RESUMO DA AULA

Aprofundando nos testes de unidade

Nesta se o, vamos nos aprofundar nos testes de unidade, cobrindo todos os poss veis cen rios de uma fun o e garantindo que ela se comporte conforme esperado em cada um deles. Vamos seguir um processo sistem tico para identificar, implementar e validar cada fluxo l gico presente na fun o que estamos testando. Entendendo o cen rio da fun o a ser testada Vamos trabalhar com a fun o calcularValorPedido, que se comporta de formas diferentes dependendo do valor total dos itens no pedido: Se o valor total dos produtos for maior que 500 reais, o frete ser gratuito. Se o valor for menor que 500 reais, o frete ser cobrado. Se o valor for exatamente 500 reais, o frete tamb m ser cobrado (considerado o limite inferior). Estrutura b sica dos testes Antes de mais nada, precisamos importar a fun o a ser testada no nosso arquivo de testes para garantir que o Jest tenha acesso a ela: const calcularValorPedido = require('./calcular-valor-pedido'); A importa o necess ria para que possamos usar a fun o calcularValorPedido nos nossos testes e verificar se os resultados retornados est o corretos. Criando o primeiro teste para um cen rio espec fico Vamos come ar escrevendo um teste b sico para o cen rio onde o valor do pedido maior que 500 reais e o frete deve ser gratuito. Exemplo 1: Cen rio em que o valor superior a 500 reais it('N o deve cobrar frete caso valor do pedido seja maior que 500 reais', () => { // Arrange - Prepara o: Criar o pedido de teste const pedido = { itens: [{ nome: 'Arco encantado', valor: 1000 }, { nome: 'Entrega', valor: 100, entrega: true },] }; // Act - A o: Executar a fun o com o pedido preparado const totalPedido = calcularValorPedido(pedido); // Assert - Verifica o: Validar que o total n o inclui o valor da entrega expect(totalPedido).toBe(1000); }); Aqui, n s preparamos o pedido e verificamos que o valor total retornado n o inclui o valor da entrega, pois o valor total dos itens excede 500 reais. Entendendo o conceito de "Arrange, Act e Assert" Nos testes, seguimos a estrutura conhecida como Arrange, Act e Assert: Arrange: Prepara os dados e o ambiente para o teste. Act: Executa a a o que desejamos testar (a chamada da fun o, por exemplo). Assert: Verifica se o

resultado da a o o esperado. Seguindo essa estrutura, fica mais f cil entender e manter nossos testes organizados e leg veis. Criando mais testes para cobrir diferentes cen rios Vamos agora testar outros cen rios para garantir que a nossa fun o se comporte corretamente em diferentes situa es. Exemplo 2: Cen rio em que o valor menor que 500 reais e o frete deve ser cobrado it('Deve cobrar frete caso valor do pedido seja menor que 500 reais', () => { // Arrange - Prepara o: Criar o pedido de teste const pedido = { itens: [{ nome: 'Sandu che', valor: 50 }, { nome: 'Bota nova', valor: 400 }, { nome: 'Entrega', valor: 100, entrega: true }] }; // Act - A o: Executar a fun o com o pedido preparado const totalPedido = calcularValorPedido(pedido); // Assert - Verifica o: Validar que o total inclui o valor da entrega expect(totalPedido).toBe(550); }); Nesse exemplo, como o valor dos itens menor que 500 reais, o valor total deve ser 550 (soma dos produtos mais o frete). Exemplo 3: Cen rio em que o valor dos produtos exatamente 500 reais Esse cen rio interessante porque um limite e muitas vezes os limites podem causar erros em fun es mal implementadas. it('Deve cobrar frete caso valor do pedido seja exatamente 500 reais', () => { // Arrange - Prepara o: Criar o pedido de teste const pedido = { itens: [{ nome: 'Sandu che', valor: 100 }, { nome: 'Bota nova', valor: 400 }, { nome: 'Entrega', valor: 100, entrega: true }] }; // Act - A o: Executar a fun o com o pedido preparado const totalPedido = calcularValorPedido(pedido); // Assert - Verifica o: Validar que o total inclui o valor da entrega expect(totalPedido).toBe(600); }); Aqui, queremos garantir que mesmo quando o valor exatamente 500, o frete seja cobrado, e o valor total seja 600 reais. Executando os testes Para rodar os testes, basta utilizar o comando: npm test Voc deve ver o Jest rodando e exibindo o resultado de cada teste. Se todos passarem, significa que nossa fun o est cobrindo corretamente todos os cen rios especificados. Feedback r pido com watch Se quisermos ver os resultados dos testes automaticamente sempre que alterarmos o c digo, podemos rodar o Jest no modo de observa o: npm run watch Dessa forma, qualquer modifica o na fun o calcularValorPedido ou nos testes ir disparar a execu o autom tica dos testes, mostrando rapidamente se uma altera o que fizemos quebrou algum cen rio. Resumo Nesta aula, criamos testes de unidade para diferentes cen rios da fun o calcularValorPedido, cobrindo tanto o comportamento esperado quanto limites de valor. Agora

temos confian a de que a fun o se comporta corretamente para diferentes entradas de dados. Testes de unidade nos ajudam a garantir que cada parte do nosso sistema est funcionando como esperado, sem precisar rodar manualmente cada parte do c digo. Na pr xima se o, vamos ver como estruturar testes ainda mais detalhados e entender como criar descri es de testes claras e f ceis de entender.
