volume=$(pwd):/src
  --network=host
  --memory=4g
  --cpus=2
  cypress -c bash "npx cypress run"
```

Run backend

```
(npm run backend & echo $! > backend.pid & (sleep
42 && npm run cypress))
kill $(echo backend.pid)
```

Niektóre kalumnie brzmią dumnie

* Java is dead [*]
* REST is dead [*]
* Redux is dead [*]
* Selenium is dead [*]

SELENIUM PROS

* Any browser
* Any language
* Selenium to Selenium
* Easy to distributed

CYPRESS PROS

* Great debug
* Great async handling
* Backend mocks
* Easy to start
* Battery included

Thank you :*