

Primeiros passos com Cypress

Carolina Santana Louzada

Analista QA - Venturus

Objetivo do curso

Um curso com foco em mostrar a base necessária para fazer seus primeiros testes automatizados usando Cypress. Serão abordados o surgimento, vantagens, desvantagens, aspectos diferenciais em relação ao tão usado Selenium e conceitos iniciais para quem ainda está caminhando para entender esse framework tão crescente no mercado.

Pré-requisitos

- Fundamentos de qualidade de software
- Fundamentos de automação de testes
- Git
- Conhecimentos básicos em Javascript
- Conhecimentos básicos em CLI

Percurso

Aula 1

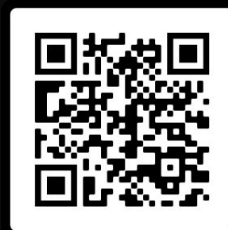
O que é Cypress?

Aula 2

Conceitos iniciais para automação E2E com Cypress

Dúvidas durante o curso?

- > Fórum do curso
- > Comunidade online (Discord)



SCAN ME

Aula 1

O que é Cypress?

// Primeiros passos com Cypress

Objetivos

1. Contextualização sobre surgimento
2. Instalação e configuração
3. Abertura e execução de testes
4. Conhecer o projeto a ser automatizado

Aula 1 . Etapa 1

Surgimento

// Primeiros passos com Cypress



O que é Cypress?

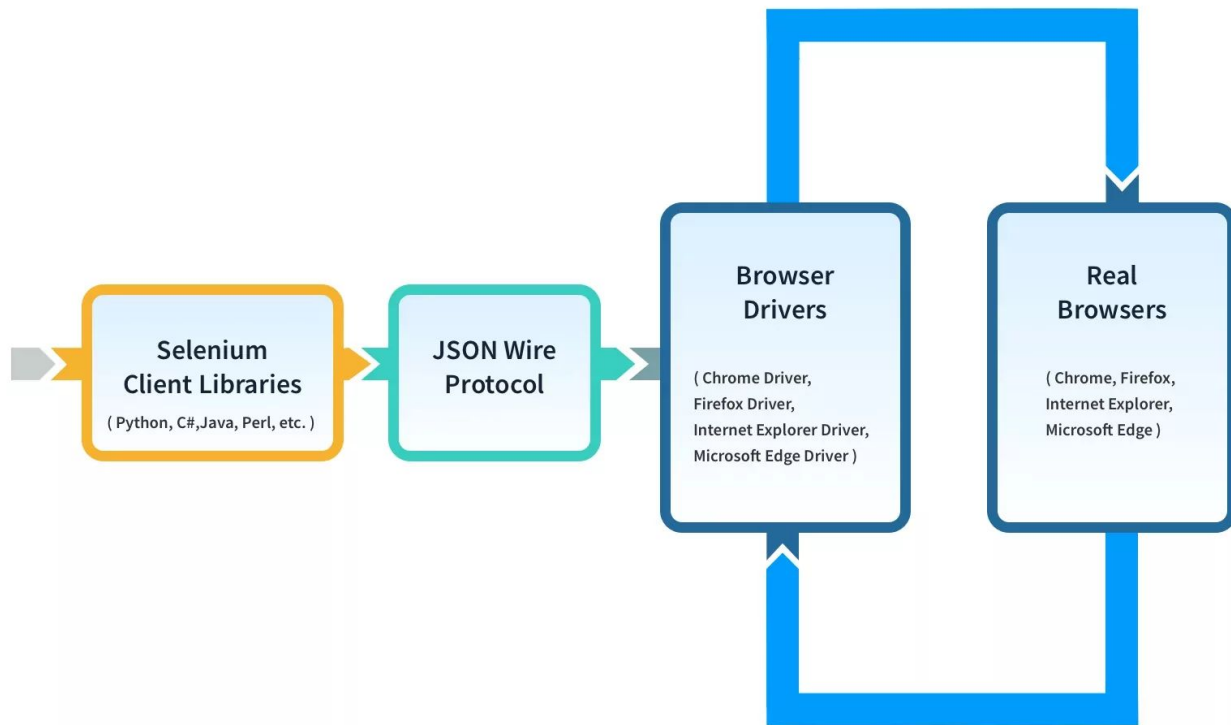
- ★ Ferramenta de auxílio à criação, configuração, execução e depuração de testes automatizados para aplicações modernas
- ★ É executado diretamente no browser, portanto é um processo servidor node, no qual temos acesso inclusive a ferramentas e ações do browser.



