

# Criando casos de testes utilizando BDD com Cypress

O framework utilizado para o BDD será o Cypress, por ser amplamente utilizado para este tipo de tarefa e pela facilidade de encontrar informações sobre.

A configuração do Cypress é relativamente simples e pode ser encontrada em vários sites na internet. Caso queira saber como é feita do zero, acesse o link:

<https://medium.com/cwi-software/testes-automatizados-com-cypress-e-cucumber-d78b211da766>

O foco deste tutorial não é a configuração do framework, já que o repositório que utilizaremos já terá a configuração correta, inclusive de versões, que podem causar “dores de cabeça” durante a configuração.

Para começarmos, clone ou baixe o repositório através do link abaixo:

<https://git.animaeducacao.com.br/henrique.costa/bddcypress.git>

Com o Visual Studio Code aberto na pasta do projeto, clique em “Terminal” > “New Terminal”

Em seguida digite, no terminal, o comando **npm install**. Este comando instalará as dependências necessárias no projeto.

Após o comando, a pasta `node_modules` estará na raiz do diretório, o que confirma a instalação das dependências.

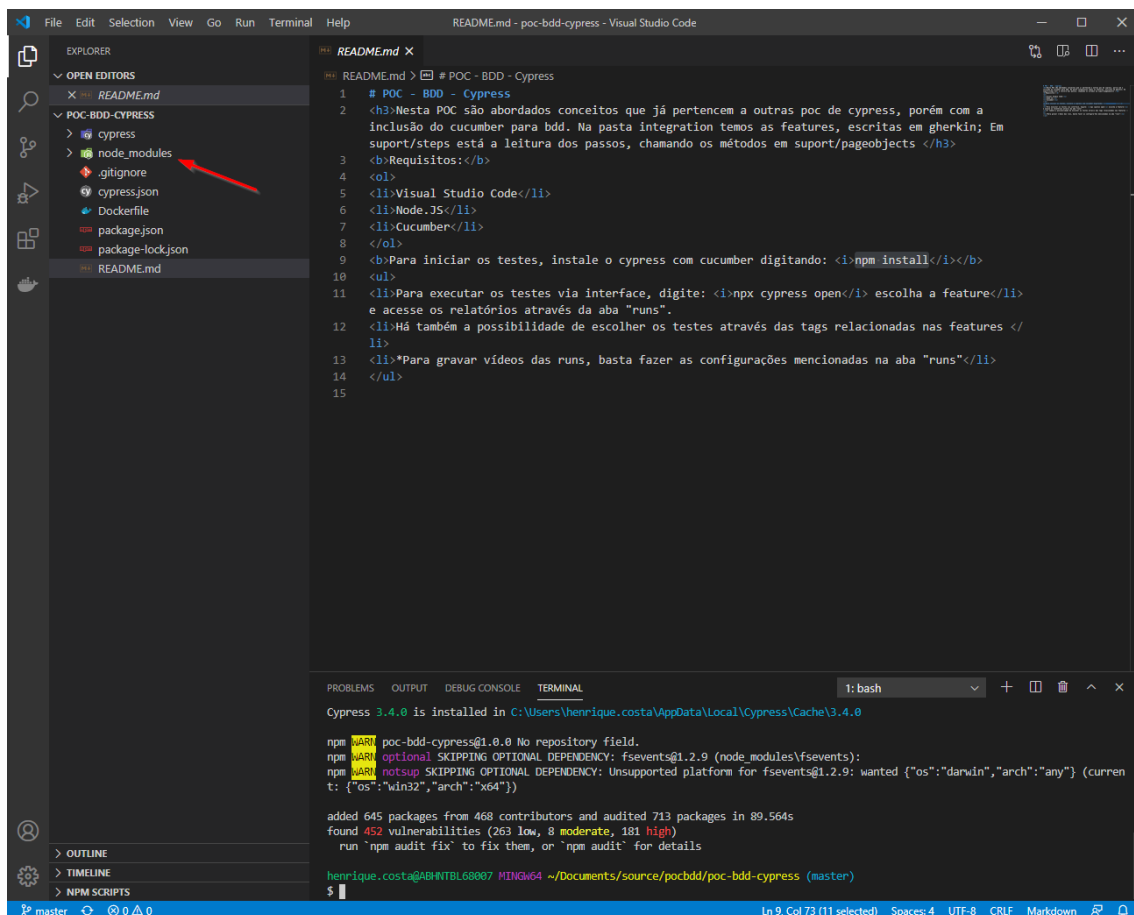


Figura 1

As pastas do projeto com BDD são, não por regra, mas por boa prática – ou sugestão minha:

/fixtures: Dados mockados que poderão ser acessados por todo o projeto.

/integration: Features, onde estarão os casos de teste escritos em Gherkin, que é uma linguagem de escrita utilizada em BDD.

/plugins: Configurações relacionadas à plugins utilizados no projeto.

/support: Classes e arquivos responsáveis pelo comportamento da aplicação de teste.

/support/elements: Mapeamento dos elementos da página, de forma que possamos ter acesso a eles simplesmente invocando os métodos de cada um.

/support/pageobjects: Classes de métodos responsáveis por interagir com os elementos das páginas.

/support/steps: Classes responsáveis pelo orquestramento e comunicação entre as features e os page objects.

/support/uteis: Classes e métodos que se repetirão em diversas partes do código. Isso auxilia na reutilização do código.

/node\_modules: Arquivos responsáveis pelo funcionamento do Cypress, cucumber e outros frameworks que podem ser utilizados.

As pastas que não estão criadas ainda devem ser criadas conforme a *figura 2*.

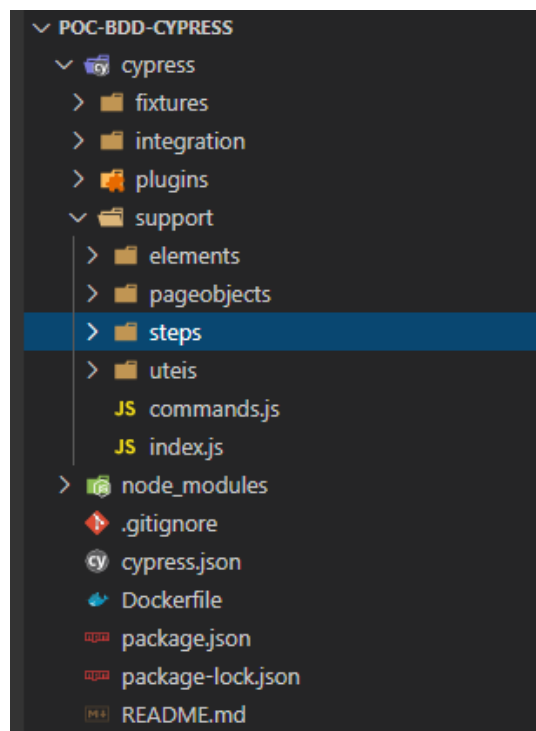


Figura 2

Como podemos observar na imagem acima, há também alguns arquivos fora das pastas, são eles:

commands.js: Arquivo de exemplo para criação, customização e reescrita de comandos.

index.js: Utilizado para configurações globais.

.gitignore: Arquivos que não subirão nos commits do git. Neste caso a pasta node\_modules, que é grande e já é criada ao rodar o comando npm install

cypress.json: Variáveis que serão utilizadas no projeto. Aqui podem ser incluídas configurações como resolução da tela do navegador, timeout de espera de elementos, url padrão, entre outros.

Dockerfile: Configurações que serão lidas pelo gerenciador de containers.

package.json: Tem diversas funções em cada projeto, um repositório central de configurações de ferramentas do Node. Ele também é onde npm armazena os nomes e versões dos pacotes instalados.

package-lock.json: É o detalhamento das versões dos pacotes utilizados no projeto. Isso evita que ao rodar o projeto em outra máquina, sem levar a node\_modules, o comando “npm install” instale versões diferentes dos pacotes do projeto original, o que ocasionaria erros.

README.md: A criação deste arquivo é uma boa prática, ele costuma conter informações que o proprietário do projeto ache relevante que sejam exibidas antes mesmo do download do projeto.

## Features

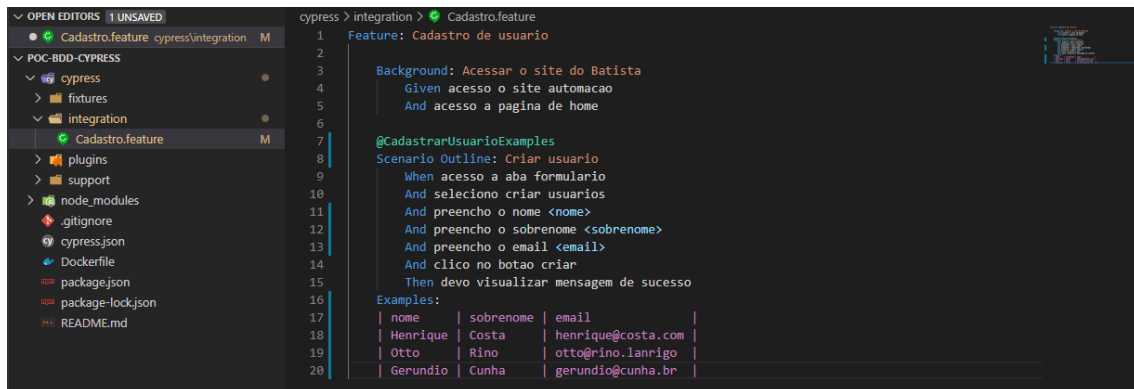
Agora que você já sabe o básico sobre funcionamento e estrutura do Cypress e vai olhar com uma carinha melhor para o projeto, pois conhece a função de cada parte, é hora de criar nossos testes!

