TRYBE
Modulo II – Front-end

Bloco 15 – React Testing Library

1) Testando React

Opções: enzyme ou RTL .

Casos de uso
Com RTL, a idéia é testar mais casos de uso do que funcionalidades.
Code Coverage é automático e apenas mostra a % de code total que é testado, enquanto o *Code Use Coverage* devemos construir nós mesmo.

"Rather than thinking about the code, think about the observable ***effect that code has for the end user and developer user****"*: é comum pensar primeiro em ciclos, eventos, estados em vez de user interaction, changing props, context changes, subscription changes.

E2E End to End test para garantir o funcionamento do caminho ideal do usuário no app.

Introdução ao RTL

Para testar, devemos **simular o comportamento** de quem usa.
Aqui cheatsheet RTL.

A/ Renderização

→ Primeiro *importar:*
import {} from '@testing-library/react';

→ *Armazenar* o rendimento de um componente numa const
→ Associar *expect* + *querie* pertinente + *matcher* (como no Jest)

```
import React from 'react';
import { render } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  const meuApp = render(<App />);
  const linkElement = meuApp.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

//query é aqui getByText, matcher toBeInTheDocument.

→ Verificar com *npm test.*
Para ter detalhes, pode ser *npm test –verbose* .
Dica para evitar falso positivo no teste: adicionar um *.not* no matcher para ver falhar.

B/ Queries (seletores)

Definição
Forma de **buscar o elemento** dentro do componente renderizado pelo render.

Sintax
Lista de queries RTL e guia sobre como escolher/priorizar seletor.

Examplos

```
test('Verificando se existe um botão', () => {
  const { getByRole } = render(<App />);
  const btn = getByRole('button');
  expect(btn).toBeInTheDocument();
});
```

getByText;
(pode conter *'texto'* ou */texto/i* para ignorar upper ou lower case)
getByLabelText para por exemplo input;

getByRole, espera apenas um elemento;
getAllByRole quando tiver mais de um, armazena todos valores encontrados num array

```
test('Verificando se existe dois botões', () => {
  const { getAllByRole } = render(<App />);
  const buttons = getAllByRole('button');
  expect(buttons.length).toBe(2);
});
```

getByTestId, associado com propriedade data-testid no elemento html testado

C/ Eventos

Definição
Forma de testar o comportamento do usuário, simulando um evento na página.

```
test('Verificando se o botão e o campo email estão funcionando.', () => {
  const { getByTestId, getByLabelText } = render(<App />);

  const EMAIL_USER = 'email@email.com';

  const textEmail = getByTestId('id-email-user');
  expect(textEmail).toBeInTheDocument();
  expect(textEmail.textContent).toBe('Valor: ');

