**Cypress: automação de testes E2E**

1. **Introdução ao Cypres**

* Iniciar um novo projeto usando o NodeJS, através do comando npm init, inicializando o arquivo package.json;
* Criar um novo arquivo de teste, com extensão .cy.js dentro da pasta integration e utilizar o comando describe para descrever a suite de testes;
* Usar a função beforeEach e o comando cy.visit(), para indicar a URL que será carregada antes de cada novo caso de teste;
* Criar um novo caso de teste utilizando a função it e utilizar os recursos cy.get(), bem como as opção .type, .click e .should para fazer os assertions;
* Usar o Selector Playground do Cypress para identificar o nome do elemento;
* Usar as ferramentas de desenvolvedor para melhor identificar os elementos html da página.

1. **Interação com elementos HTML**

* Criar uma nova suite de testes, criando um novo arquivo com a extensão .spec.js dentro da pasta integration;
* Utilizar com mais detalhes as ferramentas de desenvolvedor (DevTools) do navegador para explorar e identificar os elementos com que o Cypress irá interagir;
* Utilizar o comando cy.contains() para procurar por um texto dentro de um seletor, facilitando a busca por um elemento;
* Criar novos casos de teste, incluindo a função it(). Não há limites para a quantidade de testes incluídos.

1. **Mais recursos do Cypress**

* Executar os testes em modo headless, sem abrir a interface gráfica do Cypress;
* Verificar os relatórios e vídeos gerados ao executar testes em modo headless;
* Usar o pacote Mochawesome para personalizar a geração do relatório e ter a possibilidade de gerá-los como arquivos html, visualizando-os no browser;
* Utilizar o Dashboard para integrar os resultados dos testes e ter uma visão mais analítica e completa dos dados;
* Criar novos comandos personalizados, de acordo com a necessidade e referenciá-los como se fossem comando nativos do Cypress, como por exemplo: cy.login(), cy.registrar() ou da forma que considerar melhor.

1. **Mais possibilidades de teste**

* Utilizar dados de um arquivo externo nos casos de teste, fornecendo uma massa de dados;
* Utilizar a função require para obter dados de um módulo ou arquivo externo;
* Utilizar a função forEach para iterar sobre uma coleção de elementos e não precisar replicar código;
* Fazer testes em API’s REST, verificando o status e corpo da resposta, bem como outros comportamentos;
* Usar o comando cy.request() e usar o .expect para fazer as asserções dos resultados esperados;
* Cuidados ao testar dados sensíveis, para não expor o conteúdo a quem não poderia ter acesso;
* Criar arquivo cypress.env.json para configurar variáveis de ambiente;
* Usar o comando {logs: false} na função que faz a digitação para omitir os dados que não podem ser mostrados.

1. **Padrões de boas praticas**

* Organizar os testes separando-os por funcionalidades afins, para melhorar a manutenção e legibilidade do código;
* Qual o conceito dos flaky tests e algumas maneira de evitá-los, como por exemplo:evitar eventos orientados a tempo, dependência de aplicações de terceiros, testes que podem ser quebrados por processamento concorrente;
* Boas práticas na escrita dos testes para melhorar performance e segurança;
* O conceito de mocks e stubs e a importância de usá-los;
* A importância em já pensar em boas práticas de escrita, organização e performance desde a concepção do projeto de testes.