Como surgiu?

- ★ Surgiu em meio à **insatisfação** de engenheiros de software e QAs devido às limitações do Selenium e a necessidade cada vez maior do time de QA com desenvolvedores. **Agora os devs também se importam com toda a pirâmide de testes!**

Vamos relembrar sobre o Selenium?



Fonte: BrowserStack(2023)

Limitações gerais do Selenium

- ★ Configurações para testes de acordo com o ambiente podem ser complexas
- ★ Pré-condições de testes pode ser de difícil configuração
- ★ Não há geração automática de report
- ★ Problemas com uso de espera de elementos gerando testes não-confiáveis
- ★ Foco em testes E2E

Missão



- ★ Ecossistema open-source
- ★ Aumento de produtividade
- ★ Melhor experiência com testes
- ★ Boa documentação
- ★ Integração entre desenvolvedores + QA



Qual o público-alvo?

- ★ Desenvolvedores(principalmente frontends)
- ★ Engenheiros de automação / QA

**Foco no uso de aplicações modernas
com Javascript!**



Níveis de testes

- End-to-end
 - ◆ realizar ações via UI
- Componentes
 - ◆ testes em componentes front-end de bibliotecas conhecidas (React, Angular, Vue, Svelte)
- Integração/API
 - ◆ uso de chamadas HTTP
- Unidade



Funcionalidades

- *Time Travel*
- Depuração
- Espera automática
- *Spies, Stubs and Clocks*
- Controle de tráfego de rede
- Resultados consistentes
- *Screenshots/Videos*
- *Cross-browsing*
- Orquestração inteligente
- Detecção de testes não-confiáveis

Aula 1 . Etapa 2

Instalação e configuração de ambiente

// Primeiros passos com Cypress

Requerimentos



- macOS 10.9 +
- Linux Ubuntu 12.04 | Fedora 21 | Debian 8 e acima
- Windows 7 +
- Node.js (14 / 16 / 18) - > usar LTS
- 4GB de memória RAM no mínimo

Instalação



1. Crie seu projeto -> npm init
2. Acesse a pasta do projeto
3. Use o comando a seguir para instalar o cypress no projeto

```
npm install cypress --save-dev
```

Abrindo o App



- Na linha de comando digite o seguinte para abrir o cypress

```
npx cypress open
```

- Usando npm Scripts

```
{  
  "scripts": {  
    "cypress:open": "cypress open"  
  }  
}
```

```
npm run cypress:open
```

Abrindo o App



- Após a primeira abertura do Cypress, a configuração default do projeto aparecerá na pasta.
- Não esqueça de adicionar o .gitignore do projeto. Utilize o seguinte site se precisar de um template inicial - > gitignore.io - [Crie Arquivos .gitignore Úteis Para Seu Projeto. \(topical.com\)](https://topical.com)

Aula 1 . Etapa 3

Executando meu primeiro teste

// Primeiros passos com Cypress

Launchpad



- Guia de decisões e configurações para iniciar projetos de testes automatizados
- 1. Tipo de teste a ser desenvolvido
 - a. E2E
 - b. Component
- 2. Configurações rápidas de inicialização
- 3. Escolher um browser para executar testes de forma visual
- 4. Selecionar o teste para execução

Executando meu primeiro teste



- O Cypress nos ajuda a criar nosso primeiro teste
 - ◆ Você pode optar por escolher um conjunto de exemplos
 - ◆ Você pode optar por criar uma única spec com um *template* simples

O que é importante notarmos na UI de execução?

- Menu para trocar de browser
- Menu de acesso à documentação
- Opção para referência da versão do Cypress sendo usada
- Página de Specs
 - ◆ Opções de criação de testes
- Página de Configuração
 - ◆ Configurações de projeto
 - ◆ Configurações de dispositivo
 - ◆ Configurações para integração com Cypress Cloud



O que é importante notarmos na UI de execução?

