



Cucumber e Cypress

Vamos para o casamento dos sonhos: uma escrita voltado para comportamento junto com a automação E2e. Mas primeiramente, vamos fazer um breve resumo do que é o Cypress e Cucumber

O que é o Cypress?

Cypress é um framework utilizado para automação de testes end to end. Atualmente, utiliza JavaScript como linguagem de programação e executa os testes nos navegadores (chrome, mozilla, electron e ect).

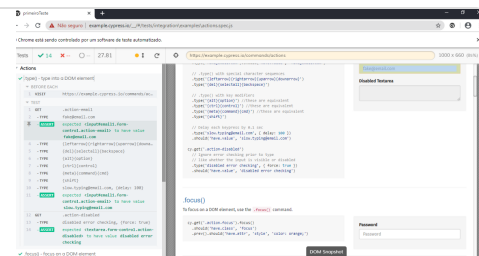
Suas principais vantagens são de possuir uma curva de aprendizado pequena e ser de fácil instalação e configuração. Em contrapartida, possui limitações por utilizar somente o browser e não trocar de janela durante os testes.

Já temos um DOJO onde iniciamos um projeto do zero, realizamos algumas configurações e teste básico (aqui o link do DOJO). Mas aqui segue um link prático de como realizar a instalação e configuração básica do Cypress:

Instalando o Cypress sem mistérios

O Cypress é um framework para automação de testes e2e para aplicações web. Com ele você vai escrever seus testes usando a linguagem javascript. É uma ferramenta que vem crescendo e

<https://medium.com/gruponewway/instalando-o-cypress-sem-mist%C3%A9rios-6d6ee66b78d8>



O que é Cumcumber?

Cucumber é uma ferramenta que pode ser utilizada em conjunto com o Cypress e permite a escrita de testes automatizados na sintaxe Gherkin, mais conhecido como formato BDD (Behaviour-Driven Development). É possível utilizar o Cypress sem cucumber, mas não é o foco deste tutorial.

Desmistificando o uso do Gherkin

Quem nunca pegou um cenário escrito em e teve a ligeira sensação de estar lendo um caso de teste tradicional, cheio de passo a passo e pré-condições? Pois bem, nesse artigo irei <https://medium.com/revista-dtar/desmistificando-o-uso-do-gherkin-d1e56c592b80>



E aí, vamos a prática?

Tendo como parâmetro que já temos o cypress intalado e rodando, vamos as passos para configurando o cumcumber:

Obs.: Todos os passos aqui serão aplicados para versões do cypress a partir da 10.0.1

- Tendo acessado a pasta do projeto, execute o comando de instalação:

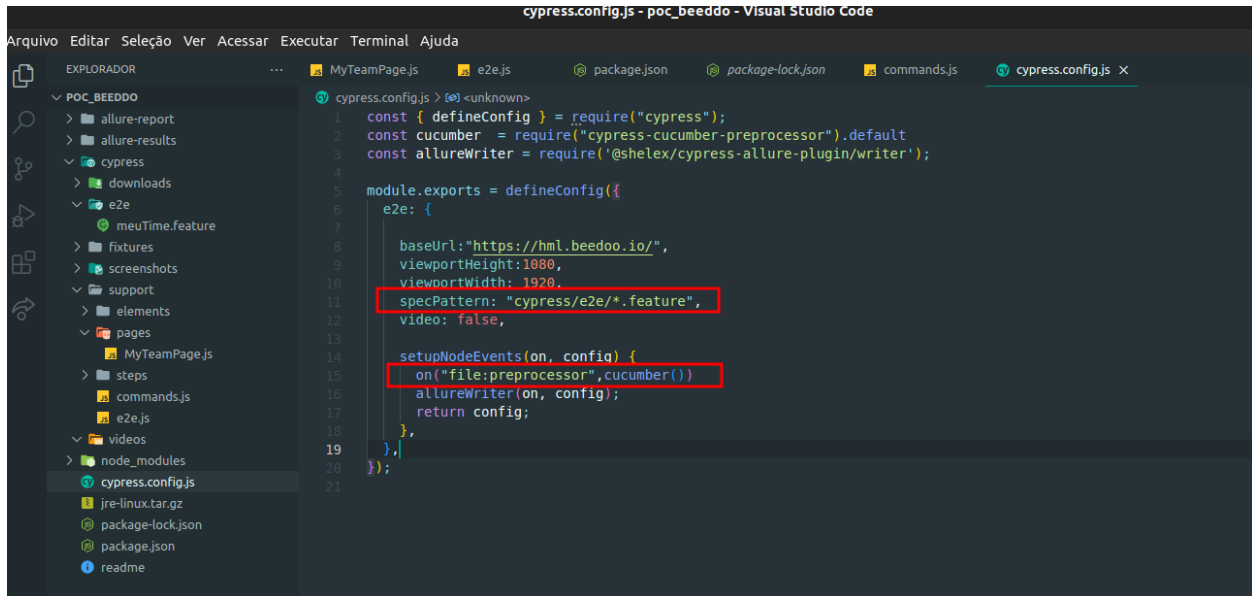
```
npm install --save-dev cypress-cucumber-preprocessor
```

- No arquivo `cypress.config.js` que está na raiz do projeto, vamos importar a dependencia e defini-la a uma variavel (linha 3 do codigo).

Adicione dentro de e2e a linha `specPattern: "cypress/e2e/*.feature"` (estamos dizendo aqui onde estarão os arquivos .feature que serão escritos os cenários em Guerking).

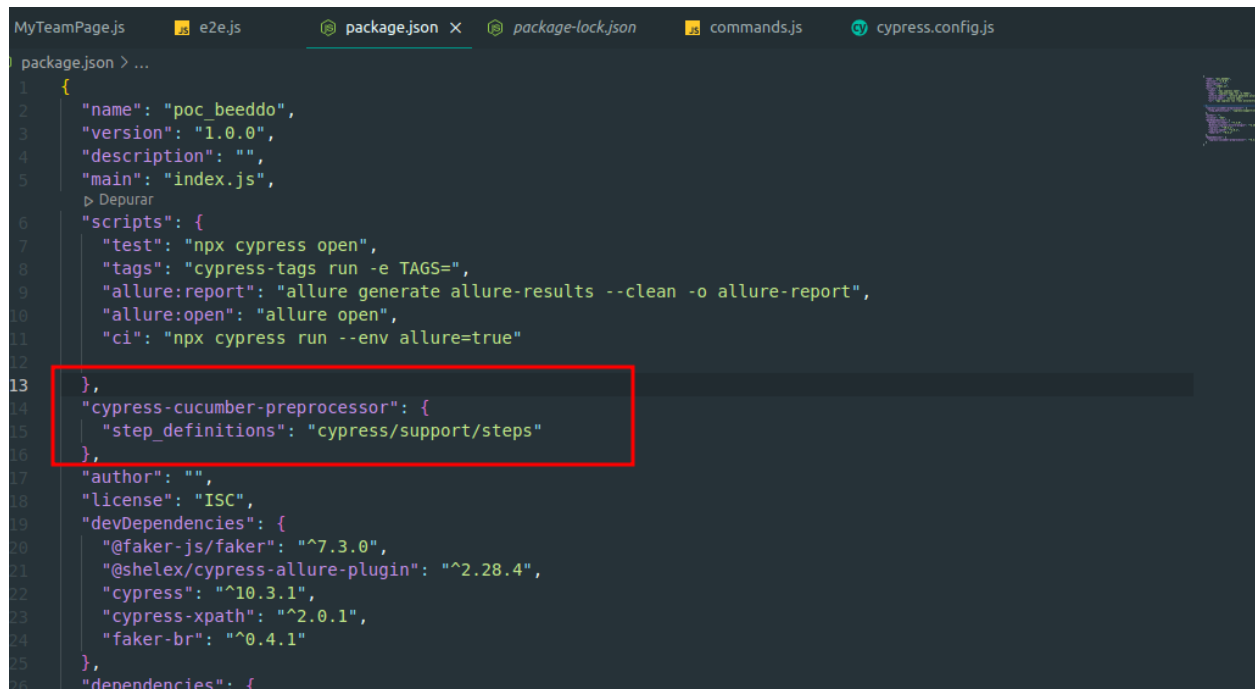
Além disso, dentro da função `setupNodeEvents` vamos “ligar” o cucumber como pré-processador com `on("file:preprocessor", cucumber())`.

