Bootcamp React 3.0 - Cap. 11

Testes e implantação

# Competências

* JEST e Testing Library
* Testes de funções JS/TS
  + Execução de testes, modo watch
  + Bloco describe e suíte de testes
  + Mock de funções com spyOn
* Testes de componentes React
  + Testes de unidade e de integração
  + Mock de funções com jest.fn()
  + Mock de requisições com MSW
  + Mock do React Router DOM
  + Simulação de interação do usuário
  + Fixtures
  + Inputs de formulário
  + Submissão de formulário
* Implantação com CI/CD
  + Netlify

# Usuários Windows: Terminal Linux com WSL

<https://github.com/devsuperior/wsl-win10>

Nome das aulas: “WSL …”

# 

# Recordando a base sobre testes

## Princípios de teste

* Padrão AAA
* Um teste para cada cenário
* Simples e linear (sem condicionais ou loops)
* Padrão de nome: <função> “should” <ação> when <cenário>
* Etc.

## O “santo graal” da maturidade do software

Quando um bug for descoberto, garanta que aquele bug nunca mais ocorrerá:

* Corrija o bug
* Escreva um teste que reproduza o bug

# 

# JEST & Testing Library

<https://jestjs.io>

<https://testing-library.com/docs/react-testing-library/intro/>

## JEST

* Test runner para JavaScript: executa testes na linha de comando (LINUX)
* É como se fosse um “navegador” do Node, que acessa a DOM

## Testing Library

* É uma biblioteca para testar UI, com foco no ponto de vista do usuário
* Core library + Frameworks

## Create React App

Um projeto React criado com o Create React App já vem com a infraestrutura de testes com JEST e Testing Library.

<https://create-react-app.dev/docs/running-tests>

# Setup do projeto

## Arquivo setupTests.ts

import '@testing-library/jest-dom';

## Atualizando as libs de teste

yarn add @testing-library/jest-dom @testing-library/react @testing-library/user-event

## Opção para não rodar no modo watch:

Package.json -> scripts:

"test": "react-scripts test --watchAll=false",

"test:watch": "react-scripts test",

# Testes de função JS/TS

## Primeiro teste: formatPrice

import { formatPrice } from '../formatters';

test('formatPrice should format number pt-BR when given 10.1', () => {

const result = formatPrice(10.1);

expect(result).toEqual('10,10');

});

## Bloco describe e suíte de testes

* Um arquivo <-> uma suíte de testes
* Um mesmo arquivo pode ter zero ou mais blocos “describe”

describe('descrição do bloco', () => {

beforeEach(() => {

...

});

afterEach(() => {

...

});

it | test('descrição do teste', () => {

...

});

it | test('descrição do teste', () => {

...

});

});

## 

## Exercício de fixação

**Projeto**: <https://github.com/devsuperior/bds09>

No projeto acima, a função *countClientsMatching(partialName: string)*, retorna quantos dos clientes possuem em seu nome o *partialName*, informado como argumento. Para isto, esta função faz a chamada de uma outra função *requestClients()*, responsável por obter os clientes cadastrados no sistema. Você deve escrever os testes para os dois cenários abaixo da função *countClientsMatching*, mockando a função *requestClients* para especificar cada um dos cenários no próprio teste.

*"should return 0 when 0 out of 2 clients match"*

*"should return 2 when 2 out of 4 clients match"*

**Solução**: <https://github.com/devsuperior/bds09/commit/bfb2f5d5928cf9ea664b9c746f0b45a47189d5c6>

# Testando componentes React

## Passos gerais

* Renderizar o HTML
* Encontrar o nó que queremos testar
* Verificar o conteúdo

## Mostrar no terminal o resultado da renderização

screen.debug();

## Queries da Testing Library para encontrar elementos

<https://testing-library.com/docs/queries/about>

# 

# Exercício de Fixação: renderização

**Solução**: próxima aula.

Escrever os testes de renderização dos componentes:

## ProductPrice

*'should render ProductPrice'*

### Asserções:

* O texto “R$” deve estar presente.
* O preço informado deve aparecer com duas casas decimais e vírgula como separador.

## ProductCard

*'should render ProductCard'*

### Asserções:

* O nome do produto informado deve estar presente.
* A imagem deve estar presente (dica: use getByAltText)
* O texto “R$” deve estar presente.
* O preço informado deve aparecer com duas casas decimais e vírgula como separador.