  const btnSend = getByTestId('id-send');
  const inputEmail = getByLabelText('Email');
  fireEvent.change(inputEmail, { target: { value: EMAIL_USER } });
  fireEvent.click(btnSend);
  expect(inputEmail.value).toBe('');
  expect(textEmail.textContent).toBe(`Valor: ${EMAIL_USER}`);
});
```

FireEvent

Sintax: passar para ele o que quer escrever dentro do input durante o teste (pode se fazer um paralelo com, em JS, a relação entre addEventListener e event.target.value)

Segue lista completa de eventos suportados pelo FireEvent.

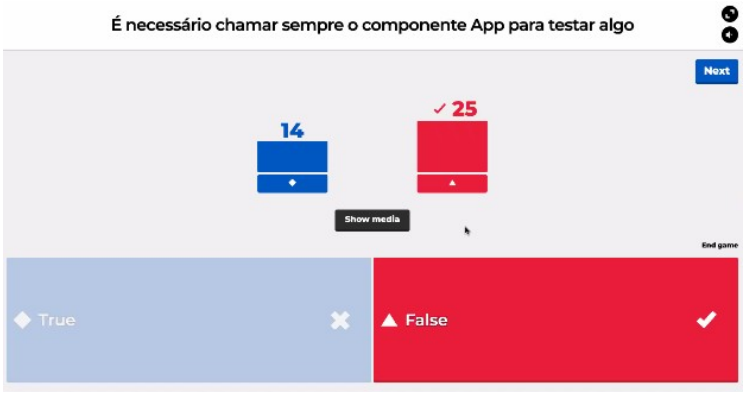
Testando apenas um componente

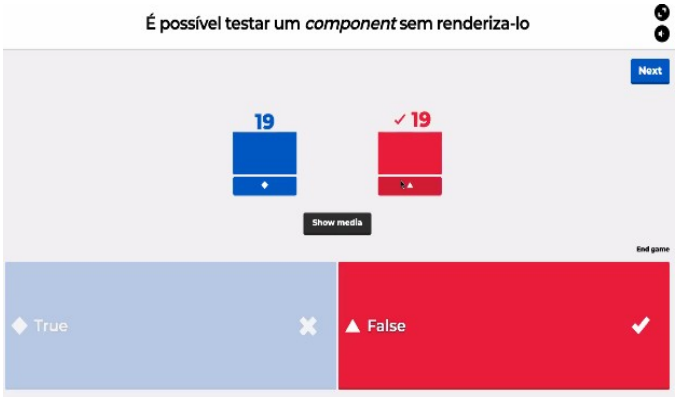
Estrutura de arquivos: component.js e component.test.js

Diferença com antes:
No lugar de render(<App />), *render(<SpecificComponent propsname={valorprops} />)* .
O componente que queremos renderizar precisa de uma props com valor para funcionar.

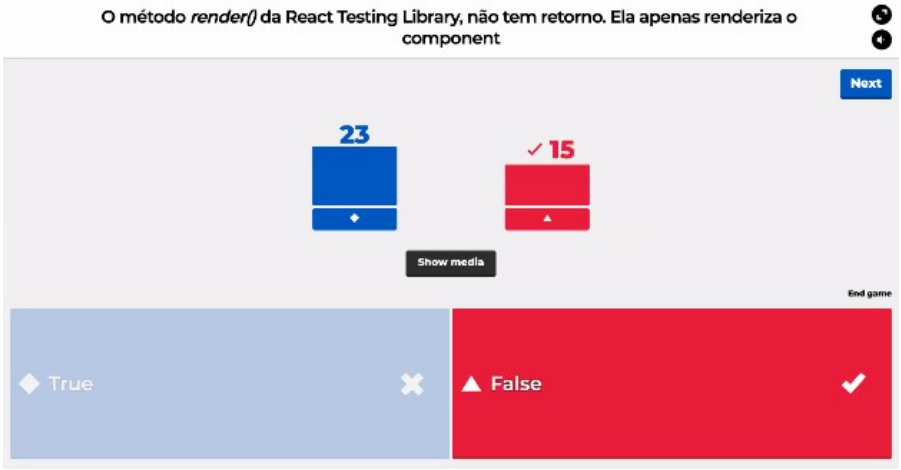
Dicas diversas

Kahoot

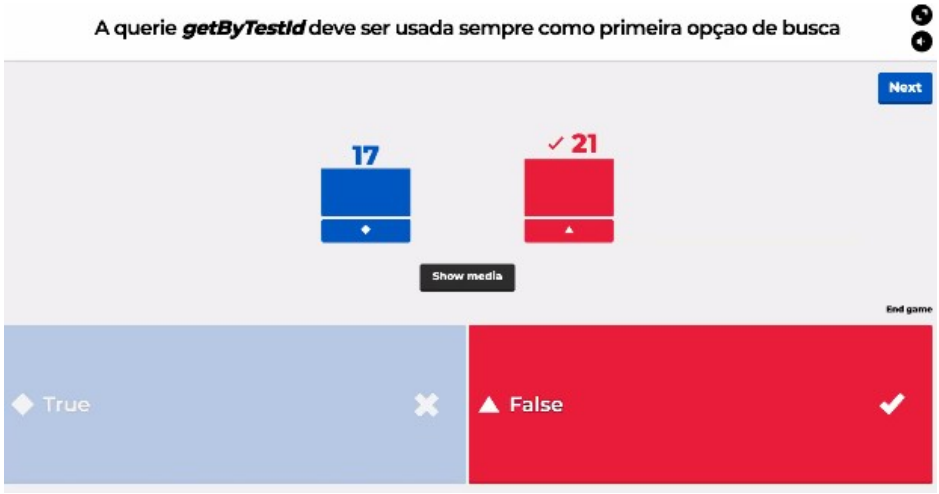




//Devemos chamar sempre um componente sim, mas nem sempre precisa ser o App.



//O método render() retorna um objetão de vários métodos, queries e outra série de elementos.



–

cleanup apaga todo dom depois de cada teste: testa comportamento, faz rodar, limpa para fazer outro teste. Usar entre cada teste.