Segue imagem abaixo exemplificando como realizar essa configuração:



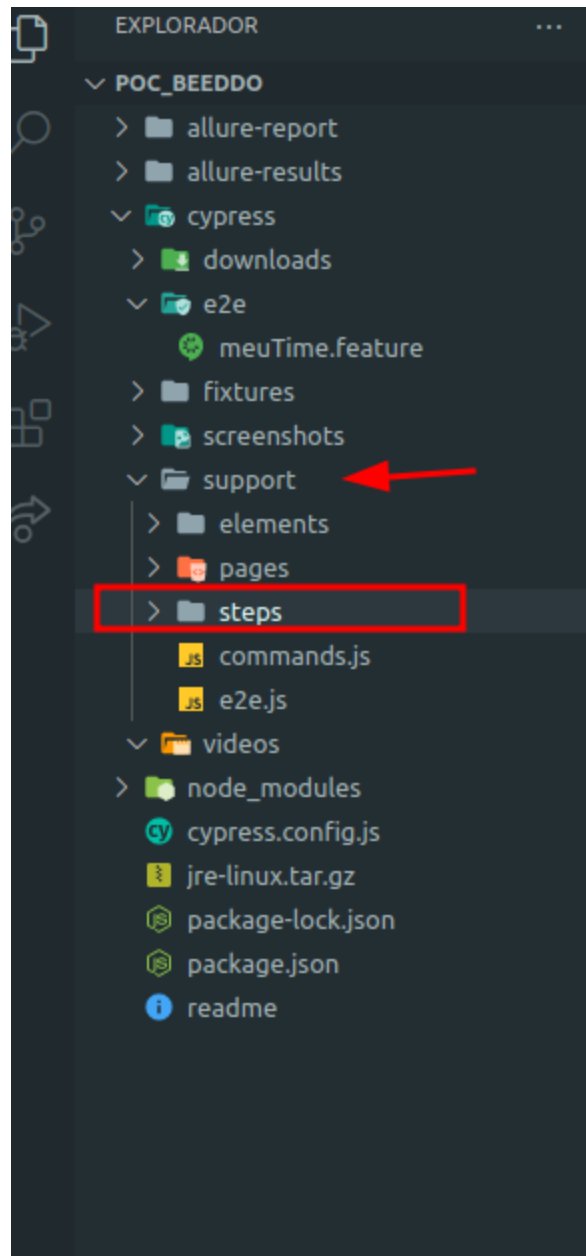
```
1 const { defineConfig } = require("cypress");
2 const cucumber = require("cypress-cucumber-preprocessor").default
3 const allureWriter = require('@shelex/cypress-allure-plugin/writer');
4
5 module.exports = defineConfig({
6   e2e: {
7     baseUrl: "https://hml.beedoo.io/",
8     viewportHeight: 1080,
9     viewportWidth: 1920,
10    specPattern: "cypress/e2e/*.feature",
11    video: false,
12
13    setupNodeEvents(on, config) {
14      on("file:preprocessor", cucumber())
15      allureWriter(on, config);
16      return config;
17    },
18  },
19 });
```

- No arquivo `package.json` vamos adiciona a linha `"step_definitions": "cypress/support/steps"` para apontarmos aonde estão nossos steps (vamos entender melhor o que é isso mais adiante...):



```
1 {
2   "name": "poc_beeddo",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "npx cypress open",
8     "tags": "cypress-tags run -e TAGS=",
9     "allure:report": "allure generate allure-results --clean -o allure-report",
10    "allure:open": "allure open",
11    "ci": "npx cypress run --env allure=true"
12  },
13  "cypress-cucumber-preprocessor": {
14    "step_definitions": "cypress/support/steps"
15  },
16  "author": "",
17  "license": "ISC",
18  "devDependencies": {
19    "@faker-js/faker": "^7.3.0",
20    "@shelex/cypress-allure-plugin": "^2.28.4",
21    "cypress": "^10.3.1",
22    "cypress-xpath": "^2.0.1",
23    "faker-br": "^0.4.1"
24  },
25  "dependencies": {
```

- Agora, precisamos criar uma pasta chamada “steps” dentro da pasta “support”, para adicionar os nossos arquivos de passos que serão integrados ao nossos cenários (caminho esse que configuramos no passo anterior):



Agora, como o nosso projeto configurado, vamos começar a escrever nossos cenários e colocar a mão na massa 😊

Criando os arquivos .feature

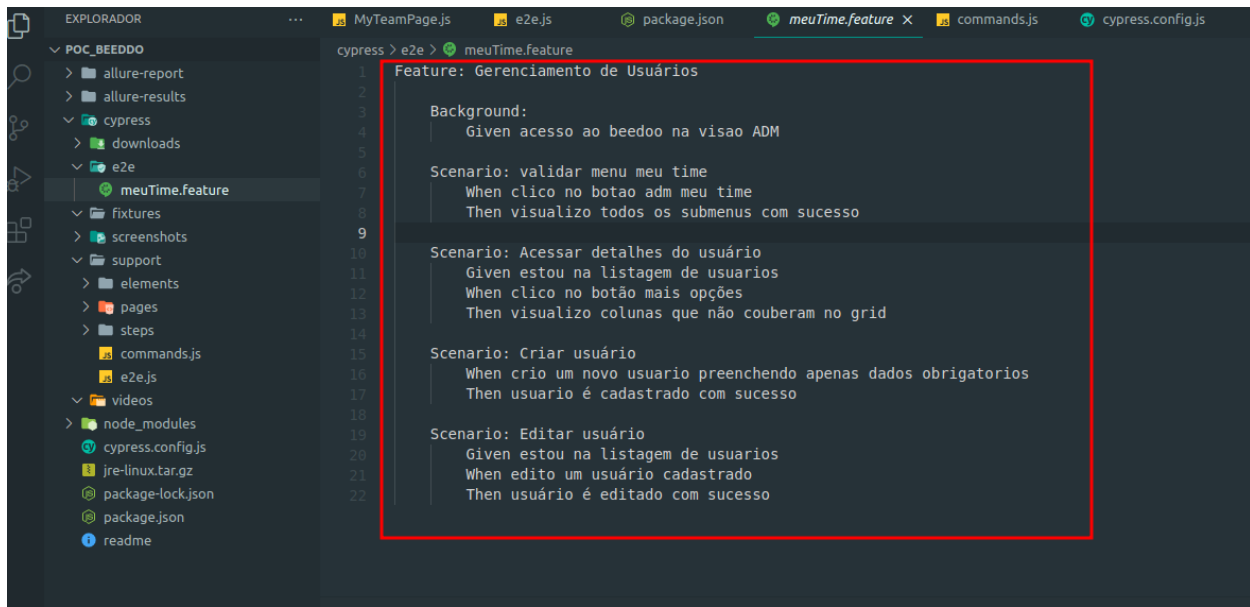
Dentro da pasta 2e2, vamos criar nosso arquivo `.feature` onde iremos escrever nossos cenários em guerkin. No nosso exemplo, vamos criar uma suite e alguns cenários relacionado ao menu **MEU TIME**, por isso, nosso arquivo vai se chamar

`meuTime.feature`. Esse será o nome da nossa suite de testes. Vamos entender a estrutura do arquivo com algumas **keywords** :

- **Given** (pt: **Dado**): Utilizado para especificar uma pré condição, dentro desse step é feita a validação de uma condição antes de se prosseguir para os próximos passos. Por se tratar de uma pré condição, normalmente vem escrito no passado;
- **When** (pt: **Quando**): Utilizado quando será executada uma ação de que se espera uma reação vinda do sistema, que será validada no step “Then”. Este passo vem escrito no presente;
- **Then** (pt: **Então**): Valida se o esperado aconteceu. Segue sempre um passo do tipo “Quando”, pois aqui é validada a reação da ação recebida. Por se tratar do resultado esperado, normalmente vem escrito na forma de futuro próximo;
- **And** (pt: **E**): Caso seja necessário mais uma interação com o sistema para complementar um fluxo, mas que não necessariamente se trata de uma ação ou reação, se utiliza “And”;
- **But** (pt: **Mas**): No geral serve a mesma funcionalidade do “And”, porém é normalmente utilizado após uma validação negativa depois do “Then”;

Temos ainda algumas **keywords** para organizar e facilitar alguns fluxos, sendo elas:

- **Feature** (pt: **Funcionalidade**): Serve para nomear a funcionalidade a ser testada nesta feature;
- **Background** (pt: **Contexto**): Quando existem mais cenários em uma feature que possuem a mesma pré condição, essa pré condição pode ser definida dentro de um único contexto para ser executada sempre antes da execução de qualquer cenário da feature, reaproveitando o mesmo passo em vários cenários diferentes;
- **Scenario** (pt: **Cenário**): Também chamado de exemplo. Descreve o comportamento esperado (Then) dado as pré condições (Given) e ação (When) especificada



Seguindo com o nosso exemplos, perceba que alguns cenários não temos o GIVEN (DADO) devido a o contexto (Background) que foi adicionado antes, e será executado antes de todos os testes.

Nos teste que realmente precisam de uma pré-condição específica, podemos utilizar o GIVEN sem problemas, as duas pré-condições serão executadas.

Com o nosso arquivo .feature criado, vamos agora criar de fato nossa automação com o cypress.

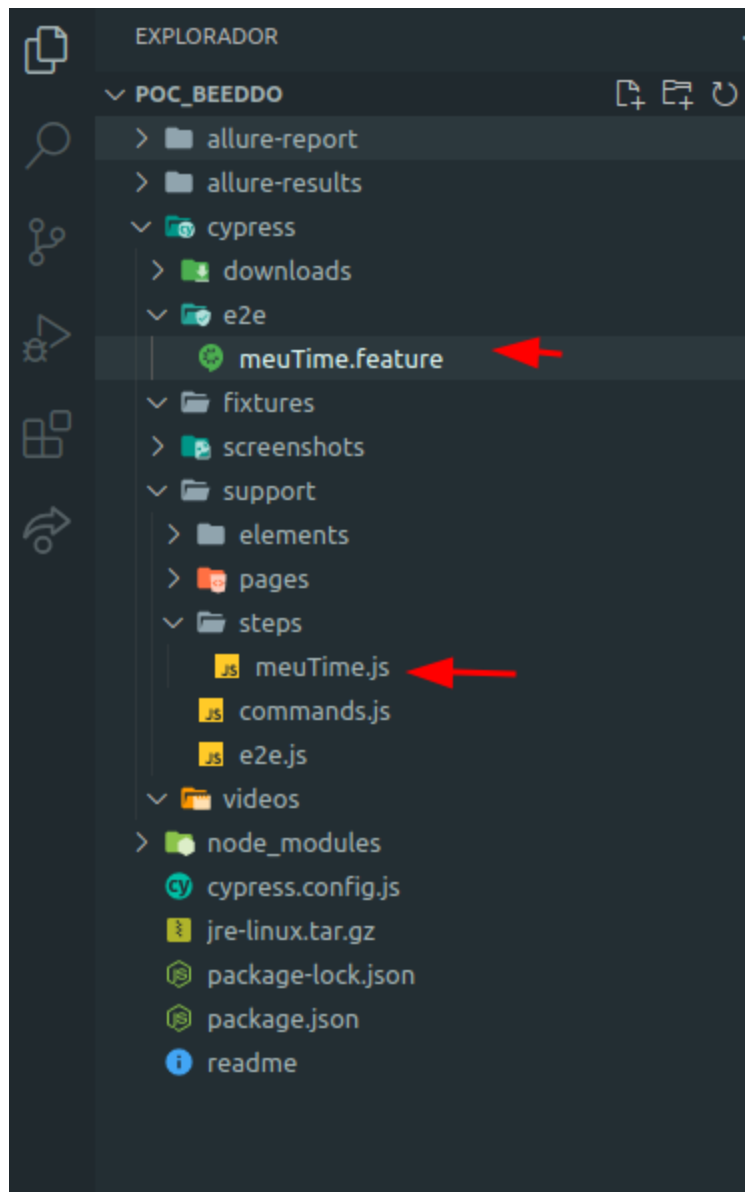
Criando os STEPS

Agora, vamos dar continuidade a nossa automação, criando o arquivo de steps da nossa automação.

