# RESUMO DA AULA

## Criando nosso primeiro teste automatizado com Jest

---

Nesta se o, vamos aprender como criar o nosso primeiro teste automatizado usando a API do Jest. Tamb m vamos explorar a estrutura b sica de um teste e como configurar o Jest para rodar automaticamente sempre que houver mudan as no c digo. Estrutura b sica de um teste com Jest Antes de come ar a criar testes complexos, vamos entender como estruturar um teste b sico usando Jest. Os testes em Jest s o constru dos usando fun es como test() ou it(). Vamos ver como criar um teste simples e entender cada parte. Criando um teste b sico Para come ar, crie um arquivo chamado primeiro-teste.test.js na pasta do seu projeto. Dentro desse arquivo, adicione o seguinte c digo: it('Verifica se o teste est  funcionando', () => {  expect(true).toBe(true); }); Aqui: it(): uma fun o fornecida pelo Jest para definir um teste. Alternativamente, voc  pode usar test(), pois ambas t m a mesma funcionalidade. 'Verifica se o teste est  funcionando': a descri o do teste. Esse texto ajuda a identificar qual o objetivo do teste quando o resultado for exibido. expect(true).toBe(true);: Essa linha a "assertiva" do teste. Estamos dizendo que esperamos que true seja igual a true. Caso seja, o teste ser  considerado como "passou"; caso contr rio, ser  "falhou". Executando o teste Para rodar o teste, basta executar o comando: npm test Voc  ver  no console um retorno indicando se o teste foi bem-sucedido ou se houve falhas. Como o nosso teste simples e sempre espera que true seja igual a true, ele deve passar. Modificando o teste para falhar: Se alterarmos o c digo para: it('Verifica se o teste falha corretamente', () => { expect(true).toBe(false); }); E rodarmos npm test novamente, o Jest retornar  um erro dizendo que o teste falhou, mostrando uma mensagem explicando que true n o  igual a false. Isso nos ajuda a ver como o Jest lida com falhas e onde precisamos corrigir. Estrutura completa de um teste com Jest Um teste completo com Jest  geralmente estruturado assim: it('Descri o do teste', () => {  // 1. Setup: Preparar o ambiente ou dados para o teste  const valor1 = 10;  const valor2 = 20;  // 2. Execu o: Realizar a a o que queremos testar  const resultado = valor1 + valor2;  // 3. Verifica o:

Validar se o resultado  o esperado  expect(resultado).toBe(30);  // 4. Cleanup (opcional): Limpar o ambiente caso necess rio }); Aqui: Setup: Preparamos vari veis e ambientes necess rios para realizar o teste. Execu o: Executamos a fun o ou a o que queremos testar. Verifica o (assertiva): Usamos expect() para validar o resultado. Cleanup: Limpar qualquer estado que tenha sido alterado pelo teste (nem sempre necess rio). Rodando testes automaticamente Uma das vantagens de usar o Jest  que ele pode monitorar automaticamente mudan as no c digo e rodar os testes sempre que detecta uma altera o. Vamos configurar isso. Abra o arquivo package.json e adicione um novo script para o Jest rodar em modo de "watch" (observa o cont nua): json "scripts": {  "test": "jest",  "watch": "jest --watchAll" } jest --watchAll: Esse comando faz com que o Jest fique "escutando" por mudan as em todos os arquivos .js e .test.js e rode os testes automaticamente sempre que houver qualquer altera o. Agora, no terminal, voc  pode rodar: npm run watch O Jest vai entrar no modo de observa o e qualquer mudan a no c digo ir  rodar os testes automaticamente. Isso  muito  til quando estamos desenvolvendo, pois permite um feedback imediato sobre a qualidade do nosso c digo. Modificando testes e vendo o feedback em tempo real Vamos voltar ao nosso arquivo primeiro-teste.test.js e modificar o teste para ver como o Jest reage em tempo real. Altere o conte do do teste para: it('Verifica se a soma de 2 + 2  igual a 4', () => {  const soma = 2 + 2; expect(soma).toBe(4); }); Salve o arquivo e observe o terminal. Se o Jest estiver rodando em modo watch, voc  ver  que ele executou o teste novamente e confirmou que 2 + 2  igual a 4. Agora, altere o c digo para: it('Verifica se a soma de 2 + 2  igual a 5 (teste falho)', () => {  const soma = 2 + 2; expect(soma).toBe(5); }); Salve novamente. Desta vez, o Jest vai indicar que o teste falhou e mostrar  o valor esperado (5) e o valor recebido (4). Por que rodar testes continuamente? Esse modo cont nuo de testes  muito  til, pois: Feedback r pido: Mostra imediatamente se uma mudan a no c digo quebrou algum teste. Desenvolvimento orientado a testes: Permite que desenvolvedores criem testes primeiro e depois escrevam o c digo para passar esses testes. Identifica o r pida de regress es: Ajuda a identificar rapidamente quando uma altera o introduz um erro em outra parte do sistema. Conclus o Nesta aula, aprendemos a estrutura b sica de um teste usando o Jest e como configurar o Jest para rodar automaticamente em modo de observa o. Tamb m vimos como os

testes s o constru dos com it() e expect(), e como configurar scripts no package.json para facilitar o desenvolvimento cont nuo.

---