```
afterEach(cleanup);
```

_

Exemplo prático de teste que não pode falhar:

```
describe('Some test', () => {
  test('Some test', () => {
    expect(true).toBeTruthy();
  })
});
```

_

Virando assincrono:

```
await waitForDomChange();
```

---

2) Testando React com RTL – parte II

Aprofundando com **mocks e rotas**.

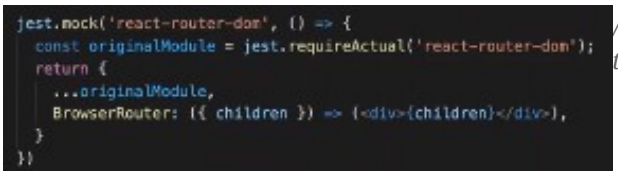
Mocking modules

Para evitar problemas de falta de controle sobre recursos externos como API.

```
// fetchJoke.test.js
const fetch = require('node-fetch');
const fetchJoke = require('./fetchJoke');

jest.mock('node-fetch');

it('should fetch users', () => {
  const joke = {
    id: '7h3oGtrOfxc',
    joke: 'Whiteboards ... are remarkable.',
    status: 200,
  };

  fetch.mockImplementation(() => Promise.resolve({
    json: () => Promise.resolve(joke),
  }));

  return fetchJoke()
    .then((data) => expect(data).toEqual('Whiteboards ... are remarkable.'));
});
```

```
jest.mock('react-router-dom', () => {
  return {
    BrowserRouter: ({ children }) => (<div>{children}</div>),
  }
})
```

//Mudar comportamento copiando toda a biblioteca router-dom para que o BrowserRouter funcione de outro jeito que originalmente, para poder testar ele.



```
jest.mock('react-router-dom', () => {
  const originalModule = jest.requireActual('react-router-dom');
  return {
    ...originalModule,
    BrowserRouter: ({ children }) => (<div>{children}</div>),
  }
})
```

// Também copiando todas opções do router dom para também dispor de Link etc.

Testando React Route

Referência sobre react router.

→ history

Para acessar a **sessão de *histórico* do browser e da URL.**

createBrowserHistory para navegadores atuais com HTML5;
createMemoryHistory para ambientes sem DOM.

```
it('deve renderizar o componente Sobre', () => {
  const { getByText, history } = renderWithRouter(<App />);
  fireEvent.click(getByText(/Sobre/i));
  const pathname = history.location.pathname;
  expect(pathname).toBe('/about');
  const aboutAll = getByText(/Você está na página Sobre/i);
  expect(aboutAll).toBeInTheDocument();
});
```

history.location.pathname para verificar se estamos na página correta.

```
test('navigating from home to projects', () => {
  const history = createMemoryHistory();

  const { getByText } = render(
    <Router history={history}>
      <App />
    </Router>
  );

  const personalInfo = getByText(/página sobre mim/i);
  expect(personalInfo).toBeInTheDocument();

  fireEvent.click(getByText(/projetos/i));

  const project = getByText(/página de projetos/i);
  expect(project).toBeInTheDocument();
});
```

Aqui, *usar o createMemoryHistory e o Router para envelopar o App dentro de um novo histórico* relevante para a realização dos testes desejados.

→ ***renderWithRouter*** function caseira neste repo para **fazer testes com rotas**.
Usa o createMemoryHistory para embutir no seu componente renderizado a lógica de histórico de navegação.

Sintax:

```
import React from 'react';
import { Router } from 'react-router-dom';
import { createMemoryHistory } from 'history';
import { render } from '@testing-library/react';

const renderWithRouter = (component) => {
  const history = createMemoryHistory();
  return {
    ...render(<Router history={history}>{component}</Router>), history,
  };
};

export default renderWithRouter;
```

Testando inputs em React

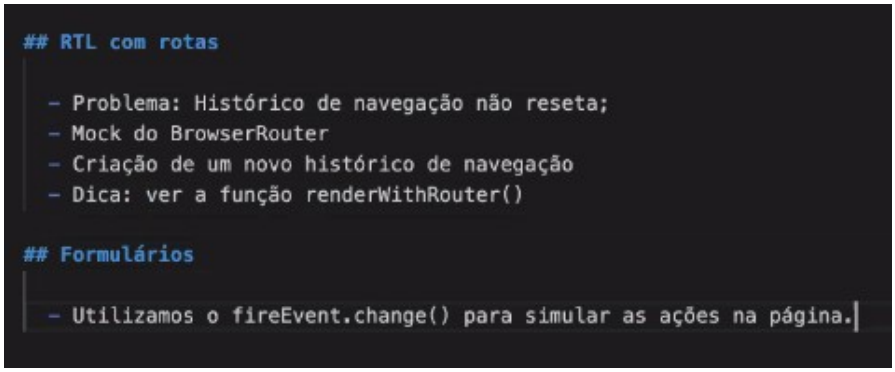
Para **teste de formulários: interagir com o input como o usuário** (pois valores dos inputs ainda não estão na tela no momento do carregamento).

→ Propriedade data-testid nos inputs:

```
 <input
//    onChange={(e) => this.handleInput(e)}
//    name='nome'
      data-testid='input-nome'
//    value={this.state.nome}
// />
```

→ fireEvent no evento change

Resumo do dia



Dicas diversas

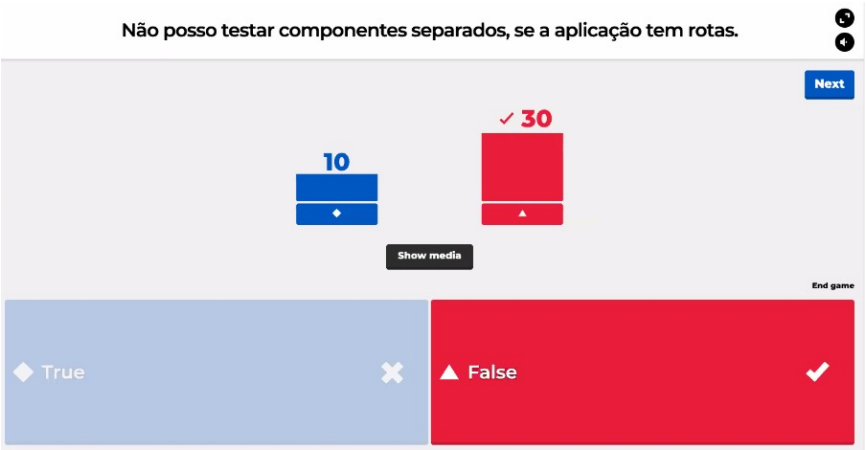
–

Sobre importar com ou sem {}

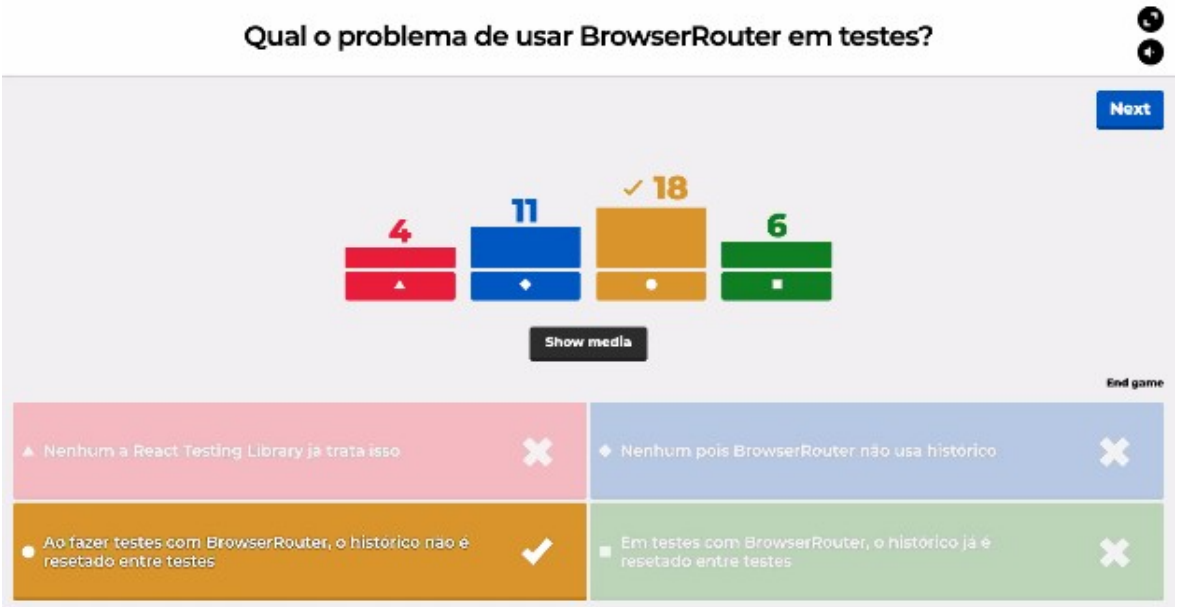
“Só pode haver um export default por arquivo (que faz o componente ser importável sem as chaves {}) e o App tomou esse espaço, então os outros componentes exportados ficam em "segundo plano". Por isso, para serem importados corretamente, necessitam do {}.”

–

Kahoot



//Quando faz teste por comportamento como um tudo, utiliza as rotas, mas não impede fazer a separação dos pedaços de código e componentes.



–

<u>Dica de projeto bloco 15:</u> pensar mais o percurso do user do que como desenvolvedor.