Antes, uma dica. Existe uma extensão do visual code muito legal que se chama “Cuke Step Definitio Generator”, que vai nos ajudar muito na produtividade e criação das steps.

Com isso, precisamos criar dentro de pasta support/steps. um arquivo com extensão .js.

É uma boa prática para nossa organização do projeto, que o arquivo tenha o mesmo nome do arquivo .feature. Nesse caso, vamos chamar o mesmo de meuTime.js.



Agora, vamos seguir os seguintes passos:

- No arquivo .feature, vamos selecionar todos os cenários e o background. Após isso, clicar com o botão direito do mouse e escolher a opção: `Generate Step Definition: copy to clipboard.`

Essa opção estará disponível graças a extensão que instalamos. Todas as steps serão criadas “automaticamente” baseado nos nossos cenários.

- No arquivo meuTime.js que criamos em support/steps, vamos apenas executar o comando CTRL + V. Perceba que “automaticamente” todas os passos serão criados:

```
3 Given(/^acesso ao beedoo na visao ADM$/, () => {
4   |   return true;
5   | });
6
7 When(/^clico no botao adm meu time$/, () => {
8   |   return true;
9   | });
10
11 Then(/^visualizo todos os submenus com sucesso$/, () => {
12   |   return true;
13   | });
14
15 Given(/^estou na listagem de usuarios$/, () => {
16   |   return true;
17   | });
18
19 When(/^clico no botão mais opções$/, () => {
20   |   return true;
21   | });
22
23 Then(/^visualizo colunas que não couberam no grid$/, () => {
24   |   return true;
25   | });
26
27 When(/^crio um novo usuario preenchendo apenas dados obrigatorios$/, () => {
28   |   return true;
29   | });
30
31 Then(/^usuario é cadastrado com sucesso$/, () => {
32   |   return true;
33   | });
34
35 Given(/^estou na listagem de usuarios$/, () => {
36   |   return true;
37   | });
38
39 When(/^edito um usuário cadastrado$/, () => {
40   |   return true;
41   | });
```

Pronto, nosso projeto já está estruturado. A partir daí, começamos a executar comandos cypress dentro de cada passo:

```
Given(/^acesso ao beedoo na visao ADM$/, () => {
  cy.visit("/")


  //intercepta o ultimo endpoint que carrega elementos da home, para não deixar o teste "flaky"
  cy.intercept({
    url: 'https://va.tawk.to/log-performance/v3',
    method: 'POST'
  }).as('getFeed')

  cy.get("#inputLogin").type("djalma")
  cy.get("#inputPassword").type("djalma112192")
  cy.get('button[type="submit"]').click({ force: true })
  cy.wait('@getFeed')
});
```

Outro ponto muito importante é, que a partir daqui, podemos também aplicar o conceito de *Page Objects*, que deixa o projeto com alto nível de reutilização de código, escrita clara e fácil manutenção. Mais informações sobre esse assunto no link abaixo:

Cypress + Page Object = Sucesso

Bom, antes de começarmos a falar aqui sobre como implementar o padrão PageObject no nosso projeto Cypress, precisamos saber se vocês realmente sabem o que é PageObject. "Eu já ouvi falar

 <https://vitormarinheiroautomation.medium.com/cypress-page-object-sucesso-6841cb7c19a0>



E é só

Até aqui pessoal, fizemos a base do nosso projeto, o que já nos dá um grande start para continuarmos evoluindo nos estudos e na criação de um bom e organizado projeto de automação.

Por hoje é só, obrigado pelo seu tempo!