Essa parte não tem uma ordem específica, explicarei na sequência que acho melhor para o entendimento.

Os nomes dos arquivos serão à sua escolha, lembre-se de manter um padrão para facilitar a vida de quem for trabalhar no seu código futuramente, que pode ser você, inclusive. Aqui citarei os nomes, como forma de exemplificar.

Crie, dentro da pasta /integration um arquivo com a extensão .feature

Ele deverá ser escrito em Gherkin como no exemplo:



```
1 Feature: Cadastro de usuario
2
3 Background: Acessar o site do Batista
4   Given acesso o site automacao
5   And acesso a pagina de home
6
7 @CadastrarUsuarioExamples
8 Scenario Outline: Criar usuario
9   When acesso a aba formulario
10  And seleciono criar usuarios
11  And preencho o nome <nome>
12  And preencho o sobrenome <sobrenome>
13  And preencho o email <email>
14  And clio no botao criar
15  Then devo visualizar mensagem de sucesso
16
17 Examples:
18 | nome | sobrenome | email |
19 | Henrique | Costa | henrique@costa.com |
20 | Otto | Rino | otto@rino.lanrigo |
21 | Gerundio | Cunha | gerundio@cunha.br |
```

Figura 3

Uma breve explicação sobre as palavras chave da linguagem:

Feature: Nome do caso de teste.

Background: O que o teste irá executar antes de todos cenários.

Given: A situação inicial.

Scenario Outline: O que o cenário testa. A palavra Outline indica que serão escritos exemplos dentro do Scenario. Cada linha representa um teste a ser realizado com os dados dela.

When: Uma ação.

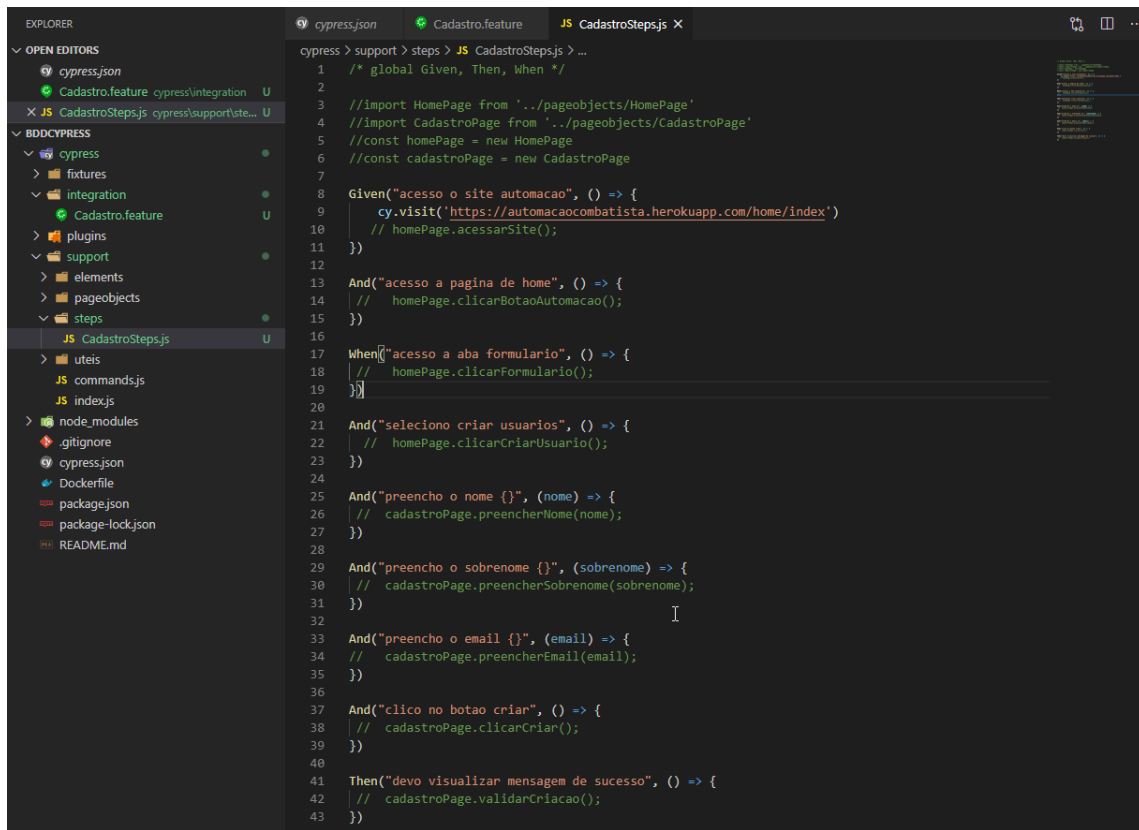
And: Ação subsequente.