# 

# Testando páginas

## Catálogo

* Rotas
* Mock de requisições com MSW
* Fixtures

### Mock Service Worker: MSW

yarn add msw

### Ignorando arquivos fixtures.ts

"test": "react-scripts test --watchAll=false --testPathIgnorePatterns='fixtures.ts'",

"test:watch": "react-scripts test --testPathIgnorePatterns='fixtures.ts'",

## Formulário

* Inputs de formulário
* Submissão de formulário
* Mock do React Router DOM

### Cenários que vamos testar

* Suíte criação de produto
  + Deve mostrar o toast e redirecionar quando submetido corretamente
  + Deve mostrar 5 erros de validação quando apenas submeter o formulário
  + Deve limpar os erros de validação quando preencher o formulário corretamente
* Suíte edição de produto
  + Deve mostrar o toast e redirecionar quando submetido corretamente

### Selecionando um Select para testes

* Tem que ser por meio de um <label>
* A tag <label> se vincula a uma tag input por meio do atributo for="" em <label> e do atributo inputId="" na tag de input.

### Instalando o selectEvent para trabalhar com Select

yarn add react-select-event

# Implantação com CI/CD no Netlify

**ATENÇÃO:** veja material sobre **back end implantado** na mesma pasta desde documento

## Passos

* Base directory: frontweb
* Build command: yarn build
* Publish directory: frontweb/build
* Advanced -> Variables:
  + REACT\_APP\_BACKEND\_URL
  + REACT\_APP\_CLIENT\_ID
  + REACT\_APP\_CLIENT\_SECRET

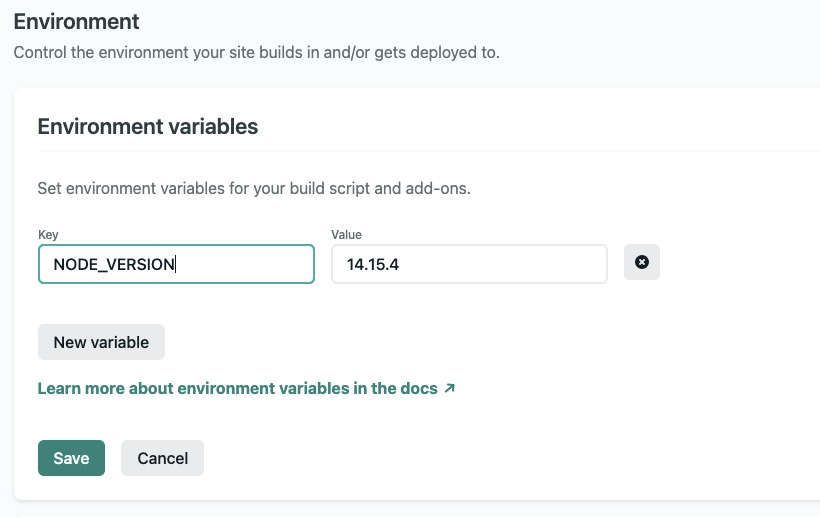
## Arquivo \_redirects na pasta public

/\* /index.html 200

## Customizando o nome do app

* Site settings -> Domain Management: (colocar o nome que você quiser)
* Deploys -> Trigger deploy

**ATENÇÃO:** pode acontecer um erro de versão do Node na implantação no Netlify. Se acontecer, configure uma variável de ambiente NODE\_VERSION como mostrado a seguir:



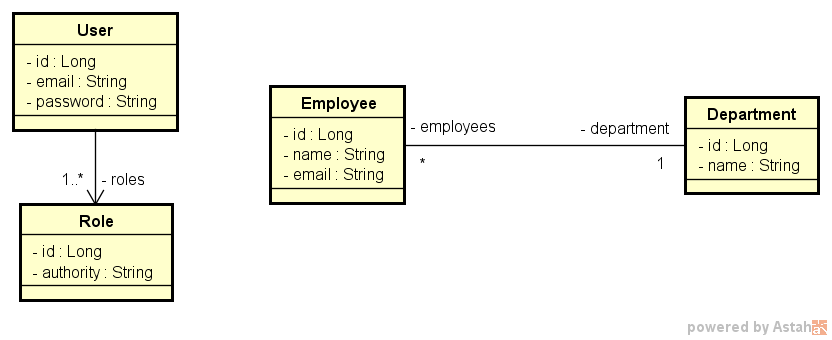
# Desafio para entregar

**TAREFA: ReactJS TDD**

Implemente as funcionalidades do projeto front end abaixo para que os testes passem:

<https://github.com/devsuperior/bds10>

Mínimo para aprovação: 6/8



Collection do Postman:

<https://www.getpostman.com/collections/bff250762d64873bd1dd>

DICAS:

* Assista atentamente os vídeos de apresentação do desafio. Eles contém orientações importantes.
* Para rodar apenas um teste específico, especifique o nome dele com a opção -t. Por exemplo:

yarn test -t 'should redirect when canceling'

* Todo layout e infraestrutura do projeto já estão prontos. Para resolver todo o desafio, os únicos arquivos que você vai ter que editar são:
  + pages/Admin/Employees/Form/index.tsx
  + pages/Admin/Employees/List/index.tsx
* Não precisa decorar nada! Este desafio presume a mesma lógica de CRUD do DSCatalog. Então fique com o DSCatalog em mão e vá aproveitando os códigos.