Módulo 13: Testes (aula 01)

Boas-vindas! Caro aspirante a dev!

Aula 01 – Introdução a testes:

Esta é nossa primeira aula do módulo de Testes, nela abordaremos os seguintes tópicos:

- O que são testes e sua importância
- Biblioteca Jest para testes em javascript
- Organização de testes com: test(), describe()
- Declarar expectativas com expect()
- Como decidir o que testar

Ao final desta aula, você estará apto(a), a:

- Instalar o Jest e configurá-lo
- Escrever testes unitários para unidades sem dependências, ou sem isolar as dependências caso existam
- Decidir quais comportamentos devem ser testados

Exercícios:

1. Crie um projeto em javascript (node.js) que tenha 1 arquivo de código principal e 1 arquivo de teste.

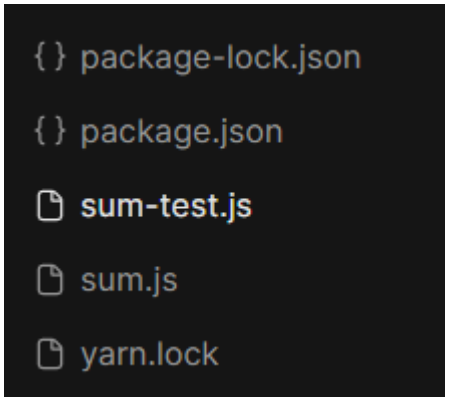
    a. O arquivo de código principal deve implementar corretamente e exportar uma função que calcula a soma entre dois números (parâmetros) e retorna essa soma.

    b. O arquivo de teste deve importar essa função e testá-la passando números para ela e verificando se o resultado está correto (com **expect**). Deve haver 3 testes: um com os números 20 e 10, outro com os números -10 e 10, e um terceiro com os números 2 e 2.

    c. Configure o **package.json** para rodar os testes com o comando **npm run test**

2. Partindo da sua solução ao exercício anterior, vamos introduzir um bug na função de soma: em vez de retornar a soma entre os parâmetros, retorne a multiplicação entre eles. Rode os testes, mostre o resultado, e explique qual(is) teste(s) passa(m) e por quê.

3. Mariazinha estava tentando realizar o exercício 01. Para isso, ela escreveu os seguintes arquivos:



Você pode assumir que ela implementou corretamente os códigos de todos os arquivos. Mas, ao executar **npm run test** (devidamente configurado no **package.json**), o terminal mostra o seguinte:

```
→ workspace git:(master) X npm run test

> templateforv2-static@1.0.0 test
> jest

No tests found, exiting with code 1
Run with `--passWithNoTests` to exit with code 0
In /project/home/megatron0000/workspace
  4 files checked.
  testMatch: **/__tests__/**/*.[jt]s?(x), **/?(*.)+(spec|test).[tj]s?(x) -
0 matches
  testPathIgnorePatterns: /node_modules/ - 4 matches
  testRegex:  - 0 matches
Pattern:  - 0 matches
```

Explique qual é o problema e por que ele aconteceu.

4. Joãozinho escreveu o seguinte código num arquivo **countLetters.js** (link):

```
function countLetters(input) {
  if (typeof input !== "string") {
    return { error: "Invalid input" };
  }

  let uppercaseCount = 0;
  let lowercaseCount = 0;

  for (let i = 0; i < input.length; i++) {
    if (input[i].toUpperCase() === input[i]) {
      uppercaseCount++;
    } else if (input[i].toLowerCase() === input[i]) {
      lowercaseCount++;
    }
  }

  return {
    uppercase: uppercaseCount,
    lowercase: lowercaseCount,
  };
}

module.exports = countLetters;
```

Essa função recebe uma string e calcula quantos caracteres maiúsculos e minúsculos há na string. Joãozinho então escreveu testes para essa função, num arquivo separado (link):

```
const countLetters = require("./countLetters");

// Testes para entradas inválidas
test("Retorna objeto com erro quando a entrada não é uma string", () => {
  expect(countLetters(123)).toEqual({ error: "Invalid input" });
});

// Testes para casos base
test("Retorna contagens em 0 quando a entrada é  uma string vazia", () => {
  expect(countLetters("")).toEqual({ uppercase: 0, lowercase: 0 });
});

// Testes para casos gerais
test("Conta corretamente letras maiúsculas e minúsculas", () => {
  expect(countLetters("AbCdEf")).toEqual({ uppercase: 3, lowercase: 3 });
});

test("Conta corretamente quando a entrada é só letras maiúsculas", () => {
  expect(countLetters("ABC")).toEqual({ uppercase: 3, lowercase: 0 });
```

```
});

test("Conta corretamente quando a entrada é só letras minúsculas", () => {
  expect(countLetters("abc")).toEqual({ uppercase: 0, lowercase: 3 });
});

test("Conta corretamente quando a entrada tem caracteres não-letra", () => {
  expect(countLetters("A1b2C3")).toEqual({ uppercase: 3, lowercase: 2 });
});
```

Os 3 comentários de Joãozinho indicam que esses testes são naturalmente agrupados em 3 grupos: entradas inválidas, casos-base e casos gerais. Use **describe()** para agrupar os testes nesses 3 grupos conforme os comentários do código. Entregue o projeto funcional contendo o arquivo **countLetters.js** sem alterações e a nova versão do arquivo de teste. Em seu projeto, os testes devem rodar com **npm run test**.

5. Assista ao vídeo disponível na referência *Uma ÚNICA Coisa Me Faz Programar "10x" Mais Rápido*. Explique quais são os argumentos do Felipe em prol do uso de testes automatizados.

6. Você foi encarregado(a) de testar uma função que faz parte de uma API de locação de filmes.

   A função é chamada **availableMovies** ("filmes disponíveis"), ela recebe:

   - um parâmetro **movies** que é um array de filmes, onde cada filme é um objeto **{title, minAge}** onde **title** é o nome do filme e **minAge** é a idade mínima recomendável para o filme. A função pode assumir que esse array é válido: tem 0 ou mais filmes, e as idades são números inteiros maiores ou iguais a 0.
   - um parâmetro **age** que representa a idade do usuário. A função pode assumir que essa idade é válida: é um número inteiro maior ou igual a 1.

   A função retorna uma outra lista filtrada de filmes, contendo somente os filmes de **movies** que o usuário tem idade suficiente para assistir. Por exemplo, se um filme tem **minAge** de 12 e o usuário tem 12 anos ou mais, ele pode assistir a esse filme. A implementação que você recebeu para testar é a seguinte (link), ela **não necessariamente está totalmente correta**:

```
function availableMovies(movies, age) {
  const allowedMovies = movies.filter((movie) => {
    return movie.minAge < age;
  });

  return allowedMovies;
}
module.exports = availableMovies;
```

Crie um projeto (node.js) com 2 arquivos: um contendo essa função (não altere a implementação dada) e outro para escrever testes da função. Seu objetivo é pensar: quais cenários devem ser testados ? quais são os “edge cases” ? sugestão: leia a referência “*Uma história de testes com Jest (básico)”*. Em seguida, implemente os testes que você avaliou que fazem sentido. Entregue um projeto javascript contendo a função **availableMovies** num arquivo, seus testes em outro, e um README explicando quais cenários de teste você decidiu que são suficientes e por quê, e também explicando se a implementação fornecida da função **availableMovies** está correta ou não (baseado nos resultados dos testes). Os testes devem rodar com **npm run test**.

Referências:

- ➔ O que são Testes Unitários e como implementar em JavaScript? (DevPleno no youtube): https://www.youtube.com/watch?v=MQs8_KIj_PU
  - ◆ Não precisamos agora de **beforeAll, beforeEach, afterAll, afterEach**
  - ◆ Ele mostra como usar a ferramenta Jest e também fala sobre boas práticas de teste (um tipo particular de teste, chamado “teste unitário”)
- ➔ Tutorial Teste Unitário com Jest (Dev Tech no youtube): https://www.youtube.com/watch?v=A6ODQSxy3Co
- ➔ Teste unitário com Jest (devmedia): https://www.devmedia.com.br/teste-unitario-com-jest/41234
  - ◆ Não precisamos das seções “Funções de Mock” em diante
- ➔ TDD: Como criar unit tests em Node.js com Jest (luiztools): https://www.luiztools.com.br/post/tdd-como-criar-unit-tests-em-node-js-com-jest/
  - ◆ Mostra como gera o arquivo de configuração do Jest (comando **jest –init**)
- ➔ Testando seu Javascript com Jest! (Medium): https://medium.com/jaguaribetech/testando-seu-javascript-com-jest-de2a4674f4ad

- Mostra o uso da função **describe**. Fala também sobre **skip** (curiosidade)

➔ Documentação do Jest: Iniciando: https://jestjs.io/pt-BR/docs/getting-started

- Mostra configurações para usar Jest em outros ambientes, como webpack, Vite, Parcel e typescript

➔ Documentação do Jest: Usando Matchers: https://jestjs.io/pt-BR/docs/using-matchers

- Mostra maneiras comuns de usar o **expect**

➔ Documentação do Jest: Testando Código Assíncrono: https://jestjs.io/pt-BR/docs/asynchronous

- Mostra como testar códigos que usam **Promise** ou **async/await**

➔ Documentação do Jest: Configuração e Desmontagem: https://jestjs.io/pt-BR/docs/setup-teardown

- Não é sobre o arquivo de configuração jest.config.js ! É sobre comandos para configurar e desmontar testes (**beforeEach, beforeAll, afterEach, afterAll**)

➔ Documentação do Jest: como usar o Jest em módulos ES6: https://jestjs.io/pt-BR/docs/ecmascript-modules

➔ Uma ÚNICA Coisa Me Faz Programar “10x” Mais Rápido (De Verdade) (Felipe Deschamps no youtube): https://www.youtube.com/watch?v=LVxhiiE4nTE

➔ Uma história de testes com Jest (básico) (Notion): https://alluring-beluga-3dd.notion.site/Uma-hist-ria-de-testes-com-Jest-b-sico-6ef0d86483fb4514851f1397c4a6b3cb