Then: O resultado final das ações.

Examples: Dados que serão utilizados durante os testes. Eles devem ser feitos no padrão mostrado na *figura 3*.

## Criando os Steps

Na pasta /support/steps crie um arquivo com extensão .js fazendo referência ao nome da feature criada anteriormente, neste caso pode ser CadastroSteps.js conforme a *figura 4*.



```
1  /* global Given, Then, When */
2
3  //import HomePage from '../pageobjects/HomePage'
4  //import CadastroPage from '../pageobjects/CadastroPage'
5  //const homePage = new HomePage
6  //const cadastroPage = new CadastroPage
7
8  Given("acesso o site automacao", () => {
9    cy.visit('https://automacaocombatista.herokuapp.com/home/index')
10    // homePage.acessarSite();
11  })
12
13  And("acesso a pagina de home", () => {
14    // homePage.clicarBotaoAutomacao();
15  })
16
17  When("acesso a aba formulario", () => {
18    // homePage.clicarFormulario();
19  })
20
21  And("seleciono criar usuarios", () => {
22    // homePage.clicarCriarUsuario();
23  })
24
25  And("preencho o nome {}", (nome) => {
26    // cadastroPage.preencherNome(nome);
27  })
28
29  And("preencho o sobrenome {}", (sobrenome) => {
30    // cadastroPage.preencherSobrenome(sobrenome);
31  })
32
33  And("preencho o email {}", (email) => {
34    // cadastroPage.preencherEmail(email);
35  })
36
37  And("clico no botao criar", () => {
38    // cadastroPage.clicarCriar();
39  })
40
41  Then("devo visualizar mensagem de sucesso", () => {
42    // cadastroPage.validarCriacao();
43  })
```

Figura 4

Note que, neste arquivo, ações são atribuídas para cada passo do Cadastro.feature.

Este arquivo funciona como uma espécie de contrato com a feature correspondente, de forma que o teste só irá rodar se todos os passos forem implementados nele.

Uma boa prática é não escrever nenhum método, mas fazer chamadas para métodos que estarão dentro do arquivo que criaremos agora.

\* O método `homePage.acessarSite()` irá acessar a url contida no elemento "baseUrl" do arquivo `cypress.json`.

### Pausa para dica importante!



O Cypress possui uma ferramenta que auxilia no mapeamento de elementos. Uma dica legal é chamar apenas o método que faz o acesso à página que será testada e utilizar a ferramenta para mapear os elementos. Isso explica o fato dos imports e dos outros passos estarem com suas ações comentadas na *figura 4* (Não se esqueça de descomentar depois, jenzinho). Vamos dar uma olhadinha no funcionamento dessa ferramenta.

Crie o método PROVISÓRIO `cy.visit('https://automacaocombatista.herokuapp.com/home/index')` dentro do primeiro passo acessado, **Given**, neste caso. Ele deverá ser excluído após o mapeamento dos elementos.

Salve todas as alterações feitas até agora, sei que você já fez isso, pois ninguém gosta de codar um monte e correr o risco de perder tudo, né? Mas vale o lembrete.

Abra o terminal novamente e digite o comando **`npx cypress open`**

Uma tela como a da *figura 5* será exibida, clique na feature que deverá ser executada.

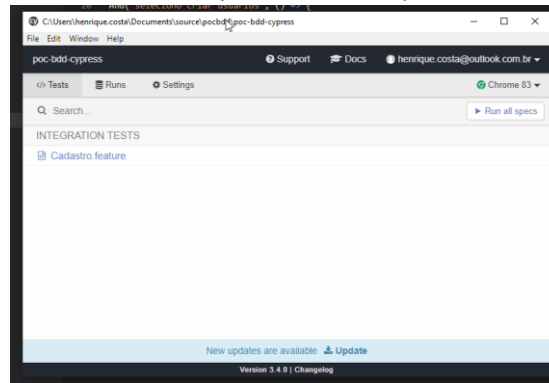



Figura 5

O navegador irá abrir a página escolhida no passo executado. Clique no botão . Ao mover o cursor sobre os elementos da página ele indicará as propriedades para mapeamento do elemento. Algumas vezes o elemento possui id, nome etc. que facilitam bastante na hora de mapear como na *figura 6*, outras ele pode ter nada disso como na *figura 7*, aí é que a ferramenta se torna ainda mais útil.

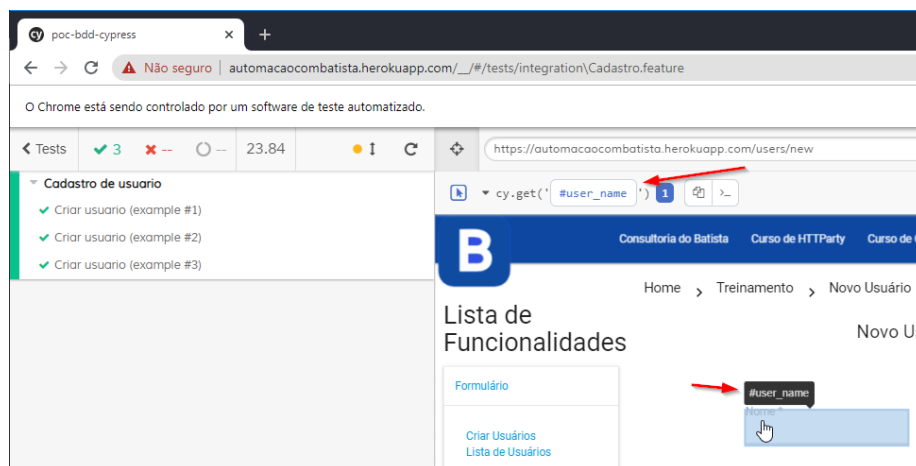


Figura 6

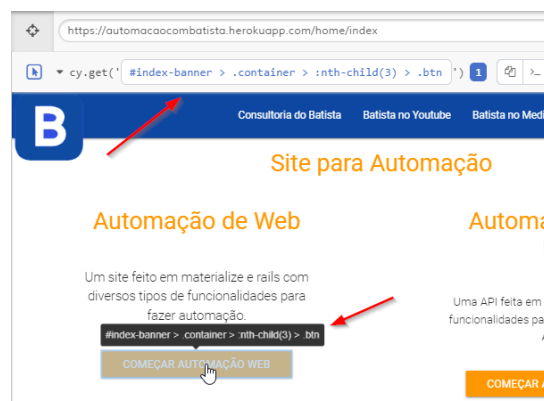


Figura 7

Ao clicar no botão copiar a instrução já estará copiada e pronta para ser utilizada no projeto.

Vale lembrar que este mapeamento pode ficar enorme e as vezes impreciso, por isso recomendo estudar sobre mapeamento de elementos utilizando o Css Selector. Para isso sugiro o site do link a seguir:

<https://medium.com/automa%C3%A7%C3%A3o-com-batista/aprenda-por-definitivo-a-usar-css-selector-adeus-xpath-1f3956763c2>

Voltemos ao tutorial...

## Page Objects

Os Page Objects são Classes responsáveis pela interação com os elementos da tela, o ideal é ter um para cada tela do sistema a ser testado. Neste caso teremos dois, o CadastroPage.js *figura 8* e HomePage.js *figura 9*.

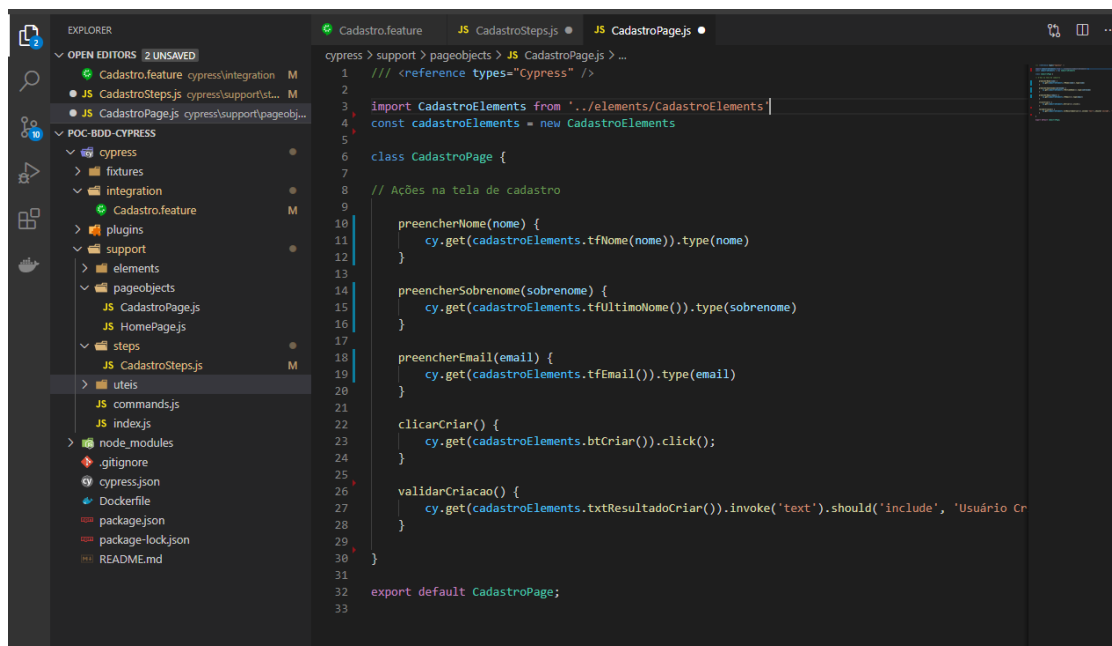


Figura 8

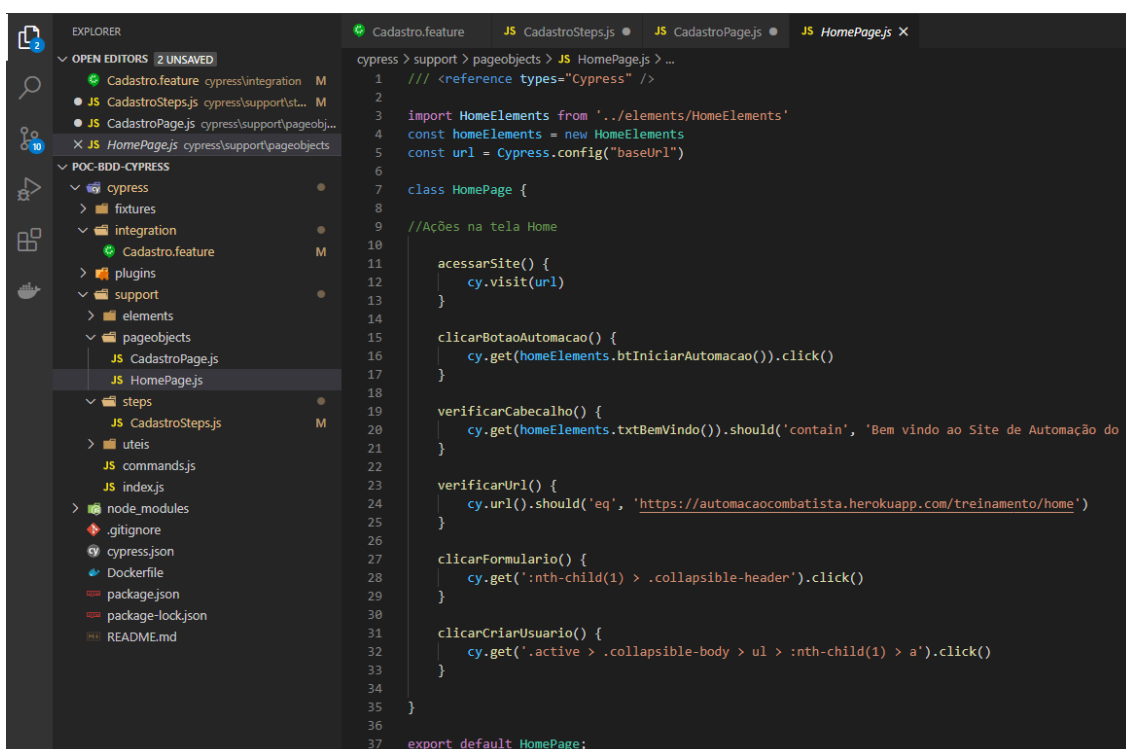


Figura 9

Estes arquivos terão as classes de métodos responsáveis pelas interações entre os CadastroSteps.js e a página. Esses métodos que são chamados dentro de cada Step.

Os arquivos dos page objects deverão importar propriedades dos arquivos da pasta /elements, assim, através das instâncias poderemos chamar os métodos que criaremos para retornar os elementos mapeados.

Aqui utilizaremos o cy.get. Que possui diversas aplicações, que podem ser conferidas mais detalhadamente dentro da própria documentação do Cypress. Link: <https://docs.cypress.io/api/commands/get.html>

A url base é obtida do arquivo cypress.json, explicado no início do tutorial.

## Elements

Aproveitando que você já ficou ninja nas técnicas de mapeamento de elementos mostradas anteriormente, agora basta criar métodos que retornarão os elementos mapeados.

Crie arquivos com extensões .js, usaremos os nomes CadastroElements.js e HomeElements.js.

Dentro deles crie uma classe homônima e escreva os métodos conforme as figuras 10 e 11.

```
cypress > support > elements > JS CadastroElements.js > CadastroElements > btCria
1 class CadastroElements {
2
3   // Leitura dos elementos da tela de cadastro
4
5   btCriar = () => { return 'input[name="commit"]' }
6   btVoltar = () => { return '.waves-light' }
7   textoBemVindo = () => { return '.orange-text' }
8   txtCriarUsuario = () => { return 'h5.center' }
9   txtResultadoCriar = () => { return '#notice' }
10  tfNome = () => { return '#user_name' }
11  tfUltimoNome = () => { return '#user_lastname' }
12  tfEmail = () => { return '#user_email' }
13
14
15 }
16
17 export default CadastroElements;
```

Figura 10

```
cypress > support > elements > JS HomeElements.js > HomeElements
1 class HomeElements {
2
3   // Leitura dos elementos da tela home
4
5   btIniciarAutomacao = () => { return 'a[href="/treinamento/home"]' }
6   txtBemVindo = () => { return '.orange-text' }
7 }
8
9 export default HomeElements;
10
```


Figura 11

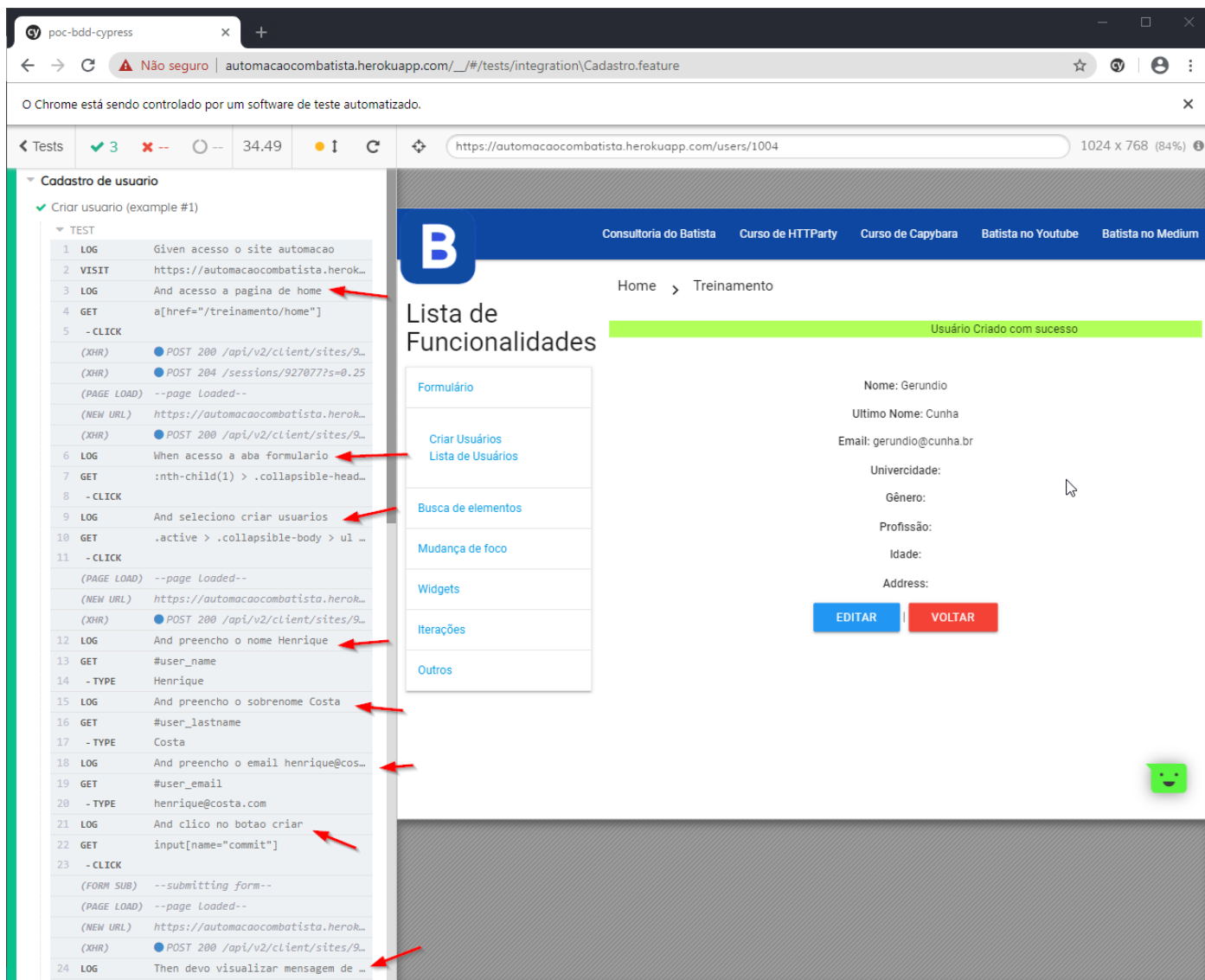
Para chamar os métodos basta instanciar a classe e chama-los através do objeto.

Exemplo: `cy.get(cadastroElements.btCriar()).click()`

Utilizações desse tipo poderão ser observadas nas figuras 8 e 9.

## Executando os testes

Agora é hora de dar aquele “salve”  em todas as alterações e abrir novamente o terminal. Digite o comando `npx cypress open` e clique na feature a ser executada e veja a “mágica” acontecer.



TEST	LOG
1	LOG Given acesso o site automacao
2	VISIT https://automacaocombatista.herokuapp.com/_/#/tests/integration/Cadastro.feature
3	LOG And acesso a pagina de home
4	GET a[href="/treinamento/home"]
5	- CLICK
	(XHR) POST 200 /api/v2/client/sites/9...
	(XHR) POST 204 /sessions/927077?s=0.25
	(PAGE LOAD) --page loaded--
	(NEW URL) https://automacaocombatista.herokuapp.com/_/#/tests/integration/Cadastro.feature
	(XHR) POST 200 /api/v2/client/sites/9...
6	LOG When acesso a aba formulario
7	GET :nth-child(1) > .collapsible-head...
8	- CLICK
9	LOG And seleciono criar usuarios
10	GET .active > .collapsible-body > ul
11	- CLICK
	(PAGE LOAD) --page loaded--
	(NEW URL) https://automacaocombatista.herokuapp.com/_/#/tests/integration/Cadastro.feature
	(XHR) POST 200 /api/v2/client/sites/9...
12	LOG And preencho o nome Henrique
13	GET #user_name
14	- TYPE Henrique
15	LOG And preencho o sobrenome Costa
16	GET #user_lastname
17	- TYPE Costa
18	LOG And preencho o email henrique@cos...
19	GET #user_email
20	- TYPE henrique@costa.com
21	LOG And clico no botao criar
22	GET input[name="commit"]
23	- CLICK
	(FORM SUB) --submitting form--
	(PAGE LOAD) --page loaded--
	(NEW URL) https://automacaocombatista.herokuapp.com/_/#/tests/integration/Cadastro.feature
	(XHR) POST 200 /api/v2/client/sites/9...
24	LOG Then devo visualizar mensagem de ...

Figura 12

Note que os passos descritos na feature são exibidos no log.

Agora é utilizar esses conhecimentos para criar seus próprios testes e explorar outras funcionalidades da aplicação. Lembrando que a documentação é muito boa e está disponível no site oficial do Cypress.

<https://docs.cypress.io/guides/overview/why-cypress.html#In-a-nutshell>



## Referências

- <https://www.cypress.io/>
- <https://github.com/TheBrainFamily/cypress-cucumber-preprocessor>
- <https://medium.com/cwi-software/testes-automatizados-com-cypress-e-cucumber-d78b211da766>
- <https://medium.com/automa%C3%A7%C3%A3o-com-batista/aprenda-por-definitivo-a-usar-css-selector-adeus-xpath-1f3956763c2>
- [https://www.luiztools.com.br/post/o-guia-completo-do-package-json-do-node-js/?gclid=Cj0KCQjwoaz3BRDnARIsAF1RfLfy9QPKtD1AJNg8mosxzNLOBDwcnKv-iFdF0u82yhrCKptg6lMu\\_2UaAh68EALw\\_wcB](https://www.luiztools.com.br/post/o-guia-completo-do-package-json-do-node-js/?gclid=Cj0KCQjwoaz3BRDnARIsAF1RfLfy9QPKtD1AJNg8mosxzNLOBDwcnKv-iFdF0u82yhrCKptg6lMu_2UaAh68EALw_wcB)