Roteiro de Atividade Prática

Nome: Turma: .

**Atividade: Escrevendo testes unitários com Jest**

Você foi contratado para garantir a qualidade de uma aplicação de   
e-commerce, que lida com pedidos e inventários de produtos. Seu objetivo é configurar um ambiente de testes unitários utilizando **Jest**, escrever testes para funções essenciais e utilizar **Mocking** para simular comportamentos externos, como chamadas de API.

Situação fictícia produzida pela SEDUC-SP.

**Passo 1 – Configurar o projeto.**

Começaremos configurando o ambiente de desenvolvimento com **Node.js** e instalando as dependências necessárias.

**1.1. Inicializar o projeto Node.js:**

Crie uma pasta para o projeto e inicialize o Node.js:

mkdir shopmaster

cd shopmaster

npm init -y

**1.2. Instalar dependências:**

Instale as dependências essenciais, incluindo o **axios** (para as requisições HTTP) e o **Jest** (para testes unitários):

npm install axios

npm install --save-dev jest

**1.3. Configurar script de teste:**

No arquivo package.json, adicione um script para rodar o Jest:

"scripts": {

"test": "jest"

}

**Passo 2 – Implementar funcionalidades da aplicação.**

Agora, implementaremos a lógica principal da aplicação: calcular o total do pedido, aplicar desconto e fazer uma chamada para uma API externa de câmbio.

**2.1. Implementar o cálculo do total do pedido e desconto:**

Crie um arquivo chamado pedidos.js no qual você definirá essas funções:

// pedidos.js

function calcularTotalPedido(produtos) {

return produtos.reduce((total, produto) => total + produto.preco, 0);

}

function aplicarDesconto(total, desconto) {

if (desconto > total) {

throw new Error('Desconto maior que o total');

}

return total - desconto;

}

module.exports = { calcularTotalPedido, aplicarDesconto }.

* **Função calcularTotalPedido**: soma os preços dos produtos no carrinho.
* **Função aplicarDesconto**: aplica um desconto ao total, garantindo que este não seja maior que o valor total.

**2.2. Implementar a função para calcular com taxa de câmbio:**

Crie um arquivo cambio.js para definir a função que faz a chamada à API de câmbio:

// cambio.js

const axios = require('axios');

async function calcularTotalComTaxaDeCambio(produtos, moeda) {

const total = produtos.reduce((total, produto) => total + produto.preco, 0);

const resposta = await axios.get(`https://api.exchangerate-api.com/${moeda}`);

const taxaDeCambio = resposta.data.taxa;

return total \* taxaDeCambio;

}

module.exports = { calcularTotalComTaxaDeCambio }.

* **Função calcularTotalComTaxaDeCambio**: calcula o total do pedido em uma moeda diferente, utilizando uma API de câmbio para obter a taxa de conversão.

**Passo 3 – Escrever os testes com Jest.**

Agora, escreveremos os testes para garantir que nossas funções funcionem corretamente.

**3.1. Testar o cálculo do total do pedido e a aplicação de desconto:**

Crie um arquivo de teste chamado pedidos.test.js:

// pedidos.test.js

const { calcularTotalPedido, aplicarDesconto } = require('./pedidos');

describe('Função calcularTotalPedido', () => {

test('Deve calcular o total corretamente para uma lista de produtos', () => {

const produtos = [

{ nome: 'Produto A', preco: 100 },

{ nome: 'Produto B', preco: 200 },

];

const total = calcularTotalPedido(produtos);

expect(total).toBe(300);

});

test('Deve retornar 0 para uma lista vazia de produtos', () => {

const total = calcularTotalPedido([]);

expect(total).toBe(0);

});

});

describe('Função aplicarDesconto', () => {

test('Deve aplicar o desconto corretamente', () => {

const total = aplicarDesconto(300, 50);

expect(total).toBe(250);

});

test('Deve lançar erro se o desconto for maior que o total', () => {

expect(() => aplicarDesconto(100, 200)).toThrow('Desconto maior que o total');

});

}).

* **Testes para calcularTotalPedido**: testamos se o cálculo do total funciona tanto para uma lista de produtos quanto para uma lista vazia.
* **Testes para aplicarDesconto**: testamos se o desconto é aplicado corretamente e se a função lança erro quando o desconto é maior que o total.

**3.2. Testar a função que usa a API de câmbio (com Mocking):**

Crie um arquivo de teste chamado cambio.test.js:

// cambio.test.js

const { calcularTotalComTaxaDeCambio } = require('./cambio');

const axios = require('axios');

// Mock da biblioteca axios

jest.mock('axios');

describe('Função calcularTotalComTaxaDeCambio', () => {

test('Deve calcular o total em outra moeda utilizando a taxa de câmbio simulada', async () => {

const produtos = [

{ nome: 'Produto A', preco: 100 },

{ nome: 'Produto B', preco: 200 },

];

// Simula a resposta da API de taxa de câmbio

axios.get.mockResolvedValue({ data: { taxa: 5 } });

const totalConvertido = await calcularTotalComTaxaDeCambio(produtos, 'USD');

expect(totalConvertido).toBe(1500); // (100 + 200) \* 5

});

}).

* **Mock da API**: neste caso, usamos jest.mock('axios') para simular a resposta da API de câmbio. A função mockResolvedValue simula a taxa de câmbio de 5, permitindo que o teste seja executado sem depender da API real.

**Passo 4 – Executar os testes.**

Agora que os testes estão configurados, você pode executá-los com o comando:

npm test

O Jest rodará os testes e mostrará os resultados no terminal, indicando se todas as funções estão funcionando corretamente e se os casos de teste passaram.

**Passo 5 – Analisar os resultados.**

* O Jest exibirá um relatório com os resultados dos testes no terminal. Se algum teste falhar, o Jest também mostrará uma mensagem detalhada, explicando o que deu errado.
* Se tudo estiver correto, o resultado será semelhante a:

PASS ./pedidos.test.js

PASS ./cambio.test.js

Test Suites: 2 passed, 2 total

Tests: 5 passed, 5 total

Snapshots: 0 total

Time: 0.949 s

Ran all test suites.

**Próximos passos**

* Você pode expandir a aplicação, adicionando mais funcionalidades, como validação de inventário, cálculo de frete, ou simulação de diferentes tipos de descontos.
* Continuar escrevendo testes para essas novas funcionalidades, garantindo que cada aspecto da aplicação seja coberto.

|  |
| --- |
|  |
|  |
| PASS ./cambio.test.js  PASS ./pedidos.test.js  Test Suites: 2 passed, 2 total  Tests: 5 passed, 5 total  Snapshots: 0 total  Time: 0.685 s, estimated 1 s |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |