Boas-vindas! Caro aspirante a dev!
Na primeira aula falamos sobre testes unitários, aprendemos a instalar a biblioteca Jest e criamos nossos primeiros casos de testes. Nesta aula, vamos nos aprofundar um pouco mais com múltiplos casos de testes e testes de integração. E aí, animado?

Aula 02 - Tipos de testes :
Esta é nossa segunda aula do módulo Testes, nela abordaremos os seguintes tópicos:

- Tipos de testes
- Testes para cenários esperados usando describe()
- Testes de integração
- *Exercícios teóricos e práticos*

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

- Conhecer os diferentes tipos de testes existentes;
- Escrever testes unitários para diferentes cenários; e
- Entender de forma específica o que é e como realizar testes de integração.

Exercícios:

1. Com suas palavras, liste os principais tipos de testes que podem ser realizados em um projeto de software. Obs.: Não precisa listar todos, só os mais comuns.
2. É possível realizarmos testes antes de termos o código principal? Se a resposta for sim, de qual conceito estamos falando e como fazer para tal? Explique.
3. Para múltiplos casos de testes, em que se espera diferentes resultados, sejam para ações de sucesso ou de erros, é possível tornar a ordem dos testes aleatória? Deste modo, quais os benefícios em se fazer tal ajuste, bem como desafios a serem ajustados? Explique.
4. Qual a importância dos testes de integração e o que de fato é testado com eles? Explique!
5. Configure a opção de cobertura de testes do jest.
6. Dada as duas funções abaixo, crie usando describe(), múltiplos casos de testes que abranjam resultados esperados para as funções.

```
function isEven(number) {
  return number % 2 === 0;
}

function getPositiveNumbers(array) {
  return array.filter((num) => num > 0);
}
```

No exemplo acima temos duas funções: *isEven*, que verifica se um número é par, e *getPositiveNumbers*, que retorna apenas os números positivos de um array. Você deve

escrever os testes para essas funções em um arquivo específico usando a função describe() para agrupar os casos de teste relacionados.
No bloco describe('isEven', ...), temos dois casos de teste: um que verifica se a função retorna true para números pares e outro que verifica se a função retorna false para números ímpares.
No bloco describe('getPositiveNumbers', ...), temos três casos de teste: um que verifica se a função retorna um array com apenas os números positivos corretamente, outro que verifica se a função retorna um array vazio quando nenhum número positivo é encontrado e um terceiro que verifica se a função retorna o mesmo array quando todos os números são positivos.
Você poderá adicionar mais casos de teste, explorando diferentes cenários e entradas possíveis para cada função. O objetivo é cobrir o máximo de casos possível e garantir que as funções se comportem conforme o esperado.

7. Dado o exemplo abaixo:

```
// api.js
const fruits = ['apple', 'banana', 'orange'];

function getFruitById(id) {
  if (id >= 0 && id < fruits.length) {
    return { status: 200, data: fruits[id] };
  } else {
    return { status: 404, data: 'Fruit not found' };
  }
}

function addFruit(fruit) {
  if (typeof fruit !== 'string') {
    return { status: 400, data: 'Invalid fruit' };
  }

  fruits.push(fruit);
  return { status: 201, data: 'Fruit added successfully' };
}
```

Neste exercício, temos um arquivo api.js que contém a função getFruitById. Essa função recebe um ID como parâmetro e retorna um objeto contendo o status da resposta (200 para sucesso ou 404 para erro) e o dado correspondente àquele ID no array de frutas.

Crie um arquivo de teste com a função describe para agrupar os casos de teste relacionados à função getFruitById. Dentro do bloco describe, temos alguns casos de teste.
No primeiro caso de teste, deveremos verificar se a função retorna corretamente a fruta esperada para IDs válidos (0, 1 e 2), com o status 200.
No segundo caso de teste, deveremos verificar se a função retorna o status 404 e a mensagem "Fruit not found" para IDs inválidos (-1, 3 e 100), ou seja, IDs que estão fora do intervalo do array de frutas.
Mais abaixo temos a função de adicionar frutas à API, addFruit. Essa função recebe uma fruta como parâmetro, a adiciona ao array de frutas e retorna um objeto contendo o status da resposta (201 para sucesso ou 400 para erro) e uma mensagem correspondente.
Os testes são agrupados utilizando a função describe. No bloco describe('addFruit', ...), deverá ser utilizado o beforeEach para redefinir o array de frutas antes de cada teste, garantindo um estado limpo e consistente para cada caso de teste.
No primeiro caso de teste, você deve verificar se a função addFruit adiciona corretamente uma nova fruta ao array, retornando o status 201 e a mensagem de sucesso. Também deve verificar se a nova fruta foi adicionada corretamente ao array e se o tamanho do array aumentou corretamente.
No segundo caso de teste, verifique se a função addFruit retorna o status 400 e a mensagem de "Invalid fruit" quando é passado um parâmetro que não é uma string. Também garantimos que o array de frutas não tenha sido alterado nesse caso.

8. Utilizando os conceitos aprendidos, crie um teste de integração para o seguinte cenário: Um sistema academia tem dois módulos dependentes, um que permite ao atendente cadastrar num objeto os dados pessoais de um aluno, tais como: nome, idade, peso e altura. E o outro, permite calcular o imc do aluno e informar, de acordo com a tabela abaixo, qual a opção de treino ele deverá realizar numa esteira de corrida.

Faixa Etária	Faixa de IMC	Treino na Esteira
18-30	< 18,5	30 min
-	18,5 - 24,9	20 min
31-45	< 18,5	25 min
-	18,5 - 24,9	15 min

Crie os módulos separadamente.

Em síntese:

Ao final desta aula você saberá como realizar múltiplos testes para diferentes cenários esperados, bem como criar testes de integração.

Lembre-se que o foco deste estudo é sempre aprofundar o seu conhecimento em web e, portanto, não se limite às referências deste módulo e mergulhe no conhecimento sobre Testes, pertinentes a esta aula.

Referências:

- ➔ Os diferentes tipos de testes em software | Atlassian
- ➔ Globais · Jest
- ➔ Opções CLI do Jest
- ➔ Integration Testing: What is, Types with Example