- Página 'Runs'
 - ◆ Necessária integração com Cypress Cloud para visualizar as execuções de testes gravadas em nuvem
- Página 'Debug'
 - ◆ Necessária integração com Cypress Cloud para visualizar execuções de testes e depurar problemas ocorridos em pipeline



Aula 1 . Etapa 4

Conhecendo a demo para automação

// Primeiros passos com Cypress



Processo de acompanhamento

1. Acompanhe o vídeo e tente fazer sozinho. O site demo que usaremos para aprendizagem será [Automation Exercise](#).
2. Acesse os links para documentação do Cypress. Leia e releia, procure outras referências, pois a documentação é bem rica!
3. Caso tenha alguma dúvida (Tente primeiro sozinho em!) acesse o link [digitalinnovationone/cursoPrimeirosPassosCypress\(github.com\)](https://digitalinnovationone.com.br/curso/primeiros-passos-cypress/) e compare o código.
4. O repositório está dividido em branches que representam a etapa das aulas. Basta acessar a *branch* relativa à etapa que você está de forma a entender em que ponto cada aula parou no código.

Formato das branches de acompanhamento

1. **Main** -> comportará o código completo
2. **aula1-etapaX** -> equivale ao código gerado da aula e etapa em questão

Aula 2

Conceitos iniciais para automação E2E com Cypress

// Primeiros passos com Cypress

Objetivos

1. Aprender a acessar e interagir com elementos e requisições de uma página
2. Aprender a fazer validações/asserções
3. Entender o encadeamento de comandos
4. Interagir via linha de comando

Aula 2 . Etapa 1

Acessando e interagindo com elementos da página

// Primeiros passos com Cypress



Como Cypress acessa os elementos da página?

→ Incorpora sintaxe **jQuery** para acessar estruturas da página

```
$('.my-selector')
```

```
cy.get('.my-selector')
```

Como Cypress acessa os elementos da página?



→ Alguns métodos úteis para aprender inicialmente:

- ◆ get
- ◆ find
- ◆ contains
- ◆ type
- ◆ click



Melhores práticas

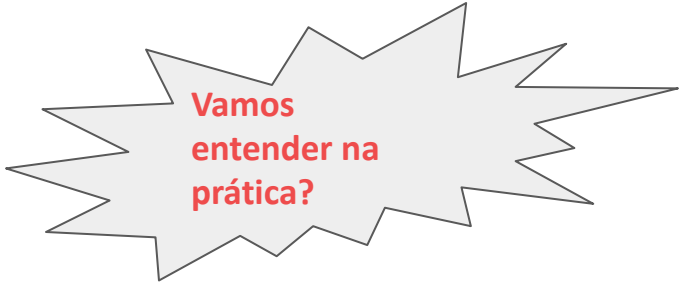
- Uso de ***data-**** para identificar elementos
- Evite seletores que tendem a mudar frequentemente = complexidade para manutenção dos testes
- Evite uso de seletores com base em textos
- Explore o playground do Cypress para aprender sobre os elementos

E onde entram os testes de API?

- Podemos fazer validações a partir dos métodos
 - ◆ **Intercept** -> Intercepta uma chamada feita pela aplicação
 - ◆ **Request** -> Faz chamada HTTP

[Network Requests | Cypress Documentation](#)

[intercept | Cypress Documentation](#)



Vamos
entender na
prática?

Aula 2 . Etapa 2

Entendendo validações e asserções

// Primeiros passos com Cypress



O que são asserções?

- São 'perguntas' que fazemos a respeito de um atributo ou estado de um elemento de forma que possamos validar o que é esperado do que é obtido no contexto de um teste.
- Cypress espera automaticamente até que os elementos alcancem o estado ou falha o teste caso contrário.



O que são asserções?

```
cy.get('button').click()  
cy.get('button').should('have.class', 'active')
```

```
cy.get(':checkbox').should('be.disabled')  
  
cy.get('form').should('have.class', 'form-horizontal')  
  
cy.get('input').should('not.have.value', 'US')
```

Quando devo fazer uma asserção?

- Sempre que queremos validar **explicitamente** o estado de um elemento.
- Atenção: Com Cypress, porém, é possível fazer validações **implícitas** a partir dos próprios comandos embutidos no cypress.

Asserções padrões embutidas em comandos

- `.visit()` -> conteúdo é retornado com status code 200
- `.contains()` -> conteúdo procurado existe na DOM
- `.get()` -> elemento existe
- `.click()` -> elemento é clicável

Não é necessário validar a existência de um elemento, pois todas as consultas validam automaticamente!

Modelos de asserções

- Cypress carrega em seu framework bibliotecas de asserções:
 - ◆ **Chai**
 - ◆ Chai-Sinon
 - ◆ **Chain-jQuery**
- Também é possível criar novas asserções!

Modelos de asserções

→ Sujeitos implícitos

```
cy.get('tbody tr:first').should('have.class', 'active')
```

◆ .should() e .and()

→ Sujeitos explícitos

```
expect(true).to.be.true
```

◆ expect

```
cy.get('.connectors-list > li').should(($lis) => {  
  expect($lis, '3 items').to.have.length(3)  
  expect($lis.eq(0), 'first item').to.contain('Walk the dog')  
  expect($lis.eq(1), 'second item').to.contain('Feed the cat')  
  expect($lis.eq(2), 'third item').to.contain('Write JavaScript')  
})
```

Auxiliares na validação - aliases e .then()

Muitas vezes precisamos usar variáveis ou constantes em nossas validações, contudo com Cypress essa necessidade é reduzida pela forma que os comandos se comunicam.

- ★ Uma forma recomendada para compartilhar elementos/contextos é através do uso dos comandos **.then()** e **.as()**

Auxiliares na validação - aliases e .then()

★ *.then()*

- comando que permite retomar o sujeito repassado pelo comando anterior, evitando assim uso de variáveis
- não é recomendado usar outros comandos aninhados com ele, mas sim dentro do contexto do próprio comando

```
cy.get('.nav').then(($nav) => {}) // Yields .nav as first arg  
cy.location().then((loc) => {}) // Yields location object as first arg
```

Auxiliares na validação - aliases e .then()

★ *.as()*

- comando que gera aliases e permite compartilhar contextos, fixtures, requisições e variáveis dentro e fora do escopo do teste

```
it('disables on click', () => {  
  cy.get('button[type=submit]').as('submitBtn')  
  cy.get('@submitBtn').click().should('be.disabled')  
})
```

Aula 2 . Etapa 3

Entendendo encadeamento de comandos

// Primeiros passos com Cypress

O que é encadeamento de comandos? Por que importa?

- Uso aninhado e/ou encadeado de métodos subsequentes, no qual o primeiro comando gera algo a ser passado ou não para o próximo comando.
- Importante entender o mecanismo para não ter erros lógicos ou de ordem de execução nos testes

```
cy.get('textarea.post-body').type('This is an excellent post.')
```


O que é encadeamento de comandos? Por que importa?

- As validações também são feitas de forma encadeada, de modo que o cypress espera cada asserção passar ou falhar

```
cy.get(':checkbox').should('be.disabled')  
  
cy.get('form').should('have.class', 'form-horizontal')
```

Considerações sobre encadeamento de comandos

- A cadeia sempre começa com um **'cy.[command]'**, no qual o sujeito repassado neste comando estabelece quais comandos podem ser chamados em seguida.
- Todos os comandos passam valores, podendo ser ***null*** inclusive.
- Alguns comandos requerem um sujeito anterior para funcionar
 - ◆ `click()`
 - ◆ `contains()`
 - ◆ `find()`

Considerações sobre encadeamento de comandos

- Os comando que passam sujeitos 'nulos' podem ser encadeados sem grandes prejuízos

```
cy.clearCookies() // Yields null
.visit('/fixtures/dom.html') // Does not care about the previous command

cy.get('.main-container') // Yields an array of matching DOM elements
.contains('Headlines') // Yields the first DOM element containing the text
.click() // Yields same DOM element from previous command.
```

E se eu quiser acessar diretamente o sujeito repassado por um comando?

- Para isso nós temos o querido `‘.then()’`
- Basta encadearmos o `‘.then()’` de forma subsequente ao comando, que ele chamará uma função callback com o sujeito repassado pelo comando.

```
cy.get('button').then(($btn) => {  
  const cls = $btn.attr('class')  
  
  cy.wrap($btn).click().should('not.have.class', cls)  
})
```

Aula 2 . Etapa 4

Cypress CLI

// Primeiros passos com Cypress



Usando a linha de comando com Cypress - Abrindo o App

- No início da automação utilizamos bastante a interface gráfica do Cypress para nos ajudar na análise e depuração dos nossos testes automatizados. Lembra qual comando usamos para abrir?

```
cypress open [options]
```



Executando nossos testes sem UI

- Contudo, durante a execução completa da suíte de forma local ou na pipeline, torna-se inviável o uso por meio da interface gráfica, portanto precisamos utilizar um comando para execução via prompt.

```
{  
  "scripts": {  
    "cy:run": "cypress run"  
  }  
}
```



Executando nossos testes sem UI

- Existem muitas configurações e argumentos possíveis de serem utilizados como opções do comando, logo é bem comum definirmos scripts para mapear essas situações.

[Command Line | Cypress Documentation](#)

```
{
  "scripts": {
    "cy:run": "cypress run"
  }
}
```




Executando nossos testes sem UI

→ Alguns argumentos bastante utilizados:

- ◆ --browser ou -b
- ◆ --config-file ou -C
- ◆ --env ou -e
- ◆ --help
- ◆ --spec
- ◆ --tag



Outros comandos úteis

- **cypress info**: retorna informação sobre o Cypress e atual ambiente
- **cypress verify**: verifica se o cypress foi corretamente instalado e está executável
- **cypress version**: retorna versão do cypress

Aula 2 . Etapa 5

Faça você mesmo!

// Primeiros passos com Cypress

Colocando a mão na massa mais um pouco!

- Agora que aprendemos os conceitos principais para fazer uma automação, o que acha de colocar em prática e pensar mais um pouco?

Colocando a mão na massa mais um pouco!

1. Te desafio a complementar ou fazer do seu modo os test cases iniciados no curso. Faça no mínimo 6 test cases com base naqueles vistos no site do curso, separando as suítes de testes de maneira lógica.
2. Escolha 8 comandos (ver documentação [Table of Contents | Cypress Documentation](#)) e obrigatoriamente use em seus testes, não sendo necessário usar todos em um teste só. O objetivo é somente pensar em como usá-los!

Colocando a mão na massa mais um pouco!

3. Que tal arriscar alguns testes de API? Escolha no mínimo dois endpoints listados no site de demonstração do curso e faça validações conforme mostrado em aula.
4. Lembra do `cy.visit()` que utilizamos no início dos nossos testes? Que tal colocar ele como pré-condição? Te desafio a usar os hooks **before** ou **beforeEach** para evitar repetições de comandos utilizados por todos os testes.

Colocando a mão na massa mais um pouco!

5. Que tal compartilhar seu conhecimento no LinkedIn, na comunidade da DIO ou em outras redes? Faça uma postagem explicando alguma funcionalidade, comando ou conceito que tenha aprendido no percurso do curso que você tenha achado super interessante ou útil! **Sugestões pra vocês:**

- Explicando JQuery
- Diferença entre XPath - CSS Selector
- Explicando mais detalhadamente a diferença entre programação síncrona e assíncrona no contexto da automação
- Funcionalidades experimentais do Cypress
- Uso do Devtools na prática de testes manuais e automação

Colocando a mão na massa mais um pouco!

6. Por fim...suba todo código no seu repositório do Github, fazendo um README contando o que foi feito. Relembrando que o repositório da DIO é [digitalinnovationone/cursoPrimeirosPassosCypress \(github.com\)](https://github.com/digitalinnovationone/cursoPrimeirosPassosCypress), lá vocês podem fazer um fork do projeto ou simplesmente consultar.

Primeiros passos com Cypress

Carolina Santana Louzada

Analista QA - Venturus

Percurso

Aula 1

O que é Cypress?

Aula 2

Conceitos iniciais para automação E2E com Cypress

Dúvidas durante o curso?

- > Fórum do curso
- > LinkedIn: Carolina Santana Louzada
- > Comunidade online (Discord)



SCAN ME