Testes Automatizados com React

Labenu_

O que vamos ver hoje?

- React Testing Library
- Usando o RTL
- Aprofundando no RTL

React Testing Library

Labenu_

react-testing-library

- Série de funções que facilitam o teste de componentes React
- Enforça **boas práticas**, pois suas funções incentivam o **teste de funcionalidades**, e não de implementação

react-testing-library

- Indicado pela documentação oficial do React
- Já vem instalado no Create React App (mas não no CodeSandbox)

Usando o RTL

Labenu_

Passo-a-passo

Para testar uma aplicação frontend com o React Testing Library, vamos seguir um passo-a-passo.

- Renderizar um componente
- Encontrar um elemento
- Disparar um evento
- Fazer verificações

Sequência de teste

1. Renderize o componente
   - Isso é feito usando o método **render**
   -
```
const { getByText } = render(<App />);
```

2. Se depender de uma ação do usuário, encontre o elemento alvo da interação
   - Tipo um botão ou input

```
const addButton = getByText(/Somar/i);
```

Sequência de teste

3. Simule a ação do usuário
    - Tipo clicar no botão ou digitar no input
    - Isso é feito usando o **userEvent**

```
userEvent.click(addButton);
```

4. Verifique se o que deveria ter mudado de fato mudou
    - Usar os matchers do Jest para saber se a ação mudou a tela corretamente.

```
expect(getByText(/Contador/i)).toHaveTextContent("Contador: 1");
```

Usando o react-testing-library

```
import React from "react";
import { render } from "@testing-library/react";
import userEvent from "@testing-library/user-event";
import App from "./App";

test("Contador mostra 1 quando somar é clicado uma vez", () => {
  const { getByText } = render(<App />);

  const addButton = getByText(/Somar/i);

  userEvent.click(addButton);

  expect(getByText(/Contador/i)).toHaveTextContent("Contador: 1");
});
```

Aprofundando no RTL

Labenu_

Encontrando um elemento

O método *render* retorna um objeto com algumas funções de busca.

- São três tipos:
  - **getBy**: retorna um elemento encontrado. Se não encontra, dá erro.
  - **queryBy**: retorna um elemento encontrado ou *null*
  - **findBy**: retorna uma promise, aguardando 1s para ver se encontra o elemento. Se não encontra, dá erro.
  - **Todos esses** retornam erro caso **mais de um elemento seja encontrado**
- Para encontrar mais de um elemento (uma array), combinar as funções com a palavra *all*:
  - getAllBy, queryAllBy, findAllBy

Type of Query	0 Matches	1 Match	>1 Matches	Retry (Async/Await)
Single Element				
`getBy...`	Throw error	Return element	Throw error	No
`queryBy...`	Return `null`	Return element	Throw error	No
`findBy...`	Throw error	Return element	Throw error	Yes
Multiple Elements				
`getAllBy...`	Throw error	Return array	Return array	No
`queryAllBy...`	Return `[]`	Return array	Return array	No
`findAllBy...`	Throw error	Return array	Return array	Yes

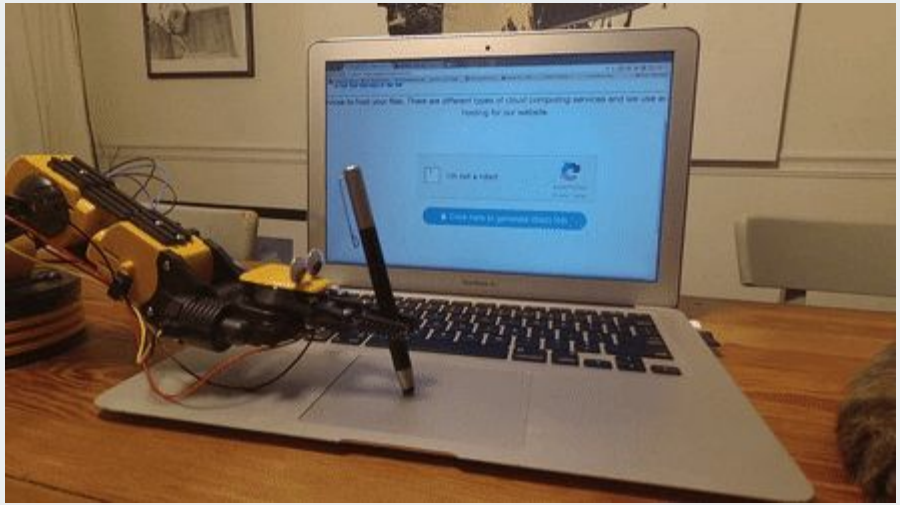
Encontrando um elemento

- Os métodos de busca devem ser combinados também com um indicativo de como a busca será feita.

    - **getByText**: encontra um elemento pelo texto que ele possui. Aceita uma string ou uma regex
    - **getByLabelText** e **getByPlaceholderText**: encontra inputs associados a uma label ou a um placeholder
    - **getByTestId**: é possível colocar um atributo *data-testid* nos elementos, e usar esse método para buscar por esse id de teste

Disparando eventos

- Para disparar um evento, devemos usar o objeto userEvent
- Ele possui métodos para todos os eventos mais comuns que podem ser disparados, simulando um usuário



click

- Passe o botão como argumento para userEvent.click()

```
1  import {render} from '@testing-library/react'
2  import userEvent from '@testing-library/user-event'
3  import App from './App'
4
5  test('Teste click', () => {
6    const {getByText} = render(<App/>)
7
8    const button = getByText(/Botao/)
9
10   userEvent.click(button)
11 }
```

type (digitação)

- Sintaxe é *userEvent.type(elementoInput, 'texto para digitar')*
- Retorna uma **promise** (só colocar um *await*)

```
1  import {render} from '@testing-library/react'
2  import userEvent from '@testing-library/user-event'
3  import App from './App'
4
5  test('Teste click', async () => {
6    const {getByLabelText} = render(<App/>)
7
8    const input = getByLabelText(/Input/)
9
10   await userEvent.type(input, 'Teste digitação')
11 }
```

selectOptions

- Recebe o elemento do select e o elemento da opção a ser selecionada

```
1  import {render} from '@testing-library/react'
2  import userEvent from '@testing-library/user-event'
3  import App from './App'
4
5  test('Teste click', async () => {
6    const {getByLabelText, getByText} = render(<App/>)
7
8    const select = getByLabelText(/Select/)
9
10   userEvent.selectOptions(select, getByText(/Opção/))
11 }
```

Pausa para relaxar 😴

10 min

Exercício 1

- Vamos testar o LabEcommerce:
  - "Quantidade de produtos" exibe número correto
  - Filtro de mínimo
  - Filtro de nome
  - Ordenação decrescente

Resumo

Labenu_

Resumo 📓

- A **react-testing-library** facilita o teste de componentes React;

- As três principais funções de busca são:
  - **getBy**: retorna um elemento encontrado.
  - **queryBy**: retorna um elemento encontrado ou null
  - **findBy**: retorna uma promise, aguardando 1s para ver se encontra o elemento.

- Eles podem ser combinados com a palavra all, para encontrar um array de elementos: **getAllBy**, **queryAllBy**, **findAllBy**

Resumo 📓

- As principais funções para encontrar um elemento:
  - **getByText**: busca pelo texto que ele possui.
  - **getByLabelText** e **getByPlaceholderText**: encontra inputs associados a uma label ou a um placeholder
  - **getByTestId**: busca por id do teste

- O objeto **user-events** dispara os eventos mais comuns
  - **Click**
  - **Digitação em inputs**
  - **Select**

Dúvidas? 🧐

Labenu_

Obrigado!